6 Interpolation and Approximation

6.0 Introduction

In this chapter we will discuss the problem of fitting data given in the form of discrete points (e.g., physical measurements, output from a differential equations solver, design points for CAD, etc.) with an appropriate function s taken from some (finite-dimensional) function space $\mathcal{S}$ of our choice. Throughout most of the chapter the data will be *univariate*, i.e., of the form (x_i, y_i), $i = 0, \ldots, n$, where the $x_i \in \mathbb{R}$ are referred to as *data sites* (or *nodes*), and the $y_i \in \mathbb{R}$ as *values* (or *ordinates*). Later on we will also consider *multivariate* data where $x_i \in \mathbb{R}^d$, $d > 1$.

The data can be fitted either by *interpolation*, i.e., by satisfying

$$s(x_i) = y_i, \qquad i = 0, \ldots, n, \tag{1}$$

or by *approximation*, i.e., by satisfying

$$\|s - y\| < \epsilon,$$

where s and y have to be considered as vectors of function or data values, and $\|\cdot\|$ is some discrete norm on $\mathbb{R}^{n+1}$.

We will focus our attention on interpolation as well as on least squares approximation.

If $\{b_0, \ldots, b_m\}$ is some basis of the function space $\mathcal{S}$, then we can express s as a linear combination in the form

$$s(x) = \sum_{j=0}^{m} a_j b_j(x).$$

The interpolation conditions (1) then lead to

$$\sum_{j=0}^{m} a_j b_j(x_i) = y_i, \qquad i = 0, \ldots, n.$$

This represents a linear system for the expansion coefficients a_j of the form

$$Ba = y,$$

where the $n \times m$ system matrix B has entries $b_j(x_i)$. We are especially interested in those function spaces and bases for which the case $m = n$ yields a unique solution.

Remark: Note that for least squares approximation we will want $m \ll n$. For certain situations, e.g., when we wish to satisfy certain additional shape constraints (such as monotonicity or convexity), we will want to choose $m > n$, and use the non-uniqueness of the resulting system to satisfy these additional constraints.

Function spaces studied below include polynomials, piecewise polynomials, trigonometric polynomials, and radial basis functions.

6.1 Polynomial Interpolation

We will begin by studying polynomials. There are several motivating factors for doing this:

- Everyone is familiar with polynomials.
- Polynomials can be easily and efficiently evaluated using Horner's algorithm.
- We may have heard of the Weierstrass Approximation Theorem which states that any continuous function can be approximated arbitrarily closely by a polynomial (of sufficiently high degree).
- A lot is known about polynomial interpolation, and serves as starting point for other methods.

Formally, we are interested in solving the following

Problem: Given data (x_i, y_i), $i = 0, \ldots, n$, find a polynomial p of minimal degree which matches the data in the sense of (1), i.e., for which

$$p(x_i) = y_i, \qquad i = 0, \ldots, n.$$

Remark: We illustrate this problem with some numerical examples in the Maple worksheet `578_PolynomialInterpolation.mws`.

The numerical experiments suggest that $n+1$ data points can be interpolated by a polynomial of degree n. Indeed,

Theorem 6.1 *Let $n+1$ distinct real numbers $x_0, x_1, \ldots, x_n$ and associated values $y_0, y_1, \ldots, y_n$ be given. Then there exists a unique polynomial p_n of degree at most n such that*

$$p_n(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

Proof: Assume

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n.$$

Then the interpolation conditions (1) lead to the linear system $Ba = y$ with

$$B = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix}, \quad a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \text{ and } y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \ldots \\ y_n \end{bmatrix}.$$

For general data y this is a nonhomogeneous system, and we know that such a system has a unique solution if and only if the associated homogeneous system has only the trivial solution. This, however, implies

$$p_n(x_i) = 0, \qquad i = 0, 1, \ldots, n.$$

Thus, p_n has $n+1$ zeros. Now, the Fundamental Theorem of Algebra states that any nontrivial polynomial of degree n has n (possibly complex) zeros. Therefore p_n must be the zero polynomial, i.e., the homogeneous linear system has only the trivial solution $a_0 = a_1 = \ldots = a_n = 0$, and by the above comment the general nonhomogeneous problem has a unique solution. ♠

Remark: The matrix B in the proof of Theorem 6.1 is referred to as a *Vandermonde matrix.*

Next, we illustrate how such a (low-degree) interpolating polynomial can be determined. Let's assume we have the following data:

x	0	1	3
y	1	0	4

.

Now, since we want a square linear system, we pick the dimension of the approximation space to be $m = 3$, i.e., we use quadratic polynomials. As basis we take the monomials $b_0(x) = 1$, $b_1(x) = x$, and $b_2(x) = x^2$. Therefore, the interpolating polynomial will be of the form

$$p(x) = a_0 + a_1 x + a_2 x^2.$$

In order to determine the coefficients a_0, a_1, and a_2 we enforce the interpolation conditions (1), i.e., $p(x_i) = y_i$, $i = 0, 1, 2$. This leads to the linear system

$$\begin{array}{lccccccc} (p(0) =) & a_0 & & & & & = & 1 \\ (p(1) =) & a_0 & + & a_1 & + & a_2 & = & 0 \\ (p(3) =) & a_0 & + & 3a_1 & + & 9a_2 & = & 4 \end{array}$$

whose solution is easily verified to be $a_0 = 1$, $a_1 = -2$, and $a_2 = 1$. Thus,

$$p(x) = 1 - 2x + x^2.$$

Of course, this polynomial can also be written as

$$p(x) = (1 - x)^2.$$

So, had we chosen the basis of shifted monomials $b_0(x) = 1$, $b_1(x) = 1 - x$, and $b_2(x) = (1 - x)^2$, then the coefficients (for an expansion with respect to this basis) would have come out to be $a_0 = 0$, $a_1 = 0$, and $a_2 = 1$.

In general, use of the monomial basis leads to a Vandermonde system as listed in the proof above. This is a classical example of an ill-conditioned system, and thus should be avoided. We will look at other bases later.

We now provide a second (constructive) proof of Theorem 6.1.

Constructive Proof: First we establish uniqueness. To this end we assume that p_n and q_n both are n-th degree interpolating polynomials. Then

$$r_n(x) = p_n(x) - q_n(x)$$

is also an n-th degree polynomial. Moreover, by the Fundamental Theorem of Algebra r_n has n zeros (or is the zero polynomial). However, by the nature of p_n and q_n we have

$$r_n(x_i) = p_n(x_i) - q_n(x_i) = y_i - y_i = 0, \qquad i = 0, 1, \ldots, n.$$

Thus, r_n has $n+1$ zeros, and therefore must be identically equal to zero. This ensures uniqueness.

Existence is constructed by induction. For $n = 0$ we take $p_0 \equiv y_0$. Obviously, the degree of p_0 is less than or equal to 0, and also $p_0(x_0) = y_0$.

Now we assume p_{k-1} to be the unique polynomial of degree at most $k-1$ that interpolates the data (x_i, y_i), $i = 0, 1, \ldots, k-1$. We will construct p_k (of degree k) such that

$$p_k(x_i) = y_i, \qquad i = 0, 1, \ldots, k.$$

We let

$$p_k(x) = p_{k-1}(x) + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1})$$

with c_k yet to be determined. By construction, p_k is a polynomial of degree k which interpolates the data (x_i, y_i), $i = 0, 1, \ldots, k-1$.

We now determine c_k so that we also have $p_k(x_k) = y_k$. Thus,

$$(p_k(x_k) =) \quad p_{k-1}(x_k) + c_k(x_k - x_0)(x_k - x_1) \ldots (x_k - x_{k-1}) = y_k$$

or

$$c_k = \frac{y_k - p_{k-1}(x_k)}{(x_k - x_0)(x_k - x_1) \ldots (x_k - x_{k-1})}.$$

This is well defined since the denominator is nonzero due to the fact that we assume distinct data sites. ♠

The construction used in this alternate proof provides the starting point for the Newton form of the interpolating polynomial.

Newton Form

From the proof we have

$$\begin{aligned} p_k(x) &= p_{k-1}(x) + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1}) \\ &= p_{k-2}(x) + c_{k-1}(x - x_0)(x - x_1) \ldots (x - x_{k-2}) + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1}) \\ &\vdots \\ &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_k(x - x_0)(x - x_1) \ldots (x - x_{k-1}). \end{aligned}$$

Thus, the Newton form of the interpolating polynomial is given by

$$p_n(x) = \sum_{j=0}^{n} c_j \prod_{i=0}^{j-1} (x - x_i). \tag{2}$$

This notation implies that the empty product (when $j - 1 < 0$) is equal to 1.

The earlier proof also provides a formula for the coefficients in the Newton form:

$$c_j = \frac{y_j - p_{j-1}(x_j)}{(x_j - x_0)(x_j - x_1) \ldots (x_j - x_{j-1})}, \qquad p_0 \equiv c_0 = y_0. \tag{3}$$

So the Newton coefficients can be computed recursively. This leads to a first algorithm for the solution of the interpolation problem.

Algorithm

```
Input $n$, $x_0, x_1, \ldots, x_n$, $y_0, y_1, \ldots, y_n$

$c_0 = y_0$

for $k$ from 1 to $n$ do

    $d = x_k - x_{k-1}$
    $u = c_{k-1}$
    for $i$ from $k-2$ by $-1$ to 0 do        % build $p_{k-1}$
        $u = u(x_k - x_i) + c_i$        % Horner
        $d = d(x_k - x_i)$        % accumulate denominator
    end
    $c_k = \frac{y_k - u}{d}$

end

Output $c_0, c_1, \ldots, c_n$
```

Remark: A more detailed derivation of this algorithm is provided in the textbook on page 310. However, the standard, more efficient, algorithm for computing the coefficients of the Newton form (which is based on the use of divided differences) is presented in the next section.

Example: We now compute the Newton form of the polynomial interpolating the data

x	0	1	3
y	1	0	4

.

According to (2) and (3) we have

$$p_2(x) = \sum_{j=0}^{2} c_j \prod_{i=0}^{j-1} (x - x_i) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1)$$

with

$$c_0 = y_0 = 1, \text{ and } c_j = \frac{y_j - p_{j-1}(x_j)}{(x_j - x_0)(x_j - x_1)\ldots(x_j - x_{j-1})}, \qquad j = 1, 2.$$

Thus, we are representing the space of quadratic polynomials with the basis $b_0(x) = 1$, $b_1(x) = x - x_0 = x$, and $b_2(x) = (x - x_0)(x - x_1) = x(x - 1)$.

We now determine the two remaining coefficients. First,

$$c_1 = \frac{y_1 - p_0(x_1)}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0} = \frac{0 - 1}{1 - 0} = -1.$$

This gives us

$$p_1(x) = c_0 + c_1(x - x_0) = 1 - x.$$

Next,

$$c_2 = \frac{y_2 - p_1(x_2)}{(x_2 - x_0)(x_2 - x_1)} = \frac{4 - (1 - x_2)}{3 \cdot 2} = 1,$$

and so

$$p_2(x) = p_1(x) + c_2(x - x_0)(x - x_1) = 1 - x + x(x - 1).$$

Lagrange Form

A third representation (after the monomial and Newton forms) of the same (unique) interpolating polynomial is of the general form

$$p_n(x) = \sum_{j=0}^{n} y_j \ell_j(x). \tag{4}$$

Note that the coefficients here are the data values, and the polynomial basis functions ℓ_j are so-called *cardinal* (or *Lagrange*) *functions*. We want these functions to depend on the data sites $x_0, x_1, \ldots, x_n$, but not the data values $y_0, y_1, \ldots, y_n$. In order to satisfy the interpolation conditions (1) we require

$$p_n(x_i) = \sum_{j=0}^{n} y_j \ell_j(x_i) = y_i, \qquad i = 0, 1, \ldots, n.$$

Clearly, this is ensured if

$$\ell_j(x_i) = \delta_{ij}, \tag{5}$$

which is called a *cardinality* (or *Lagrange*) *condition*.

How do we determine the Lagrange functions ℓ_j?

We want them to be polynomials of degree n and satisfy the cardinality conditions (5). Let's fix j, and assume

$$\begin{aligned} \ell_j(x) &= c(x - x_0)(x - x_1)\ldots(x - x_{j-1})(x - x_{j+1})\ldots(x - x_n) \\ &= c \prod_{\substack{i=0 \\ i \neq j}}^{n} (x - x_i). \end{aligned}$$

Clearly, $\ell_j(x_i) = 0$ for $j \neq i$. Also, ℓ_j depends only on the data sites and is a polynomial of degree n. The last requirement is $\ell_j(x_j) = 1$. Thus,

$$1 = c \prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i)$$

or

$$c = \frac{1}{\prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i)}.$$

Again, the denominator is nonzero since the data sites are assumed to be distinct. Therefore, the Lagrange functions are given by

$$\ell_j(x) = \prod_{\substack{i=0\\i\neq j}}^{n} \frac{x - x_i}{x_j - x_i}, \qquad j = 0, 1, \ldots, n,$$

and the Lagrange form of the interpolating polynomial is

$$p_n(x) = \sum_{j=0}^{n} y_j \prod_{\substack{i=0\\i\neq j}}^{n} \frac{x - x_i}{x_j - x_i}. \tag{6}$$

Remarks:

1. We mentioned above that the monomial basis results in an interpolation matrix (a Vandermonde matrix) that is notoriously ill-conditioned. However, an advantage of the monomial basis representation is the fact that the interpolant can be evaluated efficiently using Horner's method.

2. The interpolation matrix for the Lagrange form is the identity matrix and the coefficients in the basis expansion are given by the data values. This makes the Lagrange form ideal for situations in which many experiments with the same data sites, but different data values need to be performed. However, evaluation (as well as differentiation or integration) is more expensive.

3. A major advantage of the Newton form is its efficiency in the case of adaptive interpolation, i.e., when an existing interpolant needs to be refined by adding more data. Due to the recursive nature, the new interpolant can be determined by updating the existing one (as we did in the example above). If we were to form the interpolation matrix for the Newton form we would get a triangular matrix. Therefore, for the Newton form we have a balance between stability of computation and ease of evaluation.

Example: Returning to our earlier example, we now compute the Lagrange form of the polynomial interpolating the data

x	0	1	3
y	1	0	4

.

According to (4) we have

$$p_2(x) = \sum_{j=0}^{2} y_j \ell_j(x) = \ell_0(x) + 4\ell_2(x),$$

where (see (6))

$$\ell_j(x) = \prod_{\substack{i=0\\i\neq j}}^{2} \frac{x - x_i}{x_j - x_i}.$$

Thus,

$$\ell_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-1)(x-3)}{(-1)(-3)} = \frac{1}{3}(x-1)(x-3),$$

and

$$\ell_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{x(x-1)}{(3-0)(3-1)} = \frac{1}{6}x(x-1).$$

This gives us

$$p_2(x) = \frac{1}{3}(x-1)(x-3) + \frac{2}{3}x(x-1).$$

Note that here we have represented the space of quadratic polynomials with the (cardinal) basis $b_0(x) = \frac{1}{3}(x-1)(x-3)$, $b_2(x) = \frac{1}{6}x(x-1)$, and $b_1(x) = \ell_1(x) = -\frac{1}{2}x(x-3)$ (which we did not need to compute here).

Summarizing our example, we have found four different forms of the quadratic polynomial interpolating our data:

monomial	$p_2(x) = 1 - 2x + x^2$
shifted monomial	$p_2(x) = (1-x)^2$
Newton form	$p_2(x) = 1 - x + x(x-1)$
Lagrange form	$p_2(x) = \frac{1}{3}(x-1)(x-3) + \frac{2}{3}x(x-1)$

The basis polynomials for these four cases are displayed in Figures 1 and 2.

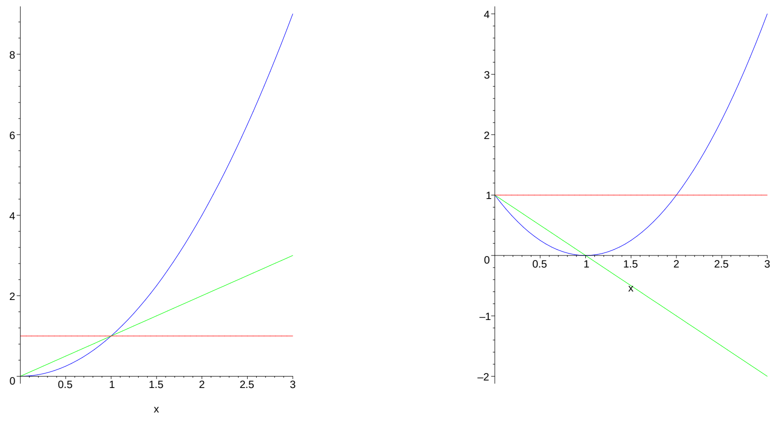

Figure 1: Monomial (left) and shifted monomial (right) bases for the interpolation example.

Error in Polynomial Interpolation

In the introduction we mentioned the Weierstrass Approximation Theorem as one of the motivations for the use of polynomials. Here are the details.

Theorem 6.2 *Let $f \in C[a,b]$ and $\epsilon > 0$. Then there exists a polynomial p of sufficiently high degree such that*

$$|f(x) - p(x)| < \epsilon$$

for all $x \in [a,b]$.

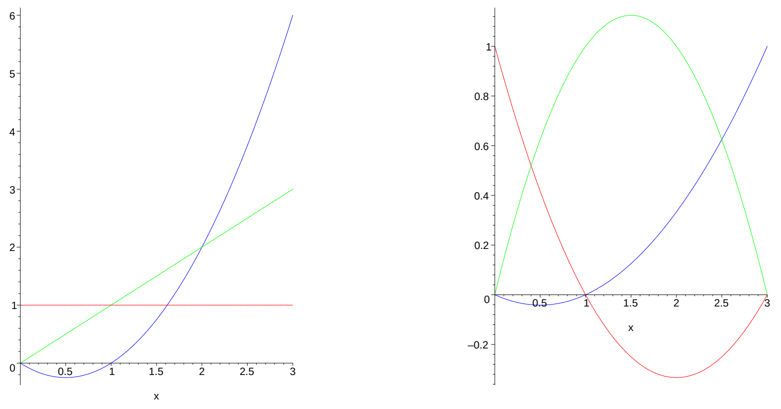
Figure 2: Basis polynomials for the Newton (left) and Lagrange form (right).

We will sketch the proof of this theorem later. Now we only point out that – as nice as this theorem is – it does not cover interpolation of (points of) f by p. For this problem we have

Theorem 6.3 *Let $f \in C^{n+1}[a,b]$ and p be a polynomial of degree at most n which interpolates f at the $n+1$ distinct points $x_0, x_1, \ldots, x_n$ in $[a,b]$ (i.e., $p(x_i) = f(x_i)$, $i = 0, 1, \ldots, n$). Then, for each $x \in [a,b]$ there exists a number $\xi_x \in (a,b)$ such that*

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) \prod_{i=0}^{n} (x - x_i). \tag{7}$$

In order to prove this result we need to recall Rolle's Theorem:

Theorem 6.4 *If $f \in C[a,b]$ and f' exists on (a,b), and if $f(a) = f(b) = 0$, then there exists a number $\xi \in (a,b)$ such that $f'(\xi) = 0$.*

Proof of Theorem 6.3: If x coincides with one of the data sites x_i, $i = 0, 1, \ldots, n$, then it is easy to see that both sides of equation (7) are zero.

Thus we now assume $x \neq x_i$ to be fixed. We start be defining

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

and

$$\phi = f - p - \lambda w$$

with λ such that $\phi(x) = 0$, i.e.,

$$\lambda = \frac{f(x) - p(x)}{w(x)}.$$

We need to show that $\lambda = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x)$ for some $\xi_x \in (a,b)$.

Since $f \in C^{n+1}[a,b]$ we know that $\phi \in C^{n+1}[a,b]$ also. Moreover,

$$\phi(t) = 0 \quad \text{for} \quad t = x, x_0, x_1, \ldots, x_n.$$

The first of these equations holds by the definition of λ, the remainder by the definition of w and the fact that p interpolates f at these points.

Now we apply Rolle's Theorem to ϕ on each of the $n+1$ subintervals generated by the $n+2$ points $x, x_0, x_1, \ldots, x_n$. Thus, ϕ' has (at least) $n+1$ distinct zeros in (a,b).

Next, by Rolle's Theorem (applied to ϕ' on n subintervals) we know that ϕ'' has (at least) n zeros in (a,b).

Continuing this argument we deduce that $\phi^{(n+1)}$ has (at least) one zero, ξ_x, in (a,b).

On the other hand, since

$$\phi(t) = f(t) - p(t) - \lambda w(t)$$

we have

$$\phi^{(n+1)}(t) = f^{(n+1)}(t) - p^{(n+1)}(t) - \lambda w^{(n+1)}(t).$$

However, p is a polynomial of degree at most n, so $p^{(n+1)} \equiv 0$. For w we have

$$w^{(n+1)}(t) = \frac{d^{n+1}}{dt^{n+1}} \prod_{i=0}^{n} (t - x_i) = (n+1)!.$$

Therefore,

$$\phi^{(n+1)}(t) = f^{(n+1)}(t) - \lambda (n+1)!.$$

Combining this with the information about the zero of $\phi^{(n+1)}$ we have

$$\begin{aligned} 0 = \phi^{(n+1)}(\xi_x) &= f^{(n+1)}(\xi_x) - \lambda(n+1)! \\ &= f^{(n+1)}(\xi_x) - \frac{f(x) - p(x)}{w(x)}(n+1)! \end{aligned}$$

or

$$f(x) - p(x) = f^{(n+1)}(\xi_x) \frac{w(x)}{(n+1)!}.$$

♠

Remarks:

1. The error formula (7) in Theorem 6.3 looks almost like the error formula for Taylor polynomials:

$$f(x) - T_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)^{n+1}.$$

   The difference lies in the fact that for Taylor polynomials the information is concentrated at one point x_0, whereas for interpolation the information is obtained at the points $x_0, x_1, \ldots, x_n$.

2. The error formula (7) will be used later to derive (and judge the accuracy of) methods for solving differential equations as well as for numerical differentiation and integration.

If the data sites are equally spaced, i.e., $\Delta x_i = x_i - x_{i-1} = h$, $i = 1, 2, \ldots, n$, it is possible to derive the bound

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

in the error formula (7). If we also assume that the derivatives of f are bounded, i.e., $|f^{(n+1)}(x)| \le M$ for all $x \in [a, b]$, then we get

$$|f(x) - p(x)| \le \frac{M}{4(n+1)} h^{n+1}.$$

Thus, the error for interpolation with degree-n polynomials is $\mathcal{O}(h^{n+1})$. We illustrate this formula for a fixed-degree polynomial in the Maple worksheet `578_PolynomialInterpolationError.mws`.

Formula (7) tells us that the interpolation error depends on the function we're interpolating as well as the data sites (interpolation nodes) we use. If we are interested in minimizing the interpolation error, then we need to choose the data sites x_i, $i = 0, 1, \ldots, n$, such that

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

is minimized. Figure 3 shows the graph of the function $|w|$ for 10 equally spaced (red,dashed) and 10 optimally spaced (green,solid) data sites on the interval $[-1, 1]$. We will derive the locations of the optimally spaced points (known as *Chebyshev points*) below.

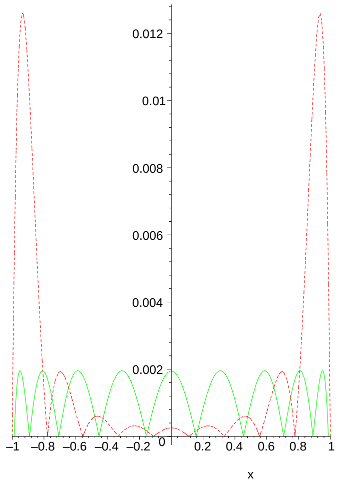

Figure 3: Graph of the function $|w|$ for 10 equally spaced (red,dashed) and 10 Chebyshev points (green,solid) on $[-1, 1]$.

Figure 3 shows that the error near the endpoints of the interval can be significant if we insist on using equally spaced points. A classical example that illustrates this phenomenon is the Runge function

$$f(x) = \frac{1}{1 + 25x^2}.$$

We present this example in the Maple worksheet `578_Runge.mws`.

In order to discuss Chebyshev points we need to introduce a certain family of orthogonal polynomials called *Chebyshev polynomials.*

Chebyshev Polynomials

The Chebyshev polynomials (of the first kind) can be defined recursively. We have

$$T_0(x) = 1, \qquad T_1(x) = x,$$

and

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad n \geq 1. \tag{8}$$

An explicit formula for the n-th degree Chebyshev polynomial is

$$T_n(x) = \cos(n \arccos x), \quad x \in [-1, 1], \quad n = 0, 1, 2, \ldots. \tag{9}$$

We can verify this explicit formula by using the trigonometric identity

$$\cos(A + B) = \cos A \cos B - \sin A \sin B.$$

This identity gives rise to the following two formulas:

$$\begin{aligned} \cos(n+1)\theta &= \cos n\theta \cos\theta - \sin n\theta \sin\theta \\ \cos(n-1)\theta &= \cos n\theta \cos\theta + \sin n\theta \sin\theta. \end{aligned}$$

Addition of these two formulas yields

$$\cos(n+1)\theta + \cos(n-1)\theta = 2\cos n\theta \cos\theta. \tag{10}$$

Now, if we let $x = \cos\theta$ (or $\theta = \arccos x$) then (10) becomes

$$\cos\left[(n+1)\arccos x\right] + \cos\left[(n-1)\arccos x\right] = 2\cos\left(n \arccos x\right)\cos\left(\arccos x\right)$$

or

$$\cos\left[(n+1)\arccos x\right] = 2x\cos\left(n\arccos x\right) - \cos\left[(n-1)\arccos x\right],$$

which is of the same form as the recursion (8) for the Chebyshev polynomials provided we identify T_n with the explicit formula (9).

Some properties of Chebyshev polynomials are

1. $|T_n(x)| \leq 1, \quad x \in [-1, 1]$.
2. $T_n\left(\cos\frac{i\pi}{n}\right) = (-1)^i, \quad i = 0, 1, \ldots, n$. These are the extrema of T_n.
3. $T_n\left(\cos\frac{2i-1}{2n}\pi\right) = 0, \quad i = 1, \ldots, n$. This gives the zeros of T_n.
4. The leading coefficient of T_n, $n = 1, 2, \ldots$, is 2^{n-1}, i.e.,

$$T_n(x) = 2^{n-1}x^n + \text{lower order terms}.$$

Items 1–3 follow immediately from (9). Item 4 is clear from the recursion formula (8). Therefore, $2^{1-n}T_n$ is a *monic* polynomial, i.e., its leading coefficient is 1. We will need the following property of monic polynomials below.

Theorem 6.5 *For any monic polynomial p of degree n on $[-1,1]$ we have*

$$\|p\|_\infty = \max_{-1\le x\le 1} |p(x)| \ge 2^{1-n}.$$

Proof: Textbook page 317. ♠

We are now ready to return to the formula for the interpolation error. From (7) we get on $[-1,1]$

$$\begin{aligned} \|f-p\|_\infty &= \max_{-1\le x\le 1} |f(x)-p(x)| \\ &\le \frac{1}{(n+1)!} \max_{-1\le x\le 1} \left|f^{(n+1)}(x)\right| \max_{-1\le x\le 1} \underbrace{\left|\prod_{i=0}^{n}(x-x_i)\right|}_{=w(x)}. \end{aligned}$$

We note that w is a monic polynomial of degree $n+1$, and therefore

$$\max_{-1\le x\le 1} |w(x)| \ge 2^{-n}.$$

From above we know that $2^{-n}T_{n+1}$ is also a monic polynomial of degree $n+1$ with extrema $T_{n+1}\left(\cos\frac{i\pi}{n+1}\right) = \frac{(-1)^i}{2^n}$, $i=0,1,\ldots,n+1$. By Theorem 6.5 the minimal value of $\max_{-1\le x\le 1} |w(x)|$ is 2^{-n}. We just observed that this value is attained for T_{n+1}.

Thus, the zeros x_i of the optimal w should coincide with the zeros of T_{n+1}, or

$$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right), \qquad i=0,1,\ldots,n. \tag{11}$$

These are the Chebyshev points used in Figure 3 and in the Maple worksheet `578_Runge.mws`.

If the data sites are taken to be the Chebyshev points on $[-1,1]$, then the interpolation error (7) becomes

$$|f(x)-p(x)| \le \frac{1}{2^n(n+1)!} \max_{-1\le t\le 1} \left|f^{(n+1)}(t)\right|, \qquad |x|\le 1.$$

Remark: The Chebyshev points (11) are for interpolation on $[-1,1]$. If a different interval is used, the Chebyshev points need to be transformed accordingly.

We just observed that the Chebyshev nodes are "optimal" interpolation points in the sense that for any given f and fixed degree n of the interpolating polynomial, if we are free to choose the data sites, then the Chebyshev points will yield the most accurate approximation (measured in the maximum norm).

For equally spaced interpolation points, our numerical examples in the Maple worksheets `578_PolynomialInterpolation.mws` and `578_Runge.mws` have shown that, contrary to our intuition, the interpolation error (measured in the maximum norm) does not tend to zero as we increase the number of interpolation points (or polynomial degree).

The situation is even worse. There is also the following more general result proved by Faber in 1914.

Theorem 6.6 *For any fixed system of interpolation nodes $a \le x_0^{(n)} < x_1^{(n)} < \ldots < x_n^{(n)} \le b$ there exists a function $f \in C[a,b]$ such that the interpolating polynomial p_n does not uniformly converge to f, i.e.,*

$$\|f - p_n\|_\infty \not\to 0, \quad n \to \infty.$$

This, however, needs to be contrasted with the positive result (very much in the spirit of the Weierstrass Approximation Theorem) for the situation in which we are free to choose the location of the interpolation points.

Theorem 6.7 *Let $f \in C[a,b]$. Then there exists a system of interpolation nodes such that*

$$\|f - p_n\|_\infty \to 0, \quad n \to \infty.$$

Proof: Uses the Weierstrass Approximation Theorem as well as the Chebyshev Alternation Theorem. ♠

Finally, if we insist on using the Chebyshev points as data sites, then we have the following theorem due to Fejér.

Theorem 6.8 *Let $f \in C[-1,1]$, and $x_0, x_1, \ldots, x_{n-1}$ be the first n Chebyshev points. Then there exists a polynomial p_{2n-1} of degree $2n-1$ that interpolates f at $x_0, x_1, \ldots, x_{n-1}$, and for which*

$$\|f - p_{2n-1}\|_\infty \to 0, \quad n \to \infty.$$

<u>Remark:</u> The polynomial p_{2n-1} also has zero derivatives at x_i, $i = 0, 1, \ldots, n-1$.

We now outline the classical (constructive) proof of the Weierstrass Approximation Theorem.

The main ingredient are the so-called *Bernstein polynomials.*

Definition 6.9 *For any $f \in C[0,1]$*

$$(B_n f)(x) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) \binom{n}{k} x^k (1-x)^{n-k}, \quad x \in [0,1],$$

is called the n-th degree Bernstein polynomial associated with f.

Here

$$\binom{n}{k} = \begin{cases} \dfrac{n!}{k!(n-k)!}, & 0 \le k \le n, \\ 0 & \text{otherwise.} \end{cases}$$

The main steps of the proof of the Weierstrass Approximation Theorem (transformed to the interval $[0,1]$) are now

1. Prove the Bohman-Korovkin Theorem

   Theorem 6.10 *Let L_n $(n \geq 1)$ be a sequence of positive linear operators. If $\|L_n f - f\|_\infty \to 0$ for $f(x) = 1$, x, and x^2, then $\|L_n f - f\|_\infty \to 0$ for all $f \in C[a,b]$.*

2. Show that the Bernstein polynomials give rise to positive linear operators from $C[0,1]$ to $C[0,1]$, i.e., show

   (a) B_n is linear, i.e.,
   $$B_n(\alpha f + \beta g) = \alpha B_n f + \beta B_n g,$$

   (b) $B_n f \geq 0$ provided $f \geq 0$.

3. Show $B_n 1 = 1$, $B_n x = x$, and $B_n x^2 \to x^2$, $n \to \infty$.

Details are provided in the textbook on pages 321–323.

We illustrate the convergence behavior of the Weierstrass Approximation Theorem based on Bernstein polynomials in the Maple worksheet `578_Bezier.mws`. It is generally accepted that the speed of convergence is too slow for practical purposes. In fact, one can show that in order to have a maximum error smaller than 0.01 one needs at least a degree of 1.6×10^7.

<u>Remark:</u> The *Bernstein basis polynomials* (cf. Definition 6.9, and Figure 4)

$$B_k^n(x) = \binom{n}{k} x^k (1-x)^{n-k}$$

play an important role in computer-aided geometric design (CAGD). There, so-called *Bézier curves* are parametric curves defined by

$$x(t) = \sum_{k=0}^{n} b_k B_k^n(t), \qquad t \in [0,1].$$

Here the b_k, $k = 0, 1, \ldots, n$, are points in $\mathbb{R}^2$ or $\mathbb{R}^3$ referred to as *control points*. Thus, x can describe either a planar curve, or a curve in space. The Bézier representation of a parametric curve has such nice properties as fast (recursive) evaluation and intuitive shape control. `578_Bezier.mws` contains examples of a 2D and 3D Bézier curve along with its *control polygon* formed by the control points b_k, $k = 0, \ldots, n$.

6.2 Divided Differences

Recall the Newton form of the interpolating polynomial (2)

$$p_n(x) = \sum_{j=0}^{n} c_j \underbrace{\prod_{k=0}^{j-1}(x - x_k)}_{=b_j(x),\ (b_0(x)=1)} = \sum_{j=0}^{n} c_j b_j(x).$$

If we enforce the interpolation conditions

$$p_n(x_i) = f(x_i), \qquad i = 0, 1, \ldots, n,$$

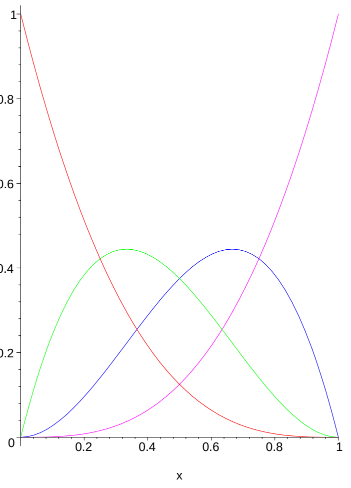

Figure 4: Graph of the cubic Bernstein basis polynomials on $[0,1]$.

then we obtain a linear system of the form

$$Bc = f,$$

where $B_{ij} = b_j(x_i)$. As mentioned earlier, the matrix B is lower triangular. This is clear since

$$b_j(x_i) = \prod_{k=0}^{j-1}(x_i - x_k) = 0 \quad \text{if } i < j.$$

In fact, in detail the linear system looks like

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & x_1 - x_0 & 0 & \dots & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x_0 & (x_n - x_0)(x_n - x_1) & \dots & \prod_{k=0}^{n-1}(x_n - x_k) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}.$$

We see that $c_0 = f(x_0)$, and c_1 depends on x_0, x_1, and f at those points. Similarly, c_n depends on all the points $x_0, x_1, \dots, x_n$, as well as f at those points. We indicate this dependence with the symbolic notation

$$c_n = f[x_0, x_1, \dots, x_n].$$

With this new notation the Newton form becomes

$$p_n(x) = \sum_{j=0}^{n} f[x_0, x_1, \dots, x_j] \prod_{k=0}^{j-1}(x - x_k). \tag{12}$$

Since the Newton form of the interpolating polynomial is constructed recursively, and since each of the basis functions b_j is a monic polynomial, we can define

Definition 6.11 *The coefficient c_n of the unique polynomial of degree n interpolating f at the distinct points $x_0, x_1, \dots, x_n$ is denoted by*

$$c_n = f[x_0, x_1, \dots, x_n],$$

and called the n-th order divided difference of f at $x_0, x_1, \dots, x_n$.

How to Compute Divided Differences Efficiently

We begin with an example for the case $n = 1$, i.e., a linear polynomial. The recursive construction of the Newton form suggests

$$p_0(x) = c_0 = f(x_0) = f[x_0]$$

and

$$\begin{aligned} p_1(x) &= p_0(x) + c_1(x - x_0) \\ &= p_0(x) + f[x_0, x_1](x - x_0). \end{aligned}$$

The interpolation condition at x_1 suggests

$$p_1(x_1) = \underbrace{p_0(x_1)}_{=f(x_0)} + f[x_0, x_1](x_1 - x_0) = f(x_1).$$

Solving this for the first order divided difference yields

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

which explains the use of the term "divided difference".

For the general case we have

Theorem 6.12 *Let the distinct points $x_0, x_1, \dots, x_n$ and f be given. Then*

$$f[x_0, x_1, \dots, x_{n-1}, x_n] = \frac{f[x_1, \dots, x_{n-1}, x_n] - f[x_0, x_1, \dots, x_{n-1}]}{x_n - x_0}.$$

Proof:

Consider the following interpolating polynomials:

1. q (of degree $n - 1$) interpolating f at $x_1, \dots, x_n$,
2. p_{n-1} (of degree $n - 1$) interpolating f at $x_0, \dots, x_{n-1}$,
3. p_n (of degree n) interpolating f at $x_0, \dots, x_n$,

Then we have

$$p_n(x) = q(x) + \frac{x - x_n}{x_n - x_0}\left(q(x) - p_{n-1}(x)\right). \tag{13}$$

This identity was essentially proven in homework problem 6.1.9. To establish the claim of the theorem we compare the coefficients of x^n on both sides of (13). For the left-hand side the leading coefficient of p_n is given by $f[x_0, x_1, \ldots, x_n]$, whereas on the right-hand side x^n has the coefficient

$$\frac{1}{x_n - x_0} \left(f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}] \right).$$

♠

Remark: The formula (13) used in the proof is closely related to two other recursive algorithms for interpolation due to Neville and Aitken (see the Maple worksheet `578_PolynomialInterpolation.mws` where Neville's method was illustrated graphically – without explicitly giving an algorithm for the method).

Remark: The formula in Theorem 6.12 can be generalized to

$$f[x_i, x_{i+1}, \ldots, x_{i+j-1}, x_{i+j}] = \frac{f[x_{i+1}, \ldots, x_{i+j-1}, x_{i+j}] - f[x_i, x_{i+1}, \ldots, x_{i+j-1}]}{x_{i+j} - x_i}. \quad (14)$$

The recursion of Theorem 6.12 (or its generalization (14)) can be used to formulate an efficient algorithm for the computation of divided differences, i.e., the coefficients in the Newton form of the interpolating polynomial. To keep the notation compact, we introduce the abbreviation

$$c_{ij} = f[x_i, \ldots, x_{i+j}],$$

i.e., the values to be interpolated are $c_{0,0} = f[x_0] = f(x_0), \ldots, c_{n,0} = f[x_n] = f(x_n)$.

Algorithm

```
Input $x_0, \ldots, x_n$, $c_{0,0}, \ldots, c_{n,0}$

for $j = 1$ to $n$ do

    for $i = 0$ to $n - j$ do
        $c_{ij} = \frac{c_{i+1,j-1} - c_{i,j-1}}{x_{i+j} - x_i}$
    end

end

Output array $c$
```

The output array c of the above algorithm can be arranged in the form of a *divided difference table*. For the case $n = 3$ we get

x_i	$c_{i,0}$	$c_{i,1}$	$c_{i,2}$	$c_{i,3}$
x_0	$f[x_0]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
x_1	$f[x_1]$	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	
x_2	$f[x_2]$	$f[x_2, x_3]$		
x_3	$f[x_3]$			

In this table, the data is entered in the first two columns, and then the remaining columns are computed recursively by combining the two entries just to the left and below according to formula (14). Also, the coefficients of the Newton form are listed in the first row of the table (cf. (12)).

Example: For the data

x	0	1	3
y	1	0	4

we get the divided difference table

x_i	$c_{i,0}$	$c_{i,1}$	$c_{i,2}$
$x_0 = 0$	$f[x_0] = 1$	$f[x_0, x_1] = \frac{f[x_1]-f[x_0]}{x_1-x_0} = \frac{0-1}{1-0} = -1$	$f[x_0, x_1, x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0} = \frac{2-(-1)}{3-0} = 1$
$x_1 = 1$	$f[x_1] = 0$	$f[x_1, x_2] = \frac{f[x_2]-f[x_1]}{x_2-x_1} = \frac{4-0}{3-1} = 2$	
$x_2 = 3$	$f[x_2] = 4$		

Therefore, the interpolating polynomial is given by

$$\begin{aligned} p_2(x) &= 1 - 1(x - x_0) + 1(x - x_0)(x - x_1) \\ &= 1 - x + x(x - 1). \end{aligned}$$

Since only the coefficients in the first row of the divided difference table are needed for the interpolating polynomial, another version of the divided difference algorithm is often used which efficiently overwrites the entries in a one-dimensional coefficient vector c.

Algorithm

```
Input $x_0, \ldots, x_n$, $c_0 = f(x_0), \ldots, c_n = f(x_n)$

for $j = 1$ to $n$ do

    for $i = n$ by $-1$ to $j$ do
        $c_i = \frac{c_i - c_{i-1}}{x_i - x_{i-j}}$
    end

end

Output Newton coefficients $c$
```

Properties of Divided Differences

Theorem 6.13 *The divided difference is a symmetric function of its arguments, i.e., the order of the arguments does not matter.*

Proof: This follows directly from the definition of the divided difference as the leading coefficient of the unique interpolating polynomial. ♠

Theorem 6.14 *Let $x_0, x_1, \dots, x_n$ be distinct nodes, and let p be the unique polynomial interpolating f at these nodes. If t is not a node, then*

$$f(t) - p(t) = f[x_0, x_1, \dots, x_n, t] \prod_{i=0}^{n} (t - x_i).$$

Proof: Let p be the degree n polynomial interpolating f at $x_0, x_1, \dots, x_n$, and q the degree $n+1$ interpolating polynomial to f at $x_0, x_1, \dots, x_n, t$. By the recursive Newton construction we have

$$q(x) = p(x) + f[x_0, x_1, \dots, x_n, t] \prod_{i=0}^{n} (x - x_i). \tag{15}$$

Now, since $q(t) = f(t)$, the statement follows immediately by replacing x by t in (15). ♠

If we compare the statement of Theorem 6.14 with that of Theorem 6.3 we get

Theorem 6.15 *If $f \in C^n[a,b]$ and $x_0, x_1, \dots, x_n$ are distinct points in $[a,b]$, then there exists a $\xi \in (a,b)$ such that*

$$f[x_0, x_1, \dots, x_n] = \frac{1}{n!} f^{(n)}(\xi).$$

Remark: Divided differences can be used to estimate derivatives. This idea is, e.g., employed in the construction of so-called ENO (essentially non-oscillating) methods for solving hyperbolic PDEs.

6.3 Hermite Interpolation

Now we consider the situation in which we also want to interpolate derivative information at the data sites. In order to be able to make a statement about the solvability of this kind of problem we will have to impose some restrictions.

To get an idea of the nature of those restrictions we begin with a few simple examples.

Example 1: If the data is given by the following four pieces of information

$$f(0) = 1, \quad f(1) = 0, \quad f'(0) = 0, \quad f'(1) = 0,$$

then we should attempt to interpolate with a polynomial of degree three, i.e.,

$$p_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

Since we also need to match derivative information, we need the derivative of the polynomial

$$p_3'(x) = a_1 + 2a_2 x + 3a_3 x^2.$$

The four interpolation conditions

$$p_3(0) = a_0 \quad = \quad f(0) = 1$$

$$
\begin{aligned}
p_3(1) = a_0 + a_1 + a_2 + a_3 &= f(1) = 0 \\
p_3'(0) = a_1 &= f'(0) = 0 \\
p_3'(1) = a_1 + 2a_2 + 3a_3 &= f'(1) = 0
\end{aligned}
$$

lead to the linear system

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}
=
\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

whose solution is easily seen to be

$$
a_0 = 1, \quad a_1 = 0, \quad a_2 = -3, \quad a_3 = 2.
$$

Therefore, the (unique) interpolating polynomial is

$$
p_3(x) = 1 - 3x^2 + 2x^3.
$$

Figure 5 shows the data along with the interpolating polynomial.

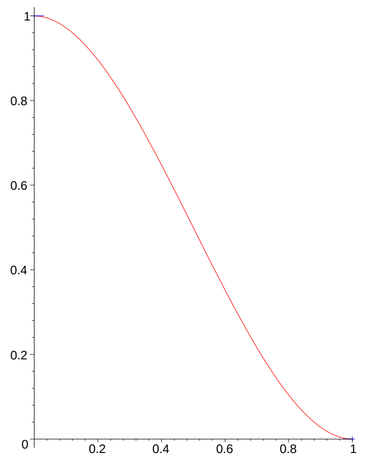

Figure 5: Graph of the cubic polynomial interpolating the data of Example 1.

Example 2: Now we consider the three pieces of information

$$
f(0) = 1, \quad f(1) = 0, \quad f'(1/2) = 0.
$$

Thus, we should attempt to interpolate with a quadratic polynomial, i.e.,

$$
p_2(x) = a_0 + a_1 x + a_2 x^2
$$

with derivative

$$
p_2'(x) = a_1 + 2a_2 x.
$$

The three interpolation conditions are

$$
p_2(0) = a_0 \quad = \quad f(0) = 1
$$

$$\begin{aligned} p_2(1) = a_0 + a_1 + a_2 &= f(1) = 0 \\ p_2'(1/2) = a_1 + a_2 &= f'(1/2) = 0 \end{aligned}$$

with the corresponding linear system

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

Clearly, $a_0 = 1$, which leaves the 2×2 sub-problem

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

for the remaining two coefficients. Since this system matrix is singular (with an inconsistent right-hand side), the given problem has no solution, i.e., there is no parabola that interpolates the given data.

Example 3: If we use the same data as in Example 2, but attempt to use a cubic polynomial

$$p_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

with derivative

$$p_3'(x) = a_1 + 2a_2 x + 3a_3 x^2$$

then we get the 3×4 linear system

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 3/4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

or equivalently

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & -1/4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix},$$

Again, $a_0 = 1$ is determined, and also $a_3 = -4$, but now the remaining condition is not enough to determine the two coefficients a_1 and a_2. Thus, a cubic polynomial that interpolates the data is not uniquely determined. Two possible solutions are depicted in Figure 6.

Hermite Interpolation Problem:

Given: $n + 1$ distinct data sites $x_0, x_1, \ldots, x_n$ in $[a, b]$, and associated data y_{ij}, $i = 0, 1, \ldots, n$, $j = 0, 1, \ldots, k_i - 1$ with positive integers k_i.

Find: a polynomial of minimal degree such that

$$p^{(j)}(x_i) = y_{ij}, \qquad i = 0, 1, \ldots, n, \ j = 0, 1, \ldots, k_i - 1. \tag{16}$$

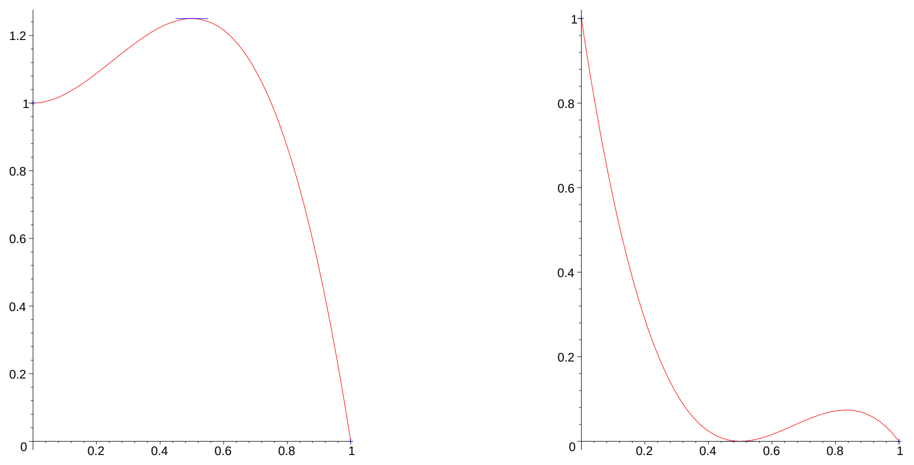

Figure 6: Graph of the two different cubic polynomials interpolating the data of Example 2.

Remarks:

1. We can think of the data values y_{ij} as function and derivative values of the form $f^{(j)}(x_i)$ so that the Hermite interpolation conditions (16) generalize the (Lagrange) interpolation conditions (1).

2. The number of conditions in (16) is
$$\sum_{i=0}^{n} k_i =: m + 1.$$

3. It is important to note that *no gap* is permitted in the sequence of derivatives given as data. This is the restriction alluded to at the beginning of this section.

4. If certain gaps are present the problem becomes a so-called *Hermite-Birkhoff interpolation problem* which is more difficult to deal with. We will come back to Hermite-Birkhoff interpolation later when we discuss radial basis functions.

We now have the following

Theorem 6.16 *There exists a unique polynomial p_m of degree at most m which solves the Hermite interpolation problem formulated above.*

Proof: There are $m+1$ linear conditions to be satisfied by the polynomial of degree m containing $m+1$ unknown coefficients. We need to show that the system matrix obtained from (16) is non-singular.

Alternatively, we can show that the corresponding homogeneous linear system
$$p^{(j)}(x_i) = 0, \quad i = 0, 1, \ldots, n, \;\; j = 0, 1, \ldots, k_i - 1,$$

has only the trivial solution $p \equiv 0$. If p and its derivatives have all the stated zeros, then it must be of the form

$$p(x) = c \underbrace{(x - x_0)^{k_0}(x - x_1)^{k_1} \dots (x - x_n)^{k_n}}_{=:q(x)}, \tag{17}$$

with some constant c.

Now,

$$\deg(q) = \sum_{i=0}^{n} k_i = m + 1,$$

and from (17) $\deg(p) = \deg(q) = m + 1$. However, p was assumed to be of degree m (with $m + 1$ unknown coefficients). The only possibility is that $p \equiv 0$. ♠

Newton Form:

We now need to discuss divided difference with repeated arguments. First, consider

$$\lim_{x \to x_0} f[x_0, x] = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} = f'(x_0).$$

This motivates the notation

$$f[x_0, x_0] = f'(x_0).$$

Using the identity

$$f[x_0, x_1, \dots, x_k] = \frac{1}{k!} f^{(k)}(\xi), \quad \xi \in (x_0, x_k),$$

listed earlier in Theorem 6.15 we are led to

$$f[\underbrace{x_0, \dots, x_0}_{k+1}] = \frac{1}{k!} f^{(k)}(x_0). \tag{18}$$

Example: Consider the data

$$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ y_{00} (= f(x_0)) &= -1, \\ y_{01} (= f'(x_0)) &= -2, \\ y_{10} (= f(x_1)) &= 0, \\ y_{11} (= f'(x_1)) &= 10, \\ y_{12} (= f''(x_1)) &= 40. \end{aligned}$$

Using the divided difference notation introduced in (18) this corresponds to

$$\begin{aligned} f[x_0] &= -1, \\ f[x_0, x_0] &= -2, \\ f[x_1] &= 0, \\ f[x_1, x_1] &= 10, \\ f[x_1, x_1, x_1] &= \frac{40}{2!} = 20. \end{aligned}$$

We can use a generalized divided difference table to determine the coefficients of the interpolating polynomial in Newton form. The table contains k_i rows for each data site x_i with the given data (framed) entered in the corresponding first rows (and possibly repeated in rows below). Here is what the table looks like based on the data in our example:

x_i	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$	f[, , , ,]
$\boxed{x_0 = 0}$	$\boxed{f[x_0] = -1}$	$\boxed{f[x_0, x_0] = -2}$	$f[x_0, x_0, x_1]$	$f[x_0, x_0, x_1, x_1]$	$f[x_0, x_0, x_1, x_1, x_1]$
$x_0 = 0$	$f[x_0] = -1$	$f[x_0, x_1]$	$f[x_0, x_1, x_1]$	$f[x_0, x_1, x_1, x_1]$	
$\boxed{x_1 = 1}$	$\boxed{f[x_1] = 0}$	$\boxed{f[x_1, x_1] = 10}$	$\boxed{f[x_1, x_1, x_1] = 20}$		
$x_1 = 1$	$f[x_1] = 0$	$f[x_1, x_1] = 10$			
$x_1 = 1$	$f[x_1] = 0$				

The missing entries are now computed as

$$\begin{aligned}
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{0 - (-1)}{1 - 0} = 1 \\
f[x_0, x_0, x_1] &= \frac{f[x_0, x_1] - f[x_0, x_0]}{x_1 - x_0} = \frac{1 - (-2)}{1 - 0} = 3 \\
f[x_0, x_1, x_1] &= \frac{f[x_1, x_1] - f[x_0, x_1]}{x_1 - x_0} = \frac{10 - 1}{1 - 0} = 9 \\
f[x_0, x_0, x_1, x_1] &= \frac{f[x_0, x_1, x_1] - f[x_0, x_0, x_1]}{x_1 - x_0} = \frac{9 - 3}{1 - 0} = 6
\end{aligned}$$

and so on. The completed table is

x_i	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$	f[, , , ,]
0	−1	−2	3	6	5
0	−1	1	9	11	
1	0	10	20		
1	0	10			
1	0				

Using the coefficients listed in the first row of the table (just like we did for the regular divided difference table earlier) the interpolating polynomial is given by

$$\begin{aligned}
p(x) &= f[x_0] + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 \\
&\quad + f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1) + f[x_0, x_0, x_1, x_1, x_1](x - x_0)^2(x - x_1)^2 \\
&= -1 - 2(x - x_0) + 3(x - x_0)^2 + 6(x - x_0)^2(x - x_1) + 5(x - x_0)^2(x - x_1)^2 \\
&= -1 - 2x + 3x^2 + 6x^2(x - 1) + 5x^2(x - 1)^2.
\end{aligned}$$

The analogy to the Newton form for Lagrange (i.e., function value only) interpolation can be seen very nicely if we introduce an auxiliary node sequence

$$z_0 = x_0,\ z_1 = x_0,\ z_2 = x_1,\ z_3 = x_1,\ z_4 = x_1.$$

Then the interpolating polynomial is given by

$$p(x) = \sum_{j=0}^{3} f[z_0, \ldots, z_j] \prod_{i=0}^{j-1} (x - z_i).$$

Remark: The principles used in this examples can (with some notational efforts) be extended to the general Hermite interpolation problem.

The following error estimate is more general than the one provided in the textbook on page 344.

Theorem 6.17 *Let $x_0, x_1, \ldots, x_n$ be distinct data sites in $[a,b]$, and let $f \in C^{m+1}[a,b]$, where $m+1 = \sum_{i=0}^{n} k_i$. Then the unique polynomial p of degree at most m which solves the Hermite interpolation problem satisfies*

$$f(x) - p(x) = \frac{1}{(m+1)!} f^{(m+1)}(\xi_x) \prod_{i=0}^{n} (x - x_i)^{k_i},$$

where ξ_x is some point in (a,b).

Proof: Analogous to the proof of Theorem 6.3. See the textbook for the case $k_i = 2$, $i = 0, 1, \ldots, n$. ♠

Cardinal Hermite Interpolation

It is also possible to derive cardinal basis functions, but due to their complexity this is rarely done. For first order Hermite interpolation, i.e., data of the form

$$(x_i, f(x_i)), \quad (x_i, f'(x_i)), \qquad i = 0, 1, \ldots, n,$$

we make the assumption that p (of degree m, since we have $m+1 = 2n+2$ conditions) is of the form

$$p(x) = \sum_{j=0}^{n} f(x_j) L_j(x) + \sum_{j=0}^{n} f'(x_j) M_j(x).$$

The cardinality conditions for the basis functions are now

$$\begin{cases} L_j(x_i) = \delta_{ij} \\ L_j'(x_i) = 0 \end{cases} \qquad \begin{cases} M_j(x_i) = 0 \\ M_j'(x_i) = \delta_{ij} \end{cases}$$

which can be shown to lead to

$$\begin{aligned} L_j(x) &= \left(1 - 2(x - x_j)\ell_j'(x_j)\right) \ell_j^2(x), \qquad j = 0, 1, \ldots, n, \\ M_j(x) &= (x - x_j)\ell_j^2(x), \qquad j = 0, 1, \ldots, n, \end{aligned}$$

where

$$\ell_j(x) = \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i}, \qquad j = 0, 1, \ldots, n,$$

are the Lagrange basis functions of (6). Since the ℓ_j are of degree n it is easily verified that L_j and M_j are of degree $m = 2n+1$.

6.4 Spline Interpolation

One of the main disadvantages associated with polynomial interpolation were the oscillations resulting from the use high-degree polynomials. If we want to maintain such advantages as simplicity, ease and speed of evaluation, as well as similar approximation properties, we are naturally led to consider *piecewise polynomial interpolation* or *spline interpolation.*

Definition 6.18 *A* spline function S of degree k is a function such that

a) S is defined on an interval $[a, b]$,

b) $S \in C^{k-1}[a, b]$,

c) there are points $a = t_0 < t_1 < \ldots < t_n = b$ (called *knots*) such that S is a polynomial of degree at most k on each subinterval $[t_i, t_{i+1})$.

Example 1: The most commonly used spline function is the piecewise linear ($k = 1$) spline, i.e., given a knot sequence as in Definition 6.18, S is a linear function on each subinterval with continuous joints at the knots.

If we are given the data

x	0	1	2	3
y	1	0	-1	3

,

then we can let the knot sequence coincide with the data sites, i.e.,

$$t_i = i, \qquad i = 0, 1, 2, 3.$$

The piecewise linear spline interpolating the data is then given by the "connect-the-dots" approach, or in formulas

$$S(x) = \begin{cases} 1 - x & 0 \le x < 1, \\ 1 - x & 1 \le x < 2, \\ 4x - 9 & 2 \le x < 3. \end{cases}$$

This spline is displayed in Figure 7

Remark: Definition 6.18 does not make any statement about the relation between the location of the knots and the data sites for interpolation. We just observe that – for linear splines – it works to let the knots coincide with the data sites. We will discuss the general problem in Section 6.5.

Example 2: We use the same data as above, i.e.,

x	0	1	2	3
y	1	0	-1	3

,

but now we want to interpolate with a quadratic (C^1) spline. Again, we let the knots and data sites coincide, i.e., $t_i = x_i$, $i = 0, 1, 2, 3$.

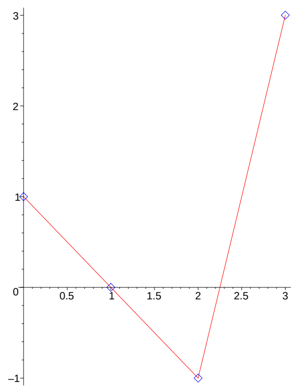

Figure 7: Graph of the linear interpolating spline of Example 1.

Now it is not so obvious what S will look like. From Definition 6.18 we know that S will be of the form

$$S(x) = \begin{cases} S_0(x) = a_0x^2 + b_0x + c_0, & 0 \le x < 1, \\ S_1(x) = a_1x^2 + b_1x + c_1, & 1 \le x < 2, \\ S_2(x) = a_2x^2 + b_2x + c_2, & 2 \le x < 3. \end{cases}$$

Since we have three quadratic pieces there are 9 parameters (a_j, b_j, c_j, $j = 0, 1, 2$) to be determined. To this end, we collect the conditions we have on S. Obviously, we have the 4 interpolation conditions

$$S(x_i) = y_i, \qquad i = 0, 1, 2, 3.$$

Moreover, S needs to satisfy the C^1 (and C^0) smoothness conditions of Definition 6.18 at the interior knots. That leads to four more conditions:

$$\begin{aligned} S_0(t_1) &= S_1(t_1), \\ S_1(t_2) &= S_2(t_2), \\ S_0'(t_1) &= S_1'(t_1), \\ S_1'(t_2) &= S_2'(t_2). \end{aligned}$$

Thus, we have a total of 8 conditions to determine the 9 free parameters. This leaves at least one undetermined parameter (provided the above conditions are linearly independent).

Hoping that the conditions above are indeed independent, we add one more (arbitrary) condition $S_0'(0) = -1$ to obtain a square linear system. Therefore, we need to solve

$$\begin{aligned} S_0(0) &= 1, \\ S_0(1) &= 0, \\ S_1(1) &= 0, \\ S_1(2) &= -1, \\ S_2(2) &= -1, \\ S_2(3) &= 3, \\ S_0'(1) - S_1'(1) &= 0, \end{aligned}$$

$$\begin{aligned} S_1'(2) - S_2'(2) &= 0, \\ S_0'(0) &= -1. \end{aligned}$$

Note that we have implemented the C^0 conditions at the two interior knots by stating the interpolation conditions for both adjoining pieces. In matrix form, the resulting linear system is

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 3 & 1 \\ 2 & 1 & 0 & -2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 1 & 0 & -4 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ a_1 \\ b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ -1 \\ 3 \\ 0 \\ 0 \\ -1 \end{bmatrix}.$$

The solution of this linear system is given by

$$\begin{gathered} a_0 = 0, \ b_0 = -1, \ c_0 = 1, \\ a_1 = 0, \ b_1 = -1, \ c_1 = 1, \\ a_2 = 5, \ b_2 = -21, \ c_2 = 21, \end{gathered}$$

or

$$S(x) = \begin{cases} 1 - x, & 0 \le x < 1, \\ 1 - x, & 1 \le x < 2, \\ 5x^2 - 21x + 21, & 2 \le x < 3. \end{cases}$$

This example is illustrated in the Maple worksheet `578_SplineInterpolation.mws` and a plot of the quadratic spline computed in Example 2 is also provided in Figure 8.

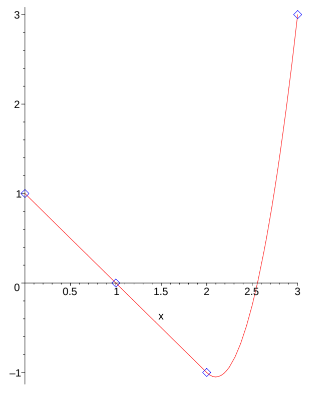

Figure 8: Graph of the quadratic interpolating spline of Example 2.

Remark: In order to efficiently evaluate a piecewise defined spline S at some point $x \in [t_0, t_n]$ one needs to be able to identify which polynomial piece to evaluate, i.e., determine in which subinterval $[t_i, t_{i+1})$ the evaluation point x lies. An algorithm (for linear splines) is given in the textbook on page 350.

Cubic Splines

Another very popular spline is the (C^2) cubic spline. Assume we are given $n+1$ pieces of data (x_i, y_i), $i = 0, 1, \ldots, n$ to interpolate. Again, we let the knot sequence $\{t_i\}$ coincide with the data sites. According to Definition 6.18 the spline S will consist of n cubic polynomial pieces with a total of $4n$ parameters.

The conditions prescribed in Definition 6.18 are

$n+1$ interpolation conditions,

$n-1$ C^0 continuity conditions at interior knots,

$n-1$ C^1 continuity conditions at interior knots,

$n-1$ C^2 continuity conditions at interior knots,

for a total of $4n-2$ conditions. Assuming linear independence of these conditions, we will be able to impose two additional conditions on S.

There are many possible ways of doing this. We will discuss:

1. $S''(t_0) = S''(t_n) = 0$, (so-called *natural* end conditions).
2. Other boundary conditions, such as
$$S'(t_0) = f'(t_0), \quad S'(t_n) = f'(t_n)$$
which lead to *complete* splines, or
$$S''(t_0) = f''(t_0), \quad S''(t_n) = f''(t_n).$$
In either case, f' or f'' needs to be provided (or estimated) as additional data.
3. The so-called "not-a-knot" condition.

Cubic Natural Spline Interpolation

To simplify the notation we introduce the abbreviation

$$z_i = S''(t_i), \qquad i = 0, 1, \ldots, n,$$

for the value of the second derivative at the knots. We point out that these values are *not* given as data, but are parameters to be determined.

Since S is cubic S'' will be linear. If we write this linear polynomial on the subinterval $[t_i, t_{i+1})$ in its Lagrange form we have

$$S_i''(x) = z_i \frac{t_{i+1} - x}{t_{i+1} - t_i} + z_{i+1} \frac{x - t_i}{t_{i+1} - t_i}.$$

With another abbreviation $h_i = t_{i+1} - t_i$ this becomes

$$S_i''(x) = \frac{z_i}{h_i}(t_{i+1} - x) + \frac{z_{i+1}}{h_i}(x - t_i). \tag{19}$$

Remark: By assigning the value z_i to both pieces joining together at t_i we will automatically enforce continuity of the second derivative of S.

Now, we obtain a representation for the piece S_i by integrating (19) twice:

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1} - x)^3 + \frac{z_{i+1}}{6h_i}(x - t_i)^3 + C(x - t_i) + D(t_{i+1} - x). \tag{20}$$

The interpolation conditions (for the piece S_i)

$$S_i(t_i) = y_i, \quad S_i(t_{i+1}) = y_{i+1}$$

yield a 2×2 linear system for the constants C and D. This leads to

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1}-x)^3 + \frac{z_{i+1}}{6h_i}(x-t_i)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right)(x-t_i) + \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right)(t_{i+1}-x). \tag{21}$$

(Note that it is easily verified that (21) satisfies the interpolation conditions.)

Once we have determined the unknowns z_i each piece of the spline S can be evaluated via (21). We have not yet employed the C^1 continuity conditions at the interior knots, i.e.,

$$S'_{i-1}(t_i) = S'_i(t_i), \qquad i = 1, 2, \ldots, n-1. \tag{22}$$

Thus, we have $n-1$ additional conditions. Since there are $n+1$ unknowns z_i, $i = 0, 1, \ldots, n$, we fix the second derivative at the endpoints to be zero, i.e.,

$$z_0 = z_n = 0.$$

These are the so-called *natural end conditions.* Differentiation of (21) leads to

$$S'_i(x) = -\frac{z_i}{2h_i}(t_{i+1} - x)^2 + \frac{z_{i+1}}{2h_i}(x - t_i)^2 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right) - \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right).$$

Using this expression in (22) results in

$$z_{i-1}h_{i-1} + 2(h_{i-1} + h_i)z_i + z_{i+1}h_i = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}), \quad i = 1, 2, \ldots, n-1.$$

This represents a symmetric tridiagonal system for the unknowns $z_1, \ldots, z_{n-1}$ of the form

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & 0 & \ldots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & \ldots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n-2} \\ r_{n-1} \end{bmatrix}.$$

Here $h_i = t_{i+1} - t_i$ and

$$r_i = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}), \quad i = 1, 2, \ldots, n-1.$$

Remark: The system matrix is even *diagonally dominant* since

$$2(h_{i-1} + h_i) > h_{i-1} + h_i$$

since all $h_i > 0$ due to the fact that the knots t_i are distinct and form an increasing sequence. Thus, a very efficient tridiagonal version of Gauss elimination without pivoting can be employed to solve for the missing z_i (see the textbook on page 353 for such an algorithm).

"Optimality" of Natural Splines

Among all smooth functions which interpolate a given function f at the knots $t_0, t_1, \ldots, t_n$, the natural spline is the smoothest, i.e.,

Theorem 6.19 *Let $f \in C^2[a,b]$ and $a = t_0 < t_1 < \ldots < t_n = b$. If S is the cubic natural spline interpolating f at t_i, $i = 0, 1, \ldots, n$, then*

$$\int_a^b [S''(x)]^2 \, dx \le \int_a^b [f''(x)]^2 \, dx.$$

Remark: The integrals in Theorem 6.19 can be interpreted as bending energies of a thin rod, or as "total curvatures" (since for small defections the curvature $\kappa \approx f''$). This optimality result is what gave rise to the name *spline*, since early ship designers used a piece of wood (called a spline) fixed at certain (interpolation) points to describe the shape of the ship's hull.

Proof: Define an auxiliary function $g = f - S$. Then $f = S + g$ and $f'' = S'' + g''$ or

$$(f'')^2 = (S'')^2 + (g'')^2 + 2S''g''.$$

Therefore,

$$\int_a^b (f''(x))^2 \, dx = \int_a^b (S''(x))^2 \, dx + \int_a^b (g''(x))^2 \, dx + \int_a^b 2S''(x)g''(x)dx.$$

Obviously,

$$\int_a^b (g''(x))^2 \, dx \ge 0,$$

so that we are done is we can show that also

$$\int_a^b 2S''(x)g''(x)dx \ge 0.$$

To this end we break the interval $[a, b]$ into the subintervals $[t_{i-1}, t_i]$, and get

$$\int_a^b S''(x)g''(x)dx = \sum_{i=1}^n \int_{t_{i-1}}^{t_i} S''(x)g''(x)dx.$$

Now we can integrate by parts (with $u = S''(x)$, $dv = g''(x)dx$) to obtain

$$\sum_{i=1}^n \left\{ \left[S''(x)g'(x)\right]_{t_{i-1}}^{t_i} - \int_{t_{i-1}}^{t_i} S'''(x)g'(x)dx \right\}.$$

The first term is a telescoping sum so that

$$\sum_{i=1}^{n} \left[S''(x)g'(x)\right]_{t_{i-1}}^{t_i} = S''(t_n)g'(t_n) - S''(t_0)g'(t_0) = 0$$

due to the natural end conditions of the spline S.

This leaves

$$\int_a^b S''(x)g''(x)dx = -\sum_{i=1}^{n} \int_{t_{i-1}}^{t_i} S'''(x)g'(x)dx.$$

However, since S_i is a cubic polynomial we know that $S'''(x) \equiv c_i$ on $[t_{i-1}, t_i)$. Thus,

$$\begin{aligned}\int_a^b S''(x)g''(x)dx &= -\sum_{i=1}^{n} c_i \int_{t_{i-1}}^{t_i} g'(x)dx \\ &= -\sum_{i=1}^{n} c_i \left[g(x)\right]_{t_{i-1}}^{t_i} = 0.\end{aligned}$$

The last equation holds since $g(t_i) = f(t_i) - S(t_i)$, $i = 0, 1, \ldots, n$, and S interpolates f at the knots t_i. ♠

Remark: Cubic natural splines should *not be considered the natural choice for cubic spline interpolation.* This is due to the fact that one can show that the (rather arbitrary) choice of natural end conditions yields an interpolation error estimate of only $\mathcal{O}(h^2)$, where $h = \max\limits_{i=1,\ldots,n} \Delta t_i$ and $\Delta t_i = t_i - t_{i-1}$. This needs to be compared to the estimate of $\mathcal{O}(h^4)$ obtainable by cubic polynomials as well as other cubic spline methods.

Cubic Complete Spline Interpolation

The derivation of cubic complete splines is similar to that of the cubic natural splines. However, we impose the additional end constraints

$$S'(t_0) = f'(t_0), \qquad S'(t_n) = f'(t_n).$$

This requires additional data (f' at the endpoints), but can be shown to yields an $\mathcal{O}(h^4)$ interpolation error estimate.

Moreover, an energy minimization theorem analogous to Theorem 6.19 also holds since

$$\begin{aligned}\sum_{i=1}^{n} \left[S''(x)g'(x)\right]_{t_{i-1}}^{t_i} &= S''(t_n)g'(t_n) - S''(t_0)g'(t_0) \\ &= S''(t_n)\left(f'(t_n) - S'(t_n)\right) - S''(t_0)\left(f'(t_0) - S'(t_0)\right) = 0\end{aligned}$$

by the end conditions.

Not-a-Knot Spline Interpolation

One of the most effective cubic spline interpolation methods is obtained by choosing the knots *different from the data sites.* In particular, if the data is of the form

x_0	x_1	x_2	$\ldots$	x_{n-1}	x_n
y_0	y_1	y_2	$\ldots$	y_{n-1}	y_n

,

then we take the $n-1$ knots as

$$\begin{array}{c|c|c|c|c|c} x_0 & x_2 & x_3 & \dots & x_{n-2} & x_n \\ \hline t_0 & t_1 & t_2 & \dots & t_{n-3} & t_{n-2} \end{array},$$

i.e., the data sites x_1 and x_{n-1} are "not-a-knot".

The knots now define $n-2$ cubic polynomial pieces with a total of $4n-8$ coefficients. On the other hand, there are $n+1$ interpolation conditions together with three sets of $(n-3)$ smoothness conditions at the interior knots. Thus, the number of conditions is equal to the number of unknown coefficients, and no additional (arbitrary) conditions need to be imposed to solve the interpolation problem.

One can interpret this approach as using only one cubic polynomial piece to represent the first two (last two) data segments.

The cubic not-a-knot spline has an $\mathcal{O}(h^4)$ interpolation error, and requires no additional data. More details can be found in the book "A Practical Guide to Splines" by Carl de Boor.

<u>Remarks:</u>

1. If we are given also derivative information at the data sites x_i, $i = 0, 1, \dots, n$, then we can perform piecewise cubic Hermite interpolation (see Section 6.3). The resulting function will be C^1 continuous, and one can show that the interpolation error is $\mathcal{O}(h^4)$. However, this function is *not* considered a spline function since it does not have the required smoothness.

2. There are also piecewise cubic interpolation methods that *estimate* derivative information at the data sites, i.e., no derivative information is provided as data. Two such (local) methods are named after Bessel (yielding an $\mathcal{O}(h^3)$ interpolation error) and Akima (with an $\mathcal{O}(h^2)$ error). Again, they are not spline functions as they are only C^1 smooth.

6.5 B-Splines

B-splines will generate local bases for the spaces of spline functions defined in the previous section in Definition 6.18. Local bases are important for reasons of efficiency. Moreover, we will see that there are efficient recurrence relations for the coefficients of these basis functions.

B-splines were first introduced to approximation theory by Iso Schoenberg in 1946. The name "*B*-spline", however, was not used until 1967. On the other hand, the functions known as *B*-splines today came up as early as 1800 in work by Laplace and Lobachevsky as convolutions of probability density functions.

The main books on *B*-splines are "A Practical Guide to Splines" by Carl de Boor (1978) and "Spline Functions: Basic Theory" by Larry Schumaker (1980). The development presented in our textbook (which we follow here) is not the traditional one (based on the use of divided differences) given in the two main monographs, but rather based on the paper "*B*-splines without divided differences" by Carl de Boor and Klaus Höllig (1987). There are alternate derivations based on the idea of knot insertion (mostly for

CAGD purposes) by Wolfgang Boehm (1980), and on a method called "blossoming" by Lyle Ramshaw (1987).

According to Definition 6.18 the space of piecewise polynomial spline functions is determined by the polynomial degree and a (bi-infinite) *knot sequence*

$$\ldots < t_{-2} < t_{-1} < t_0 < t_1 < t_2 < \ldots .$$

In practice, the knot sequence is usually finite, and later on knots with varying multiplicities will be allowed.

***B*-Splines of Degree 0**

Definition 6.20 *Given an interval* $[t_i, t_{i+1})$*, the* B-spline of degree 0 *is defined as*

$$B_i^0(x) = \begin{cases} 1 & x \in [t_i, t_{i+1}), \\ 0 & \textit{otherwise.} \end{cases}$$

The graphs of two different B-splines of degree 0 for a given knot sequence are presented in Figure 9.

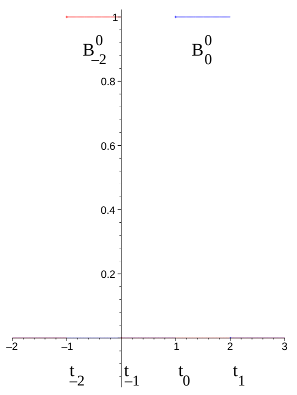

Figure 9: Graph of degree 0 B-splines B_{-2}^0 and B_0^0.

Remark: From Definition 6.20 we see that B_i^0 is just the *characteristic function* of the interval $[t_i, t_{i+1})$.

The most important properties of degree-0 B-splines are

1. $\text{supp} B_i^0 = [t_i, t_{i+1})$, i.e., the B_i^0 have local support.
2. $B_i^0(x) \geq 0$ for all i and x, i.e., the B_i^0 are nonnegative.
3. All B_i^0 are right-continuous on $\mathbb{R}$.
4. The B_i^0 form a *partition of unity*, i.e.,

$$\sum_{i=-\infty}^{\infty} B_i^0(x) = 1, \qquad \forall x \in \mathbb{R} .$$

This can be seen by letting j be the index such that $x \in [t_j, t_{j+1})$. Then

$$B_i^0(x) = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases}$$

5. $\{B_i^0\}$ form a *basis* for the space of piecewise constant spline functions based on the knot sequence $\{t_i\}$, i.e., if
$$S(x) = c_i, \qquad i \in [t_i, t_{i+1}),$$
then
$$S(x) = \sum_{i=-\infty}^{\infty} c_i B_i^0(x).$$

Remark: If the knot sequence is finite, then the spline function S will have a finite-sum representation due to the local support of the B-splines.

Higher-Degree B-Splines

Higher-degree B-splines are defined recursively.

Definition 6.21 *Given a knot sequence* $\{t_i\}$*,* B-splines of degree k*,* $k = 1, 2, 3, \ldots$*, are defined as*
$$B_i^k(x) = \frac{x - t_i}{t_{i+k} - t_i} B_i^{k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1}^{k-1}(x).$$

Remark: The traditional way to define B-splines is as divided differences of the truncated power function, and then the recurrence relation follows as a theorem from the recurrence relation for divided differences.

Example 1: For $k = 1$ we get linear B-splines as
$$B_i^1(x) = \frac{x - t_i}{t_{i+1} - t_i} B_i^0(x) + \frac{t_{i+2} - x}{t_{i+2} - t_{i+1}} B_{i+1}^0(x).$$
Thus, B_i^1 has support on the combined interval $[t_i, t_{i+2})$. Its graph is displayed in Figure 10.

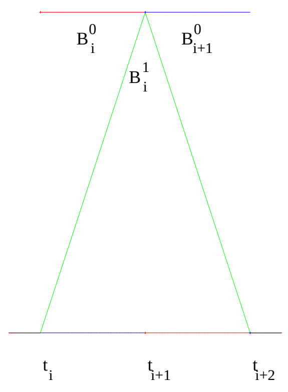

Figure 10: Graph of the linear B-spline B_i^1 along with the lower-degree B-splines B_i^0 and B_{i+1}^0.

Example 2: For $k = 2$ we get quadratic B-splines as
$$B_i^2(x) = \frac{x - t_i}{t_{i+2} - t_i} B_i^1(x) + \frac{t_{i+3} - x}{t_{i+3} - t_{i+1}} B_{i+1}^1(x).$$
Thus, B_i^2 has support on the combined interval $[t_i, t_{i+3})$. Its graph is displayed in Figure 11.

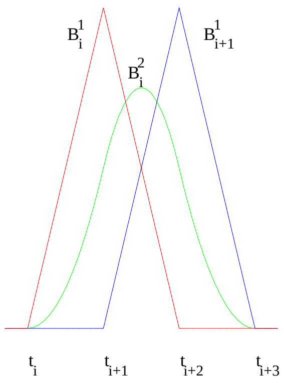

Figure 11: Graph of the quadratic B-spline B_i^2 along with the lower-degree B-splines B_i^1 and B_{i+1}^1.

The properties of higher-degree B-splines are very similar to those for B-splines of degree 0 listed earlier:

1. $\text{supp} B_i^k = [t_i, t_{i+k+1})$, i.e., the B_i^k have local support. This follows by a simple induction argument from the recursive Definition 6.21 and the support of B-splines of degree 0.

2. $B_i^k(x) \geq 0$ for all i and x, i.e., the B_i^k are nonnegative. This also follows by induction.

3. For $k \geq 1$, $B_i^k \in C^{k-1}(\mathbb{R})$. We will later derive a formula for derivatives of B-splines that will establish this fact.

4. The B_i^k form a *partition of unity*, i.e.,

$$\sum_{i=-\infty}^{\infty} B_i^k(x) = 1, \qquad \forall x \in \mathbb{R}.$$

   This will be proved later.

5. The B-splines $\{B_i^k\}_{i=0}^k$ restricted to the interval $[t_k, t_{k+1})$ form a *basis* for the space of polynomials of degree k on $[t_k, t_{k+1})$. We will say more about this later.

6. For any given knot sequence, the B-splines $\{B_i^k\}$ of degree k form a basis for the space of spline functions of degree k defined on the same knot sequence, i.e., the spline function S can be represented as

$$S(x) = \sum_{i=-\infty}^{\infty} c_i^k B_i^k(x) \tag{23}$$

   with appropriate coefficients c_i^k. We will provide more details later.

Remark: The combination of properties 1 and 6 is very useful for computational purposes. In fact, for finite knot sequences (as in the spline interpolation problems of the previous section), the B-splines will form a finite-dimensional basis for the spline functions of degree k, and due to property 1 the linear system obtained by applying the interpolation conditions (1) to a (finite) B-spline expansion of the form (23) will be banded and non-singular. Moreover, changing the data locally will only have a local effect on the B-spline coefficients, i.e., only a few of the coefficients will change.

Evaluation of Spline Functions via B-Splines

We now fix k, and assume the coefficients c_i^k are given. Then, using the recursion of Definition 6.21 and an index transformation on the bi-infinite sum,

$$\begin{aligned}
\sum_{i=-\infty}^{\infty} c_i^k B_i^k(x) &= \sum_{i=-\infty}^{\infty} c_i^k \left[\frac{x-t_i}{t_{i+k}-t_i} B_i^{k-1}(x) + \frac{t_{i+k+1}-x}{t_{i+k+1}-t_{i+1}} B_{i+1}^{k-1}(x)\right] \\
&= \sum_{i=-\infty}^{\infty} c_i^k \frac{x-t_i}{t_{i+k}-t_i} B_i^{k-1}(x) + \underbrace{\sum_{i=-\infty}^{\infty} c_i^k \frac{t_{i+k+1}-x}{t_{i+k+1}-t_{i+1}} B_{i+1}^{k-1}(x)}_{= \sum_{i=-\infty}^{\infty} c_{i-1}^k \frac{t_{i+k}-x}{t_{i+k}-t_i} B_i^{k-1}(x)} \\
&= \sum_{i=-\infty}^{\infty} \underbrace{\left[c_i^k \frac{x-t_i}{t_{i+k}-t_i} + c_{i-1}^k \frac{t_{i+k}-x}{t_{i+k}-t_i}\right]}_{=:\, c_i^{k-1}} B_i^{k-1}(x).
\end{aligned}$$

This gives us a recursion formula for the coefficients c_i^k. We now continue the recursion until the degree of the B-splines has been reduced to zero. Thus,

$$\begin{aligned}
\sum_{i=-\infty}^{\infty} c_i^k B_i^k(x) &= \sum_{i=-\infty}^{\infty} c_i^{k-1} B_i^{k-1}(x) \\
&= \ldots = \sum_{i=-\infty}^{\infty} c_i^0 B_i^0(x) = c_j^0,
\end{aligned}$$

where j is the index such that $x \in [t_j, t_{j+1})$.

The recursive procedure just described is known as *de Boor algorithm.* It is used to evaluate a spline function S at the point x. In fact,

Theorem 6.22 *Let*

$$S(x) = \sum_{i=-\infty}^{\infty} c_i^k B_i^k(x)$$

with known coefficients c_i^k. Then $S(x) = c_j^0$, where j is such that $x \in [t_j, t_{j+1})$, and the coefficients are computed recursively via

$$c_i^{k-1} = c_i^k \frac{x-t_i}{t_{i+k}-t_i} + c_{i-1}^k \frac{t_{i+k}-x}{t_{i+k}-t_i}. \tag{24}$$

Remark: The computation of the coefficients can be arranged in a triangular tableau where each coefficient is a convex combination of the two coefficients to its left (in the same row, and the row below) as stated in (24):

$$\begin{array}{cccccc}
c_j^k & c_j^{k-1} & c_j^{k-2} & \cdots & c_j^1 & c_j^0 \\
c_{j-1}^k & c_{j-1}^{k-1} & c_{j-1}^{k-2} & \cdots & c_{j-1}^1 & \\
\vdots & \vdots & \vdots & & & \\
c_{j-k+2}^k & c_{j-k+2}^{k-1} & c_{j-k+2}^{k-2} & & & \\
c_{j-k+1}^k & c_{j-k+1}^{k-1} & & & & \\
c_{j-k}^k & & & & &
\end{array}$$

Now we are able to show the partition of unity property. Consider

$$S(x) = \sum_{i=-\infty}^{\infty} c_i^k B_i^k(x)$$

with all coefficients $c_i^k = 1$. Then, from (24),

$$\begin{aligned}
c_i^{k-1} &= c_i^k \frac{x - t_i}{t_{i+k} - t_i} + c_{i-1}^k \frac{t_{i+k} - x}{t_{i+k} - t_i} \\
&= \frac{(x - t_i) + (t_{i+k} - x)}{t_{i+k} - t_i} = 1.
\end{aligned}$$

This relation holds throughout the entire de Boor algorithm so that

$$S(x) = \sum_{i=-\infty}^{\infty} B_i^k(x) = c_j^0 = 1.$$

Derivatives of B-splines

Theorem 6.23 *For any $k \geq 2$*

$$\frac{d}{dx} B_i^k(x) = \frac{k}{t_{i+k} - t_i} B_i^{k-1}(x) - \frac{k}{t_{i+k+1} - t_{i+1}} B_{i+1}^{k-1}(x). \tag{25}$$

Remark: The derivative formula (25) also holds for $k = 1$ except at the knots $x = t_i, t_{i+1}, t_{i+2}$, where B_i^1 is not differentiable.

Proof: By induction. The cases $k = 1, 2$ are done in homework problem 6.5.14. ♠

We are now able to establish the smoothness of B-splines.

Theorem 6.24 *For $k \geq 1$, $B_i^k \in C^{k-1}(\mathbb{R})$.*

Proof: We use induction on k. The case $k = 1$ is clear. Now we assume the statement holds for k, and show it is also true for $k + 1$, i.e., we show $B_i^{k+1} \in C^k(\mathbb{R})$.

From (25) we know $\frac{d}{dx} B_i^{k+1}$ is a linear combination of B_i^k and B_{i+1}^k which, by the induction hypothesis are both in $C^{k-1}(\mathbb{R})$. Therefore, $\frac{d}{dx} B_i^{k+1} \in C^{k-1}(\mathbb{R})$, and thus $B_i^{k+1} \in C^k(\mathbb{R})$. ♠

Remark: Simple integration formulas for B-splines also exist. See, e.g., page 373 of the textbook.

Linear Independence

We now return to property 5 listed earlier. We will be slightly more general and consider the set $\{B_j^k, B_{j+1}^k, \ldots, B_{j+k}^k\}$ of $k+1$ B-splines of degree k.

Example 1: Figure 12 shows the two linear B-splines B_j^1 and B_{j+1}^1. From the figure it is clear that they are linearly independent on the intersection of their supports $[t_{j+1}, t_{j+2})$. Since the dimension of the space of linear polynomials (restricted to this interval) is two, and the two B-splines are linearly independent they form a basis for the space of linear polynomials restricted to $[t_{j+1}, t_{j+2})$.

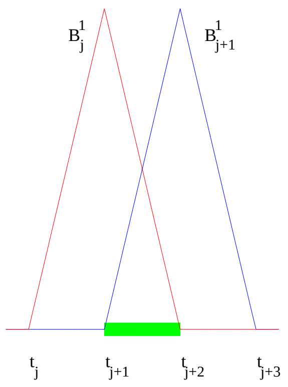

Figure 12: Graph of the linear B-splines B_j^1 and B_{j+1}^1.

Example 2: Figure 13 shows the three quadratic B-splines B_j^2, B_{j+1}^2 and B_{j+2}^2. Again, the B-splines are linearly independent on the intersection of their supports $[t_{j+2}, t_{j+3})$, and form a basis for the space of quadratic polynomials restricted to $[t_{j+2}, t_{j+3})$.

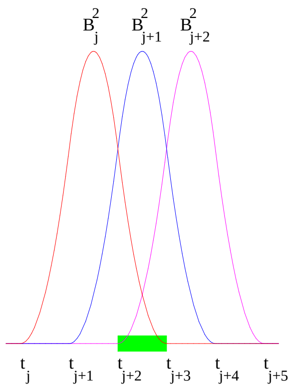

Figure 13: Graph of the quadratic B-splines B_j^2, B_{j+1}^2 and B_{j+2}^2.

For the general case we have

Lemma 6.25 *The set $\{B_j^k, B_{j+1}^k, \dots, B_{j+k}^k\}$ of $k+1$ B-splines of degree k is linearly independent on the interval $[t_{j+k}, t_{j+k+1})$ and forms a basis for the space of polynomials of degree k there.*

Proof: By induction. See the textbook on page 373. ♠

Now we assume that we have a finite knot sequence $\{t_0, t_1, \dots, t_n\}$ (as we had in the previous section on spline interpolation). Then we can find a finite-dimensional basis of B-splines of degree k for the space of spline functions of degree k defined on the given knot sequence. As a first step in this direction we have

Lemma 6.26 *Given a knot sequence $\{t_0, t_1, \dots, t_n\}$, the set $\{B_{-k}^k, B_{-k+1}^k, \dots, B_{n-1}^k\}$ of $n+k$ B-splines of degree k is linearly independent on $[t_0, t_n)$.*

Proof: According to the definition of linear independence, we define a spline function

$$S(x) = \sum_{i=-k}^{n-1} c_i B_i^k(x),$$

and show that, for any $x \in [t_0, t_n)$, $S(x) = 0$ is only possible if all coefficients c_i, $i = -k, \dots, n-1$, are zero.

The main idea is to break the interval $[t_0, t_n)$ into smaller pieces $[t_j, t_{j+1})$ and show linear independence on each of the smaller intervals.

If we consider the interval $[t_0, t_1)$, then by Lemma 6.25 only the $k+1$ B-splines $\{B_{-k}^k, \dots, B_0^k\}$ are "active" on this interval, and linearly independent there.

Therefore, restricted to $[t_0, t_1)$ we have

$$S|_{[t_0,t_1)} = \sum_{i=-k}^{0} c_i B_i^k = 0 \quad \Longrightarrow \quad c_i = 0,\ i = -k, \dots, 0.$$

Similarly, we can argue that

$$S|_{[t_j,t_{j+1})} = \sum_{i=j-k}^{j} c_i B_i^k = 0 \quad \Longrightarrow \quad c_i = 0,\ i = j-k, \dots, j.$$

Now, for any $x \in [t_j, t_{j+1})$ the coefficients of all B-splines whose supports intersect the interval are zero. Finally, since x is arbitrary, all coefficients must be zero. ♠

<u>Remark:</u> The representation of polynomials in terms of B-splines is given by *Marsden's identity*

$$\sum_{i=-\infty}^{\infty} \prod_{j=1}^{k} (t_{i+j} - s) B_i^k(x) = (x-s)^k.$$

This is covered in homework problems 6.5.9–12.

6.6 More B-Splines

We introduce the notation $\mathcal{S}_n^k$ for the space of spline functions of degree k defined on the knot sequence $\{t_0, t_1, \ldots, t_n\}$. Our goal is to find a basis for $\mathcal{S}_n^k$.

A first basis is given by so-called *truncated power functions*

$$(x)_+^k = \begin{cases} x^k, & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases} \tag{26}$$

Figure 14 shows graphs of the three truncated power functions $(x)_+$, $(x-1)_+^2$, and $(x-2)_+^3$.

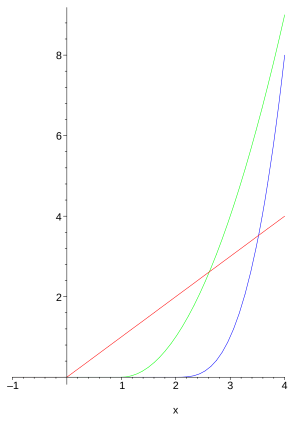

Figure 14: Graph of the truncated powers $(x)_+$, $(x-1)_+^2$, and $(x-2)_+^3$.

Theorem 6.27 *Let $\mathcal{S}_n^k$ be the space of spline functions of degree k with knots $t_0, t_1, \ldots, t_n$. Any $S \in \mathcal{S}_n^k$ can be written as*

$$S(x) = \sum_{i=0}^{k} a_i x^i + \sum_{i=1}^{n-1} b_i (x - t_i)_+^k \tag{27}$$

with appropriate coefficients a_i and b_i.

Proof: We start with $x \in [t_0, t_1)$. In this case

$$(x - t_i)_+ = 0, \qquad i = 1, \ldots, n-1.$$

Thus, the expansion (27) reduces to

$$S(x) = \sum_{i=0}^{k} a_i x^i,$$

and since S is a polynomial p_0 of degree k on $[t_0, t_1)$ the coefficients a_i are uniquely determined.

Now, on the next interval $[t_1, t_2)$ the spline function S is again a (different) polynomial p_1 of degree k. Moreover, it must join the polynomial piece on the previous interval with C^{k-1} smoothness, i.e.,

$$(p_1 - p_0)^{(r)}(t_1) = 0, \qquad r = 0, \ldots, k-1.$$

Therefore, $\deg(p_1 - p_0) \le k$, and we have the representation

$$(p_1 - p_0)(x) = b_1(x - t_1)^k,$$

and

$$S(x) = \sum_{i=0}^{k} a_i x^i + b_1(x - t_1)_+^k, \qquad x \in [t_0, t_2).$$

Expansion (27) can now be obtained by continuing this process interval by interval. ♠

Example: In an earlier example we determined

$$S(x) = \begin{cases} 1 - x & 0 \le x < 1, \\ 1 - x & 1 \le x < 2, \\ 4x - 9 & 2 \le x < 3. \end{cases}$$

as the piecewise linear interpolant of the data

x	0	1	2	3
y	1	0	-1	3

.

We take knots $t_i = i$, $i = 0, 1, 2, 3$, and $k = 1$. According to Theorem 6.27 all continuous piecewise linear functions S on $[0, 3]$ can be represented in the form

$$S(x) = a_0 + a_1 x + b_1(x - 1)_+ + b_2(x - 2)_+.$$

For this example the truncated power expansion can easily be determined as

$$S(x) = 1 - x + 5(x - 2)_+,$$

i.e., $a_0 = 1$, $a_1 = -1$, $b_1 = 0$, and $b_2 = 5$.

The basis functions for this example are plotted in Figure 15.

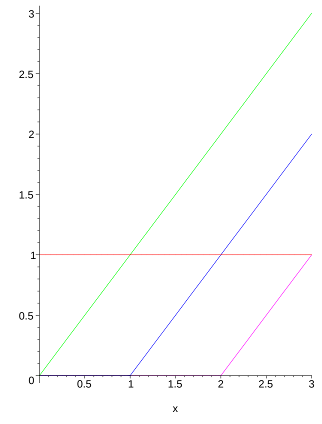

Figure 15: Basis functions of the example.

Remark: The truncated power basis for $\mathcal{S}_n^k$ is not a good choice for computational purposes (just as the monomials were not a good choice to represent polynomials). The main reason is that this basis leads to ill-conditioned system matrices and numerical instabilities. Therefore, the truncated power basis can be viewed as the analogue of the monomial basis for the space of polynomials in the case of piecewise polynomials or spline functions. Just as the Newton basis and Lagrange basis gave various computational advantages (e.g., better conditioned interpolation matrices), we need to find a basis for the space of spline functions that yields similar advantages.

Corollary 6.28 *The dimension of the space $\mathcal{S}_n^k$ of spline functions of degree k with knots $t_0, t_1, \ldots, t_n$ is*

$$\dim \mathcal{S}_n^k = n + k.$$

Proof: There are $(k+1) + (n-1) = n + k$ basis functions in (27). ♠

A Better Basis for $\mathcal{S}_n^k$

Theorem 6.29 *Any spline function $S \in \mathcal{S}_n^k$ can be represented with respect to the B-spline basis*

$$\left\{ B_i^k|_{[t_0,t_n]} : i = -k, \ldots, n-1 \right\}. \tag{28}$$

Proof: According to Corollary 6.28 we need $n+k$ basis functions. The set (28) contains $n+k$ functions in $\mathcal{S}_n^k$, and according to Lemma 6.26 they are linearly independent on $[t_0, t_n]$. ♠

Figure 16 shows the B-spline basis for the spline space of the previous example.

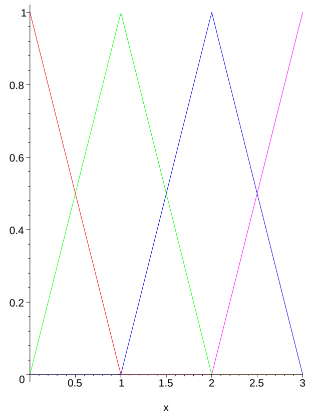

Figure 16: B-spline basis functions for the example.

The construction of the linear interpolating spline in B-spline representation is illustrated in the Maple worksheet `578_SplineInterpolation.mws`. Since the basis functions for this special case are cardinal functions we have

$$S(x) = B_0^1(x) - B_2^1(x) + 3B_3^1(x)$$

since the knots are given by $t_i = i$, $i = 0, 1, 2, 3$.

Remark: Besides being a numerically stable basis for $\mathcal{S}_n^k$ the B-splines are also a local basis. This fact was established in the previous section. Therefore, computational efficiency must be counted as another major advantage of the B-spline representation of spline functions.

Knot Locations vs. Data Sites

Earlier we chose (except in the case of not-a-knot splines) the knots to coincide with the data sites of the data to be interpolated. This need not be so in general. The following *Schoenberg-Whitney* Theorem gives a beautiful characterization of all possible knot placements with respect to the data sites.

Theorem 6.30 *Assume $x_1 < x_2 < \ldots < x_n$ and*

$$S(x) = \sum_{j=1}^{n} c_j B_j^k(x),$$

We can use S to interpolate arbitrary data given at the points x_i, $i = 1, \ldots, n$, if and only if the knots t_i are chosen so that there is at least one data site x_i in the support of every B-spline.

Proof: The proof of this theorem is fairly complicated and long and presented in the form of Lemmas 1, 2 and Theorems 2 and 3 on pages 378–383 of the textbook. ♠

Remarks:

1. From an algebraic point of view, the Schoenberg-Whitney Theorem says that the interpolation matrix B with entries $B_j^k(x_i)$ (cf. Section 6.0) should not have any zeros on its diagonal.
2. Theorem 6.30 does not guarantee a unique interpolant. In fact, as we saw in the examples for quadratic and cubic spline interpolation in Section 6.4, additional conditions may be required to ensure uniqueness.
3. Common choices for the knot locations are
   - For linear splines, choose $t_i = x_i$, $i = 1, \ldots, n$.
   - For $k > 1$, choose
     - the knots at the node averages,
     - the nodes at the knot averages.

Example: We illustrate some possible relative knot placements in the Maple worksheet `578_SplineInterpolation.mws`.

Approximation by Splines

Recall that the Weierstrass Approximation Theorem states that any continuous function can be approximated arbitrarily closely by a polynomial (of sufficiently high degree) on $[a, b]$.

We will now show that splines (of *fixed* degree k, but with sufficiently many knots) do the same.

Definition 6.31 *Let f be defined on $[a, b]$.*

$$\omega(f; h) = \max_{|s-t| \le h} |f(s) - f(t)|$$

is called the modulus of continuity *of f.*

Remarks:

1. If f is continuous on $[a, b]$, then it is also uniformly continuous there, i.e., for any $\epsilon > 0$ there exists a $\delta > 0$ such that for all $s, t \in [a, b]$
$$|s - t| < \delta \quad \Longrightarrow \quad |f(s) - f(t)| < \epsilon.$$
Therefore, $\omega(f; \delta) = \max_{|s-t|\le\delta} |f(s) - f(t)| \le \epsilon$, and for any continuous f we have
$$\omega(f; h) \to 0 \quad \text{as} \quad h \to 0.$$

2. If f is Lipschitz continuous with Lipschitz constant λ on $[a, b]$, i.e.,
$$|f(s) - f(t)| \le \lambda |s - t|,$$
then $\omega(f; h) \le \lambda h$.

3. If f is differentiable with $|f'(t)| \le M$ on $[a, b]$, then we also have $\omega(f; h) \le Mh$, but its rate of decay to zero as $h \to 0$ depends on the maximum of the derivative of f.

4. By comparing how fast $\omega(f; h)$ tends to zero for $h \to 0$ the modulus of continuity enables us to measure varying degrees of continuity.

An important property of the modulus of continuity is its so-called subadditivity, i.e.,
$$\omega(f; kh) \le k\omega(f; h).$$
We will use this property below, and you will prove it in homework problem 6.6.20.

The approximation power of splines can be obtained from the following *Whitney-type* theorem.

Theorem 6.32 *Let f be a function defined on $[t_0, t_n]$, and assume knots $\ldots < t_{-2} < t_{-1} < t_0 < t_1 < t_2 < \ldots$ are given, such that*
$$h = \max_{-k\le i\le n+1} |t_i - t_{i-1}|$$
denotes the meshsize. Then the spline function
$$s(x) = \sum_{i=-\infty}^{\infty} f(t_{i+2}) B_i^k(x)$$
satisfies
$$\max_{t_0\le x\le t_n} |f(x) - s(x)| \le k\omega(f; h).$$

Remark: If k is fixed, then for a continuous f we have

$$\omega(f;h) \to 0 \quad \text{as} \quad h \to 0,$$

and therefore the Weierstrass-like approximation property of splines is established provided we add sufficiently many knots to let the meshsize h go to zero.

Proof: Using the partition of unity property and nonnegativity of B-splines of degree k we have

$$\begin{aligned} |f(x) - s(x)| &= \Big| f(x) \underbrace{\sum_{i=-\infty}^{\infty} B_i^k(x)}_{=1} - \sum_{i=-\infty}^{\infty} f(t_{i+2}) B_i^k(x) \Big| \\ &\le \sum_{i=-\infty}^{\infty} |f(x) - f(t_{i+2})| \underbrace{B_i^k(x)}_{\ge 0}. \end{aligned}$$

Next we take j to be an index such that $x \in [t_j, t_{j+1})$. Then the bi-infinite sum can be replaced by a finite sum over the active B-splines

$$\begin{aligned} |f(x) - s(x)| &\le \sum_{i=j-k}^{j} |f(x) - f(t_{i+2})| B_i^k(x) \\ &\le \max_{j-k \le i \le j} |f(x) - f(t_{i+2})| \underbrace{\sum_{i=j-k}^{j} B_i^k(x)}_{=1}. \end{aligned}$$

Now, by the special choice of j we have $x \ge t_j$, as well as $i \le j$, and therefore

$$t_{i+2} - x \le t_{j+2} - t_j \le 2h.$$

On the other hand, since $x < t_{j+1}$ and $i \ge j - k$

$$x - t_{i+2} \le t_{j+1} - t_{j-k+2} \le kh.$$

Summarizing, we have

$$\max_{j-k \le i \le j} |f(x) - f(t_{i+2})| \le \max_{|x - t_{i+2}| \le kh} |f(x) - f(t_{i+2})| = \omega(f, kh),$$

and therefore, by the subadditivity of ω,

$$|f(x) - s(x)| \le \omega(f; kh) \le k\omega(f; h).$$

♠

Remarks:

1. The error bound in Theorem 6.32 does not say anything about interpolating splines. However, similar results hold for that case also.

2. More general results also exist taking into account higher smoothness of f, with the general statement being that smoother functions can be approximated at a faster rate. However, these results usually include a *saturation* statement, i.e., an increased rate of approximation is only possible up to a certain amount of smoothness. For example, for linear interpolating splines we have

   (a) If $f \in C[a,b]$ then $\|f - s\|_\infty \le \omega(f;h)$.

   (b) If $f \in C^1[a,b]$ then $\|f - s\|_\infty \le h\|f'\|_\infty$.

   (c) If $f \in C^2[a,b]$ then $\|f - s\|_\infty \le \frac{h^2}{8}\|f''\|_\infty$.

   (d) If $f \in C^3[a,b]$ then the bound is the same as for C^2 functions, i.e., saturation sets in.

B-Splines with Multiple Knots

If we allow multiple knots, then the degree of smoothness is reduced at those points. Figure 17 shows cubic B-splines B_0^3 with knot sequences $\{0,1,2,3,4\}$, $\{0,0,2,3,4\}$, $\{0,0,0,3,4\}$, $\{0,0,0,0,4\}$, $\{0,1,1,3,4\}$, and $\{0,1,1,1,4\}$. We can clearly see how the triple knots have reduced the smoothness to C^0 continuity at that point, and how the quadruple knot introduces even a discontinuity.

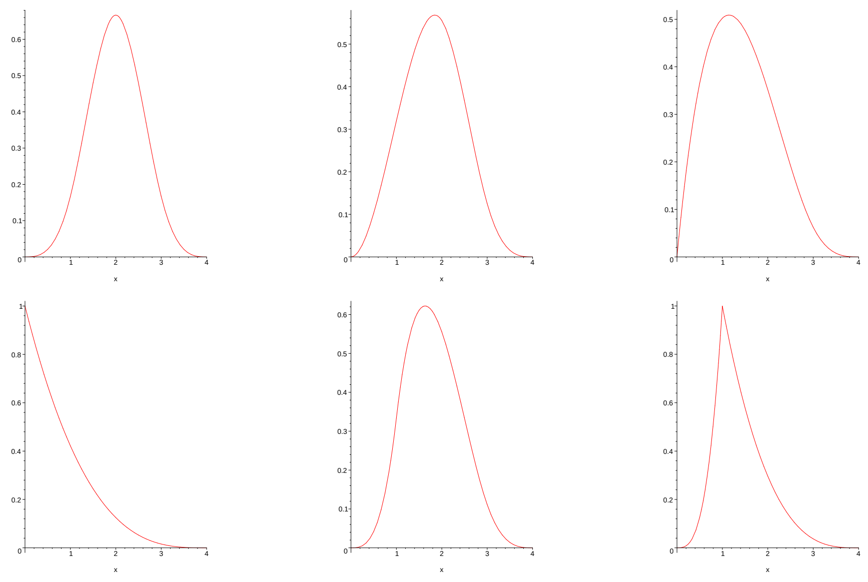

Figure 17: Cubic B-splines whose knot sequences allow various multiplicities.

Remark: B-splines of degree k with $k+1$-fold knots reduce to the Bernstein basis polynomials discussed earlier. In particular, the B-splines B_i^k, $i = 0,\dots,k$, associated with the knot sequence

$$\underbrace{0,\dots,0}_{k+1},\underbrace{1,\dots,1}_{k+1}$$

are exactly the Bernstein basis polynomials $\binom{k}{i}x^i(1-x)^{k-i}$ (see, e.g., Figure 4 for the case $k=3$). This connection is heavily exploited in CAGD where Bézier curves can thereby be expressed in terms of B-splines (with multiple knots). More details are given, e.g., in the books by Farin, Hoschek and Lasser, and the new book "Bézier and B-Spline Techniques" by Prautzsch, Boehm and Paluszny.

6.8 Best Approximation: Least Squares Theory

Given a normed linear space E and some function $f \in E$, as well as a subspace $G \subset E$ (the so-called *approximation space*) the *best approximation problem* is to find a function $g \in G$ such that

$$\|f-g\| \to \min.$$

A function g which minimizes this quantity is called the *best approximation of f from G*.

We will consider E to be an inner product space with one of the following two inner products:

1. For $f, g \in E$ with $E = C[a,b]$ or $E = L^2[a,b]$
$$\langle f, g\rangle = \int_a^b f(x)g(x)w(x)dx$$
with $w > 0$ a continuous *weight function.*

2. For $x, y \in E = \mathbb{R}^n$
$$\langle x, y\rangle = \sum_{i=1}^n x_i y_i.$$

We restrict our attention to inner product spaces because it is easy to characterize best approximation in those spaces.

Theorem 6.33 *Let E be an inner product space. Then $g \in G \subset E$ is a best approximation to $f \in E$ from G if and only if $f - g \perp G$.*

<u>Remark:</u> The notation used in Theorem 6.33 is to be interpreted as

$$f-g \perp G \quad \Longleftrightarrow \quad \langle f-g, h\rangle = 0 \text{ for all } h \in G.$$

Proof: We recall the Pythagorean Theorem in an inner product space (see, e.g., Lemma 5.6 in the Math 577 class notes, or page 274 of the textbook):

$$\text{If} \quad f \perp g \quad \text{then} \quad \|f+g\|^2 = \|f\|^2 + \|g\|^2.$$

(a) "$\Longleftarrow$": We assume $f - g \perp G$, i.e., $f - g \perp h$ for all $h \in G$. Let's take an arbitrary $h \in G$. Then, using the Pythagorean Theorem,

$$\|f-h\|^2 = \|\underbrace{f-g}_{\perp G} + \underbrace{g-h}_{\in G}\|^2$$

$$\begin{aligned} &= \|f-g\|^2 + \underbrace{\|g-h\|^2}_{\geq 0} \\ &\geq \|f-g\|^2, \end{aligned}$$

and so g is a best approximation.
(b) "$\Longrightarrow$": Now we assume $g \in G$ is a best approximation to f. Then

$$\|f-g\|^2 \leq \|f - (\underbrace{g+\lambda h}_{\in G})\|^2, \qquad \lambda > 0, h \in G$$

or

$$\|f-g\|^2 \leq \|f-g\|^2 - 2\lambda\langle f-g, h\rangle + \lambda^2\|h\|^2.$$

This statement is equivalent to

$$\begin{aligned} 0 &\leq \lambda^2\|h\|^2 - 2\lambda\langle f-g,h\rangle \\ &= \lambda\left(\lambda\|h\|^2 - 2\langle f-g,h\rangle\right). \end{aligned}$$

Thus, if $\lambda \to 0^+$, then we need to have $\langle f-g,h\rangle \leq 0$.

Similarly – using $-h$ instead of h – we can argue that $\langle f-g,h\rangle \geq 0$. Together we have $\langle f-g,h\rangle = 0$, and since h was arbitrary we have $f-g \perp G$. ♠

Remarks:

1. In an inner product space the best approximation is unique. This follows from the proof of Theorem 6.33.
2. If we take $E = \mathbb{R}^3$ and $G = \mathbb{R}^2$, then the best approximation g is given by the *orthogonal projection* of f onto G.

If we can obtain an *orthonormal basis* for G then

Theorem 6.34 *Let $\{g_1, \ldots, g_n\}$ be an orthonormal basis of $G \subset E$. The best approximation g of f from G is a generalized Fourier series of the form*

$$g = \sum_{i=1}^n c_i g_i$$

with generalized Fourier coefficients

$$c_i = \langle f, g_i\rangle.$$

Remark: The best approximation can be interpreted as the sum of the projections of f onto the basis elements g_i of G.

Proof: By the previous theorem the best approximation g is characterized by

$$f - \underbrace{\sum_{i=1}^n c_i g_i}_{=g} \perp G. \tag{29}$$

We need to show that (29) is equivalent to the expression $c_i = \langle f, g_i \rangle$ for the Fourier coefficients.

(29) implies that

$$f - \sum_{i=1}^{n} c_i g_i \perp g_j, \qquad j = 1, \ldots, n.$$

Therefore, taking the inner product with g_j we have

$$\langle f, g_j \rangle - \langle \sum_{i=1}^{n} c_i g_i, g_j \rangle = 0.$$

However,

$$\langle \sum_{i=1}^{n} c_i g_i, g_j \rangle = \sum_{i=1}^{n} c_i \underbrace{\langle g_i, g_j \rangle}_{\delta_{ij}} = c_j,$$

and therefore

$$\langle f, g_j \rangle - c_j = 0.$$

♠

Theorem 6.35 (Parseval's Identity) *Let $\{g_1, \ldots, g_n\}$ be an orthonormal basis of G, and let g be the best approximation of f from G. Then*

$$\|g\|^2 = \sum_{i=1}^{n} \langle f, g_i \rangle^2.$$

Proof: By induction. First, note that the best approximation is given by the Fourier series

$$g = \sum_{i=1}^{n} \langle f, g_i \rangle g_i.$$

For $n = 1$ this implies $g = \langle f, g_1 \rangle g_1$. Therefore, since g_1 is normalized,

$$\|g\|^2 = \|\langle f, g_1 \rangle g_1\|^2 = \langle f, g_1 \rangle^2 \underbrace{\|g_1\|^2}_{=1}.$$

Now, assume the statement holds for arbitrary n, show it is also true for $n+1$.

$$\begin{aligned} \|g\|^2 &= \|\sum_{i=1}^{n+1} \langle f, g_i \rangle g_i\|^2 \\ &= \|\sum_{i=1}^{n} \langle f, g_i \rangle g_i + \langle f, g_{n+1} \rangle g_{n+1}\|^2. \end{aligned}$$

Since g_{n+1} is orthogonal to all of $g_1, \ldots, g_n$, we can apply the Pythagorean Theorem and get

$$\|g\|^2 = \|\sum_{i=1}^{n} \langle f, g_i \rangle g_i\|^2 + \|\langle f, g_{n+1} \rangle g_{n+1}\|^2$$

$$\begin{aligned}
&= \sum_{i=1}^{n} \langle f, g_i \rangle^2 + \langle f, g_{n+1} \rangle^2 \underbrace{\|g_{n+1}\|^2}_{=1} \\
&= \sum_{i=1}^{n+1} \langle f, g_i \rangle^2,
\end{aligned}$$

where we have used the induction hypothesis and the normalization of g_{n+1}. ♠

Theorem 6.36 (Bessel's Inequality) *Let $\{g_1, \ldots, g_n\}$ be an orthonormal basis of G. Then*

$$\sum_{i=1}^{n} \langle f, g_i \rangle^2 \le \|f\|^2.$$

Proof: Consider the best approximation g to f. Now, by the Pythagorean Theorem,

$$\begin{aligned}
\|f\|^2 &= \| \underbrace{f - g}_{\perp G} + \underbrace{g}_{\in G} \|^2 \\
&= \underbrace{\|f - g\|^2}_{\ge 0} + \|g\|^2 \\
&\ge \|g\|^2.
\end{aligned}$$

The proof is completed by applying Parseval's identity. ♠

<u>**Remark:**</u> Given a linearly independent set $\{h_1, \ldots, h_n\}$ we can always obtain an orthonormal set by applying the Gram-Schmidt method.

<u>**Best Approximation with Orthogonal Polynomials**</u>

We now take the approximation space G to be the space of polynomials of degree n, and show how one can find orthogonal bases for this approximation space (with respect to a specific inner product).

Theorem 6.37 *Let $\langle\ ,\ \rangle$ be an inner product. Any sequence of polynomials defined recursively via*

$$p_0(x) = 1, \qquad p_1(x) = x - a_1,$$
$$p_n(x) = (x - a_n) p_{n-1}(x) - b_n p_{n-2}(x), \quad n \ge 2$$

with

$$a_n = \frac{\langle x p_{n-1}, p_{n-1} \rangle}{\langle p_{n-1}, p_{n-1} \rangle}, \qquad b_n = \frac{\langle x p_{n-1}, p_{n-2} \rangle}{\langle p_{n-2}, p_{n-2} \rangle}$$

is orthogonal with respect to the inner product $\langle\ ,\ \rangle$.

Proof: First note that all formulas are well-defined since all polynomials p_n, $n = 0, 1, 2, \ldots$, are monic (and thus all denominators are nonzero). We use induction to show that

$$\langle p_n, p_i \rangle = 0, \qquad i = 0, 1, \ldots, n-1. \tag{30}$$

For the case $n = 1$ we use the definition of p_0, p_1 and a_1 to get

$$\begin{aligned}
\langle p_1, p_0 \rangle &= \langle x - a_1, 1 \rangle \\
&= \langle x - \frac{\langle x, 1 \rangle}{\langle 1, 1 \rangle}, 1 \rangle \\
&= \langle x, 1 \rangle - \frac{\langle \langle x, 1 \rangle, 1 \rangle}{\langle 1, 1 \rangle} \\
&= \langle x, 1 \rangle - \langle x, 1 \rangle \underbrace{\frac{\langle 1, 1 \rangle}{\langle 1, 1 \rangle}}_{=1} \\
&= 0.
\end{aligned}$$

Now we assume that (30) holds for n, and show that it is also true for $n+1$. Using the recurrence relation we have

$$\begin{aligned}
\langle p_{n+1}, p_i \rangle &= \langle (x - a_{n+1}) p_n - b_{n+1} p_{n-1}, p_i \rangle \\
&= \langle x p_n, p_i \rangle - a_{n+1} \underbrace{\langle p_n, p_i \rangle}_{=0} - b_{n+1} \underbrace{\langle p_{n-1}, p_i \rangle}_{=0} \\
&= \langle x p_n, p_i \rangle.
\end{aligned}$$

For any real inner product the last expression is equal to $\langle p_n, x p_i \rangle$. Next, the recurrence relation (for $n = i + 1$) implies

$$\begin{aligned}
& p_{i+1}(x) = (x - a_{i+1}) p_i(x) - b_{i+1} p_{i-1}(x) \\
\iff \quad & x p_i(x) = p_{i+1}(x) + a_{i+1} p_i(x) + b_{i+1} p_{i-1}(x)
\end{aligned}$$

Therefore, we get

$$\begin{aligned}
\langle p_{n+1}, p_i \rangle &= \langle p_n, p_{i+1} + a_{i+1} p_i + b_{i+1} p_{i-1} \rangle \\
&= \underbrace{\langle p_n, p_{i+1} \rangle}_{=0,\ i=0,1,\ldots,n-2} + a_{i+1} \underbrace{\langle p_n, p_i \rangle}_{=0,\ i=0,1,\ldots,n-1} + b_{i+1} \underbrace{\langle p_n, p_{i-1} \rangle}_{=0,\ i=0,1,\ldots,n} \\
&= 0
\end{aligned}$$

for $i = 0, 1, \ldots, n-2$, and we still need to show

$$\langle p_{n+1}, p_i \rangle = 0 \qquad \text{for } i = n-1, n.$$

For these two cases we apply the recurrence relation directly:

$$\begin{aligned}
\langle p_{n+1}, p_{n-1} \rangle &= \langle x p_n, p_{n-1} \rangle - a_{n+1} \underbrace{\langle p_n, p_{n-1} \rangle}_{=0} - b_{n+1} \langle p_{n-1}, p_{n-1} \rangle \\
&= \langle x p_n, p_{n-1} \rangle - \underbrace{\frac{\langle x p_n, p_{n-1} \rangle}{\langle p_{n-1}, p_{n-1} \rangle}}_{=b_{n+1}} \langle p_{n-1}, p_{n-1} \rangle \\
&= 0,
\end{aligned}$$

and

$$\begin{aligned}\langle p_{n+1}, p_n\rangle &= \langle xp_n, p_n\rangle - a_{n+1}\langle p_n, p_n\rangle - b_{n+1}\underbrace{\langle p_{n-1}, p_n\rangle}_{=0} \\ &= \langle xp_n, p_n\rangle - \underbrace{\frac{\langle xp_n, p_n\rangle}{\langle p_n, p_n\rangle}}_{=a_{n+1}}\langle p_n, p_n\rangle \\ &= 0.\end{aligned}$$

♠

Different families of orthogonal polynomials are now obtained by choosing an inner product, and then applying the recurrence relation. We now discuss the "classical" orthogonal polynomials.

Example 1: Legendre polynomials. We take the inner product

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)dx$$

We always start with $p_0(x) = 1$. According to Theorem 6.37 we then have

$$p_1(x) = x - a_1$$

with

$$a_1 = \frac{\langle xp_0, p_0\rangle}{\langle p_0, p_0\rangle} = \frac{\int_{-1}^{1} xdx}{\int_{-1}^{1} dx} = 0,$$

so that $p_1(x) = x$. Next,

$$p_2(x) = (x - a_2)p_1(x) - b_2p_0(x),$$

with

$$a_2 = \frac{\langle xp_1, p_1\rangle}{\langle p_1, p_1\rangle} = \frac{\int_{-1}^{1} x^3dx}{\int_{-1}^{1} x^2dx} = 0$$

and

$$b_2 = \frac{\langle xp_1, p_0\rangle}{\langle p_0, p_0\rangle} = \frac{\int_{-1}^{1} x^2dx}{\int_{-1}^{1} dx} = \frac{1}{3},$$

so that $p_2(x) = x^2 - \frac{1}{3}$. Similarly, one can obtain

$$\begin{aligned}p_3(x) &= x^3 - \frac{3}{5}x, \\ p_4(x) &= x^4 - \frac{6}{7}x^2 + \frac{3}{35}, \\ p_5(x) &= x^5 - \frac{10}{9}x^3 + \frac{5}{21}x.\end{aligned}$$

Example 2: The Chebyshev polynomials of the first kind are obtained via the inner product

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)\frac{1}{\sqrt{1-x^2}}dx.$$

<u>**Remark:**</u> Recall that we encountered these polynomials when discussing optimal placement of data sites for polynomial interpolation.

Example 3: The Jacobi polynomials are obtained via the inner product

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)(1-x)^{\alpha}(1+x)^{\beta}dx, \quad \alpha, \beta > -1.$$

Example 4: The Hermite polynomials are obtained via the inner product

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f(x)g(x)e^{-x^2}dx.$$

Example 5: The Chebyshev polynomials of the second kind are obtained via the inner product

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)\sqrt{1-x^2}dx.$$

Example 6: The Laguerre polynomials are obtained via the inner product

$$\langle f, g\rangle = \int_{0}^{\infty} f(x)g(x)x^{\alpha}e^{-x}dx, \quad \alpha > -1.$$

<u>**Remarks:**</u>

1. The Maple worksheet `578_OrthogonalPolynomials.mws` lists the first few (monic) polynomials of each type listed above.

2. Of course, the definition of Jacobi polynomials includes that of Legendre polynomials (for $\alpha = \beta = 0$, Chebyshev polynomials of the first kind ($\alpha = \beta = -\frac{1}{2}$), and Chebyshev polynomials of the second kind ($\alpha = \beta = \frac{1}{2}$).

<u>Normal Equations and Gram Matrix</u>

From above we know that the best approximation g to f from G is characterized by $f - g \perp G$. In particular, if G has a basis $\{g_1, \ldots, g_n\}$ (orthogonal or not), then the best approximation is characterized by

$$\langle f - g, g_i\rangle = 0, \qquad i = 1, \ldots, n.$$

Now $g \in G$, so

$$g = \sum_{j=1}^{n} c_j g_j,$$

and we have

$$\langle f - \sum_{j=1}^{n} c_j g_j, g_i\rangle = 0, \qquad i = 1, \ldots, n.$$

By linearity we have

$$\sum_{j=1}^{n} c_j \langle g_j, g_i \rangle = \langle f, g_i \rangle, \qquad i = 1, \ldots, n.$$

This is a system of linear equations of the form

$$\underbrace{\begin{bmatrix} \langle g_1, g_1 \rangle & \ldots & \langle g_n, g_1 \rangle \\ \vdots & & \vdots \\ \langle g_1, g_n \rangle & \ldots & \langle g_n, g_n \rangle \end{bmatrix}}_{=\mathcal{G}} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \langle f, g_1 \rangle \\ \vdots \\ \langle f, g_n \rangle \end{bmatrix}, \tag{31}$$

where $\mathcal{G}$ is a so-called *Gram matrix.* The system (31) is referred to as the *normal equations* of the least squares problem (cf. the earlier discussion in Sect.5.1).

Remarks:

1. If the set $\{g_1, \ldots, g_n\}$ is linearly independent, then the matrix $\mathcal{G}$ is nonsingular. This follows from the fact that g is the unique best least squares approximation, and the fact that g has a unique representation with respect to the basis $\{g_1, \ldots, g_n\}$.
2. If the set $\{g_1, \ldots, g_n\}$ is orthogonal, then $\mathcal{G}$ is diagonal. For an orthonormal basis $\mathcal{G}$ is even the identity matrix.

According to these remarks any basis of G can be used to compute g. However, an orthogonal basis leads to a trivial linear system. Other bases may lead to ill-conditioned Gram matrices.

Example: Find the best least squares approximation from the space of quadratic polynomials to $f(x) = e^x$ on $[-1, 1]$.

First we use the monomial basis $\{1, x, x^2\}$, so that

$$g(x) = \sum_{i=0}^{2} c_i x^i.$$

The normal equations become

$$\begin{bmatrix} \langle 1, 1 \rangle & \langle x, 1 \rangle & \langle x^2, 1 \rangle \\ \langle 1, x \rangle & \langle x, x \rangle & \langle x^2, x \rangle \\ \langle 1, x^2 \rangle & \langle x, x^2 \rangle & \langle x^2, x^2 \rangle \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \langle e^x, 1 \rangle \\ \langle e^x, x \rangle \\ \langle e^x, x^2 \rangle \end{bmatrix}.$$

Using the inner product

$$\langle f, g \rangle = \int_{-1}^{1} f(x) g(x) dx$$

the system can be evaluated to

$$\begin{bmatrix} 2 & 0 & \frac{2}{3} \\ 0 & \frac{2}{3} & 0 \\ \frac{2}{3} & 0 & \frac{2}{5} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} e - \frac{1}{e} \\ \frac{2}{e} \\ e - \frac{5}{e} \end{bmatrix}.$$

The Gram matrix in this example has an ℓ_∞ condition number of 20. In fact, the structure of Gram matrices arising for best least squares approximation with monomial bases is similar to that of a Hilbert matrix, which is a well-known example of an ill-conditioned matrix.

After solving the normal equations we get

$$g(x) = -\frac{3}{4}e + \frac{33}{4e} + \frac{3}{e}x + \left(\frac{15e}{4} - \frac{105}{4e}\right)x^2. \tag{32}$$

The graphs of f and g are shown in Figure 18.

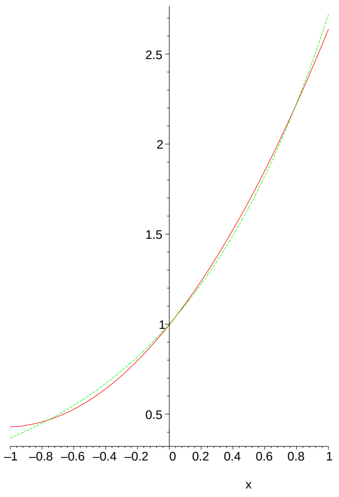

Figure 18: Graphs of $f(x) = e^x$ (dashed green) and its best least squares approximation g (solid red).

Now we use the Legendre polynomials $\{p_0, p_1, p_2\} = \{1, x, x^2 - 1/3\}$ which form an orthogonal basis for the space of quadratic polynomials on $[-1, 1]$ (with respect to the inner product chosen above). Therefore, the best approximation will be represented as

$$g(x) = \sum_{i=0}^{2} c_i p_i(x).$$

The normal equations become

$$\begin{bmatrix} \langle 1,1\rangle & \langle x,1\rangle & \langle x^2-1/3,1\rangle \\ \langle 1,x\rangle & \langle x,x\rangle & \langle x^2-1/3,x\rangle \\ \langle 1,x^2-1/3\rangle & \langle x,x^2-1/3\rangle & \langle x^2-1/3,x^2-1/3\rangle \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \langle e^x,1\rangle \\ \langle e^x,x\rangle \\ \langle e^x,x^2-1/3\rangle \end{bmatrix}$$

or

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & \frac{2}{3} & 0 \\ 0 & 0 & \frac{8}{45} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} e - \frac{1}{e} \\ \frac{2}{e} \\ \frac{2}{3}e - \frac{14}{3e} \end{bmatrix}.$$

$\mathcal{G}$ has an ℓ_∞ condition number of 11.25 (which is almost 1/2 that of the previous approach), and the system is trivial to solve. The representation of g is now

$$g(x) = \frac{1}{2}e - \frac{1}{2e} + \frac{3}{e}x + \left(\frac{15}{4}e - \frac{105}{4e}\right)\left(x^2 - \frac{1}{3}\right),$$

which of course is equivalent to (32).

An alternative to continuous least squares approximation is

Discrete Least Squares Approximation

This method is useful when the data is given as a set of discrete points (x_i, y_i), $i = 1, \ldots, N$, instead of as a function f. Now we want the best ℓ_2 approximation from G.

Assuming that G has a basis $\{g_1, \ldots, g_n\}$ (usually with $n \ll N$) we represent the best approximation in the form

$$g = \sum_{j=1}^{n} c_j g_j,$$

and determine the unknown coefficients c_j, $j = 1, \ldots, n$, by minimizing the discrete ℓ_2 norm of the difference between the data and the best approximation, i.e.

$$\sum_{i=1}^{N} (y_i - g(x_i))^2 = \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{n} c_j g_j(x_i) \right)^2 \quad \to \quad \text{min.}$$

Remarks:

1. A solution of this problem can be found (cf. Sect. 5.4.3 or Sect. 5.5.5) by computing the QR or singular value decomposition of the Gram matrix $\mathcal{G}$ with entries

   $$\mathcal{G}_{ij} = g_j(x_i), \qquad i = 1, \ldots, N,\ j = 1, \ldots, n.$$

   An example involving polynomial discrete least squares approximation was presented in Sect. 5.1.3 and the Matlab demo `PolynomialDemo.m`.

2. If $n = N$ then we are back to the case of interpolation to the values y_i at x_i, $i = 1, \ldots, N$.

3. If a set of linear constraints is added to the quadratic minimization problem then the Lagrange multiplier technique is used to convert the entire problem to a linear system.

4. We will return to discrete least squares approximation (with linear constraints) when we discuss the moving least squares method for approximation of multivariate data.

6.9 Trigonometric Interpolation

This section can be found as Section 6.12 in the textbook.

If we need to interpolate or approximate periodic functions or periodic data, then it is more natural to use an approximation space consisting of periodic functions. In the case of continuous least squares approximation on $[-\pi, \pi]$ this leads to classical Fourier series. The following theorem is proved in any standard undergraduate class on Fourier series.

Theorem 6.38 *If $f \in C^1[-\pi,\pi]$ and f is 2π-periodic then its best approximation from*

$$G = \operatorname{span}\{1, \cos x, \cos 2x, \ldots, \sin x, \sin 2x, \ldots\}$$

is given by

$$f(x) \sim g(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos kx + b_k \sin kx)$$

with

$$\begin{aligned} a_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos kt dt, \\ b_k &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin kt dt. \end{aligned}$$

Moreover, g converges uniformly to f.

<u>Complex Fourier Series</u>

By employing Euler's formula

$$e^{i\phi} = \cos\phi + i\sin\phi$$

the Fourier series approach can be unified. Then

$$f(x) \sim \sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

with

$$c_k = \frac{1}{2}(a_k - ib_k) \qquad (a_{-k} = a_k,\ b_{-k} = b_k,\ b_0 = 0).$$

This implies

$$\begin{aligned} c_k &= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\left(\cos kt - i\sin kt\right) dt \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)e^{ikt}dt = \hat{f}(k) \end{aligned}$$

and the coefficients $c_k = \hat{f}(k)$ are referred to as the *Fourier transform* of f.

<u>Remark:</u> Just as $\{1, \cos x, \cos 2x, \ldots, \sin x, \sin 2x, \ldots\}$ are orthonormal with respect to the inner product

$$\langle f, g\rangle = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)g(x)dx,$$

the functions $\{e^{ikx}\}_{k=-\infty}^{\infty}$ are orthonormal with respect to the (complex) inner product

$$\langle f, g\rangle = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)\overline{g(x)}dx.$$

Let's verify the statement just made. Let E_k be defined by

$$E_k(x) := e^{ikx}.$$

Then

$$\langle E_k, E_k\rangle = \frac{1}{2\pi}\int_{-\pi}^{\pi} \underbrace{e^{ikx}e^{-ikx}}_{=1} dx = 1.$$

For $k \neq \ell$ we have

$$\begin{aligned}
\langle E_k, E_\ell\rangle &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{ikx}e^{-i\ell x}dx \\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{i(k-\ell)x}dx \\
&= \frac{1}{2\pi}\left.\frac{e^{i(k-\ell)x}}{i(k-\ell)}\right|_{-\pi}^{\pi} \\
&= \frac{1}{2\pi i(k-\ell)}\left[e^{i(k-\ell)\pi} - e^{-i(k-\ell)\pi}\right] \\
&= \frac{1}{2\pi i(k-\ell)}e^{i(k-\ell)\pi}\left[1 - \underbrace{e^{2\pi(\ell-k)i}}_{=1}\right] = 0.
\end{aligned}$$

In order to discuss discrete least squares approximation or interpolation with trigonometric functions we introduce

Exponential Polynomials

Let

$$P(x) = \sum_{k=0}^{n} c_k E_k(x).$$

Since we can write

$$P(x) = \sum_{k=0}^{n} c_k e^{ikx} = \sum_{k=0}^{n} c_k \left(e^{ix}\right)^k$$

P is a polynomial of degree k in the complex exponential e^{ix}.

For discrete least squares approximation we now consider equally spaced data on the interval $[0, 2\pi)$, i.e.,

$$(x_j, f(x_j)) \quad \text{with} \quad x_j = \frac{2\pi j}{N},\ j = 0, 1, \ldots, N-1, \tag{33}$$

and introduce a discrete (pseudo-)inner product

$$\langle f, g\rangle_N = \frac{1}{N}\sum_{j=0}^{N-1} f\left(\frac{2\pi j}{N}\right)\overline{g\left(\frac{2\pi j}{N}\right)}. \tag{34}$$

Remark: This is only a pseudo-inner product since $\langle f, f\rangle_N = 0$ implies $f = 0$ only at the discrete nodes $\frac{2\pi j}{N}$, $j = 0, 1, \ldots, N-1$.

Theorem 6.39 *The set $\{E_k\}_{k=0}^{N-1}$ of complex exponentials is orthonormal with respect to the inner product $\langle\ ,\ \rangle_N$ defined in (34).*

Proof: First,

$$\begin{aligned}\langle E_k, E_k\rangle_N &= \frac{1}{N}\sum_{j=0}^{N-1} E_k\left(\frac{2\pi j}{N}\right)\overline{E_k\left(\frac{2\pi j}{N}\right)}\\ &= \frac{1}{N}\sum_{j=0}^{N-1} \underbrace{e^{ik2\pi j/N}e^{-ik2\pi j/N}}_{=1} = 1.\end{aligned}$$

Next, for $k \neq \ell$ we have

$$\begin{aligned}\langle E_k, E_\ell\rangle_N &= \frac{1}{N}\sum_{j=0}^{N-1} E_k\left(\frac{2\pi j}{N}\right)\overline{E_\ell\left(\frac{2\pi j}{N}\right)}\\ &= \frac{1}{N}\sum_{j=0}^{N-1} e^{ik2\pi j/N}e^{-i\ell 2\pi j/N}\\ &= \frac{1}{N}\sum_{j=0}^{N-1}\left(e^{i\frac{2\pi(k-\ell)}{N}}\right)^j.\end{aligned}$$

Now, since $k \neq \ell$,

$$e^{i\frac{2\pi(k-\ell)}{N}} \neq 1$$

we can apply the formula for the sum of a finite geometric series to get

$$\langle E_k, E_\ell\rangle_N = \frac{1}{N}\frac{\left(e^{i\frac{2\pi(k-\ell)}{N}}\right)^N - 1}{e^{i\frac{2\pi(k-\ell)}{N}} - 1}.$$

Finally, since

$$\left(e^{i\frac{2\pi(k-\ell)}{N}}\right)^N = 1$$

the orthogonality result follows. ♠

Corollary 6.40 *Let data be given as in (33) and consider $G = \operatorname{span}\{E_k\}_{k=0}^n$ with $n < N$. Then the best least squares approximation to the data from G (with respect to the inner product (34)) is given by the discrete Fourier series*

$$g(x) = \sum_{k=0}^{n} c_k E_k(x)$$

with

$$c_k = \langle f, E_k\rangle_N = \frac{1}{N}\sum_{j=0}^{N-1} f\left(\frac{2\pi j}{N}\right)e^{-ik\frac{2\pi j}{N}}, \qquad k = 0, 1, \ldots, n.$$

Proof: This follows immediately from the orthonormality of $\{E_k\}_{k=0}^n$ and from Theorem 6.34 on generalized Fourier series. ♠

Remarks:

1. The coefficients c_k of the discrete Fourier series are called the *discrete Fourier transform* (DFT) of f.

2. In the next section we will compute the DFT (and also evaluate g) via the fast Fourier transform (FFT).

3. Note that, even if the data is real, the DFT will in general be complex. If, e.g., $N = 4$, and the values $f\left(\frac{2\pi j}{N}\right)$, $j = 0, 1, 2, 3$ are $4, 0, 3, 6$ respectively, then the coefficients c_k are

$$\begin{aligned} c_0 &= 13 \\ c_1 &= 1 + 6i \\ c_2 &= 1 \\ c_3 &= 1 - 6i. \end{aligned}$$

Corollary 6.41 *Let data be given as in (33), then the exponential polynomial*

$$P(x) = \sum_{k=0}^{N-1} \langle f, E_k \rangle_N E_k(x)$$

is the unique interpolant to the data from $G = \text{span}\{E_k\}_{k=0}^{N-1}$.

Proof: Apply Corollary 6.40 with $n = N - 1$. ♠

Remark: The DFT can be interpreted as a matrix-vector product with matrix E whose entries are given by

$$E_{jk} = e^{-ik\frac{2\pi j}{N}},$$

which can be computed in $\mathcal{O}(N^2)$ operations. On the other hand, the DFT yields an interpolant to the given data, and interpolation problems usually require solution of a linear system (at $\mathcal{O}(N^3)$ cost). Thus, the DFT (even if implemented naively) yields an improvement of one order of magnitude.

Example: Consider the data

x	0	π
y	$f(0)$	$f(\pi)$

.

We therefore have $N = 2$, and according to Corollary 6.41 the unique interpolant from $\text{span}\{E_0, E_1\}$ is given by

$$\begin{aligned} P(x) &= \sum_{k=0}^{1} c_k E_k(x) \\ &= \langle f, E_0 \rangle_2 E_0(x) + \langle f, E_1 \rangle_2 E_1(x) \\ &= \frac{1}{2}\left[\sum_{j=0}^{1} f\left(\frac{2\pi j}{2}\right) \underbrace{e^{-i0\frac{2\pi j}{2}}}_{=1}\right] \underbrace{e^{i0x}}_{=1} + \frac{1}{2}\left[\sum_{j=0}^{1} f\left(\frac{2\pi j}{2}\right) e^{-i\frac{2\pi j}{2}}\right] e^{ix} \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2}\left[f(0)+f(\pi)\right] + \frac{1}{2}\left[f(0)\underbrace{e^{-i0}}_{=1} + f(\pi)\underbrace{e^{-i\pi}}_{=-1}\right] e^{ix} \\
&= \frac{1}{2}\left[f(0)+f(\pi)\right] + \frac{1}{2}\left[f(0)-f(\pi)\right] e^{ix}.
\end{aligned}
$$

6.10 Fast Fourier Transform (FFT)

This can be found as Section 6.13 in the textbook.

We begin with the trigonometric polynomial

$$P(x) = \sum_{k=0}^{N-1} c_k E_k(x)$$

with DFT coefficients

$$
\begin{aligned}
c_k &= \frac{1}{N}\sum_{j=0}^{N-1} f\left(\frac{2\pi j}{N}\right) e^{-ik\frac{2\pi j}{N}} \\
&= \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)(\lambda_k)^j
\end{aligned}
$$

with

$$x_j = \frac{2\pi j}{N} \qquad \text{and} \qquad \lambda_k = e^{-i\frac{2\pi k}{N}}.$$

Remark: The values λ_k (or more naturally λ_{-k}) are often referred to as N-th roots of unity. This is illustrated in Figure 19 for $N = 5$.

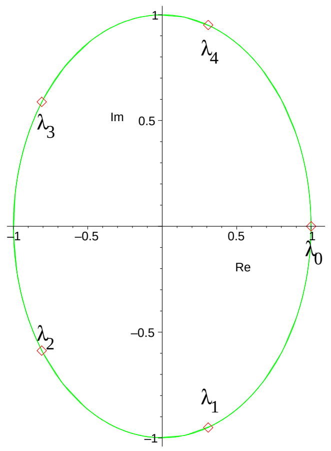

Figure 19: Roots of unity λ_k, $k = 0, 1, \ldots, N-1$, for $N = 5$.

The computation of each coefficient c_k corresponds to evaluation of a polynomial of degree $N-1$ in λ_k. Using Horner's method this can be done in $\mathcal{O}(N)$ operations.

Since we have N coefficients, the polynomial P can be constructed in $\mathcal{O}(N^2)$ operations. (Earlier we saw a matrix-vector multiplication argument to the same effect.)

The major benefit of the fast Fourier transform is that it reduces the amount of work to $\mathcal{O}(N \log_2 N)$ operations.

The FFT was discovered by Cooley and Tukey in 1965. However, Gauss seemed to already be aware of similar ideas. One of the most popular modern references is the "DFT Owners Manual" by Briggs and Henson (published by SIAM in 1995). It is the dramatic reduction in computational complexity that earned the FFT a spot on the list of "Top 10 Algorithms of the 20th Century" (see handout).

The FFT is based on the following observation:

Theorem 6.42 *Let data $(x_k, f(x_k))$, $x_k = \frac{k\pi}{n}$, $k = 0, 1, \dots, 2n-1$, be given, and assume that p and q are exponential polynomials of degree at most $n-1$ which interpolate part of the data according to*

$$p(x_{2j}) = f(x_{2j}), \qquad q(x_{2j}) = f(x_{2j+1}), \qquad j = 0, 1, \dots, n-1,$$

then the exponential polynomial of degree at most $2n-1$ which interpolates all the given data is

$$P(x) = \frac{1}{2}\left(1 + e^{inx}\right) p(x) + \frac{1}{2}\left(1 - e^{inx}\right) q\left(x - \frac{\pi}{n}\right). \tag{35}$$

Proof: Since $e^{inx} = \left(e^{ix}\right)^n$ has degree n, and p and q have degree at most $n-1$ it is clear that P has degree at most $2n-1$.

We need to verify the interpolation claim, i.e., show that

$$P(x_k) = f(x_k), \qquad k = 0, 1, \dots, 2n-1.$$

By assumption we have

$$P(x_k) = \frac{1}{2}\left(1 + e^{inx_k}\right) p(x_k) + \frac{1}{2}\left(1 - e^{inx_k}\right) q\left(x_k - \frac{\pi}{n}\right)$$

where

$$e^{inx_k} = e^{in\frac{k\pi}{n}} = \left(e^{i\pi}\right)^k = (-1)^k.$$

Therefore

$$P(x_k) = \begin{cases} p(x_k) & \text{if } k \text{ even,} \\ q\left(x_k - \frac{\pi}{n}\right) & \text{if } k \text{ odd.} \end{cases}$$

Let k be even, i.e., $k = 2j$. Then, by the assumption on p

$$P(x_{2j}) = p(x_{2j}) = f(x_{2j}), \qquad j = 0, 1, \dots, n-1.$$

On the other hand, for $k = 2j+1$ (odd), we have (using the assumption on q)

$$P(x_{2j+1}) = q\left(x_{2j+1} - \frac{\pi}{n}\right) = q(x_{2j}) = f(x_{2j+1}), \qquad j = 0, 1, \dots, n-1.$$

This is true since

$$x_{2j+1} - \frac{\pi}{n} = \frac{(2j+1)\pi}{n} - \frac{\pi}{n} = \frac{2j\pi}{n} = x_{2j}.$$

♠

The second useful fact tells us how to obtain the coefficients of P from those of p and q.

Theorem 6.43 *Let*

$$p = \sum_{j=0}^{n-1} \alpha_j E_j, \qquad q = \sum_{j=0}^{n-1} \beta_j E_j, \qquad \text{and} \qquad P = \sum_{j=0}^{2n-1} \gamma_j E_j.$$

Then, for $j = 0, 1, \ldots, n-1$,

$$\begin{aligned} \gamma_j &= \frac{1}{2}\alpha_j + \frac{1}{2}e^{-ij\pi/n}\beta_j \\ \gamma_{j+n} &= \frac{1}{2}\alpha_j - \frac{1}{2}e^{-ij\pi/n}\beta_j. \end{aligned}$$

Proof: In order to use (35) we need to rewrite

$$\begin{aligned} q\left(x - \frac{\pi}{n}\right) &= \sum_{j=0}^{n-1} \beta_j E_j \left(x - \frac{\pi}{n}\right) \\ &= \sum_{j=0}^{n-1} \beta_j e^{ij\left(x - \frac{\pi}{n}\right)} \\ &= \sum_{j=0}^{n-1} \beta_j e^{ijx} e^{-ij\pi/n}. \end{aligned}$$

Therefore

$$\begin{aligned} P(x) &= \frac{1}{2}\left(1 + e^{inx}\right) p(x) + \frac{1}{2}\left(1 - e^{inx}\right) q\left(x - \frac{\pi}{n}\right) \\ &= \frac{1}{2}\sum_{j=0}^{n-1}\left(1 + e^{inx}\right)\alpha_j e^{ijx} + \frac{1}{2}\sum_{j=0}^{n-1}\left(1 - e^{inx}\right)\beta_j e^{ijx} e^{-ij\pi/n} \\ &= \frac{1}{2}\sum_{j=0}^{n-1}\left[\left(\alpha_j + \beta_j e^{-ij\pi/n}\right)e^{ijx} + \left(\alpha_j - \beta_j e^{-ij\pi/n}\right)e^{i(n+j)x}\right]. \end{aligned}$$

However, the terms in parentheses (together with the factor 1/2) lead precisely to the formulæ for the coefficients γ. ♠

Example 1: As in the previous example, we consider the data

x	0	π
y	$f(0)$	$f(\pi)$

.

Now we apply the DFT to find the interpolating polynomial P. We have $n = 1$ and

$$P(x) = \sum_{j=0}^{1} \gamma_j E_j(x) = \gamma_0 e^0 + \gamma_1 e^{ix}.$$

According to Theorem 6.43 the coefficients are given by

$$\begin{aligned} \gamma_0 &= \frac{1}{2}\left(\alpha_0 + \beta_0 e^0\right), \\ \gamma_1 &= \frac{1}{2}\left(\alpha_0 - \beta_0 e^0\right). \end{aligned}$$

Therefore, we still need to determine α_0 and β_0. They are found via interpolation at only one point (cf. Theorem 6.42):

$$p(x) = \sum_{j=0}^{0} \alpha_j E_j(x) = \alpha_0 \quad \text{such that} \quad p(x_0) = f(x_0) \quad \Longrightarrow \quad \alpha_0 = f(x_0)$$
$$q(x) = \sum_{j=0}^{0} \beta_j E_j(x) = \beta_0 \quad \text{such that} \quad q(x_0) = f(x_1) \quad \Longrightarrow \quad \beta_0 = f(x_1).$$

With $x_0 = 0$ and $x_1 = \pi$ we obtain

$$P(x) = \frac{1}{2}\left(f(0) + f(\pi)\right) + \frac{1}{2}\left(f(0) - f(\pi)\right) e^{ix}$$

which is the same as we computed earlier.

Example 2: We now refine the previous example, i.e., we consider the data

x	0	$\frac{\pi}{2}$	π	$\frac{3\pi}{2}$
y	$f(0)$	$f\left(\frac{\pi}{2}\right)$	$f(\pi)$	$f\left(\frac{3\pi}{2}\right)$

.

Now $n = 2$, and P is of the form

$$P(x) = \sum_{j=0}^{3} \gamma_j E_j(x).$$

According to Theorem 6.43 the coefficients are given by

$$\begin{aligned}
\gamma_0 &= \frac{1}{2}\left(\alpha_0 + \beta_0\right), \\
\gamma_1 &= \frac{1}{2}\Big(\alpha_1 + \beta_1 \underbrace{e^{-i\pi/2}}_{=-i}\Big), \\
\gamma_2 &= \frac{1}{2}\left(\alpha_0 - \beta_0\right), \\
\gamma_3 &= \frac{1}{2}\Big(\alpha_1 - \beta_1 \underbrace{e^{-i\pi/2}}_{=-i}\Big).
\end{aligned}$$

This leaves the coefficients α_0, β_0 and α_1, β_1 to be determined. They are found via interpolation at two points (cf. Theorem 6.42):

$$p(x) = \sum_{j=0}^{1} \alpha_j E_j(x) \quad \text{such that} \quad p(x_0) = f(x_0),\ p(x_2) = f(x_2),$$
$$q(x) = \sum_{j=0}^{1} \beta_j E_j(x) \quad \text{such that} \quad q(x_0) = f(x_1),\ q(x_2) = f(x_3).$$

These are both problems of the form dealt with in the previous example, so a recursive strategy is suggested.

A general recursive algorithm for the FFT is given in the textbook on pages 455 and 456.

Computational Complexity

Assume $N(=2n) = 2^m$ so that the recursive strategy for the FFT can be directly applied. (If N is not a power of 2, then the data can be padded appropriately with zeros). In order to evaluate the interpolating exponential polynomial

$$P(x) = \sum_{k=0}^{N-1} c_k E_k(x) = \sum_{j=0}^{2n-1} \gamma_j E_j(x)$$

we can use the FFT to compute the coefficients c_k (or γ_j).

Instead of giving an exact count of the arithmetic operations involved in this task, we estimate the minimum number of multiplications required to compute the coefficients of an exponential polynomial with N terms.

Lemma 6.44 *Let $R(N)$ be the minimum number of multiplications required to compute the coefficients of an exponential polynomial with $N = 2n$ terms. Then*

$$R(N) \leq 2R(n) + 2n.$$

Proof: There are N coefficients γ_j computed recursively via the formulas

$$\begin{aligned} \gamma_j &= \frac{1}{2}\alpha_j + \frac{1}{2}e^{-ij\pi/n}\beta_j \\ \gamma_{j+n} &= \frac{1}{2}\alpha_j - \frac{1}{2}e^{-ij\pi/n}\beta_j \end{aligned}$$

listed in Theorem 6.43. If the factors $\frac{1}{2}e^{-ij\pi/n}$, $n = 0, 1, \ldots, n-1$, are pre-computed, then each of these formulas involves 2 multiplications for a total of $2n$ multiplications. Moreover, the coefficients α_j and β_j are determined recursively using $R(n)$ multiplications each. ♠

Theorem 6.45 *Under the same assumptions as in Lemma 6.44 we have*

$$R(2^m) \leq m2^m.$$

Proof: We use induction on m. For $m = 0$ there are no multiplications since $P(x) = c_0$. Now, assume the inequality holds for m. Then, by Lemma 6.44 (with $n = 2^m$)

$$R\left(2^{m+1}\right) = R\left(2 \cdot 2^m\right) \leq 2R\left(2^m\right) + 2 \cdot 2^m.$$

By the induction hypothesis this is less than or equal to

$$2m2^m + 2 \cdot 2^m = (m+1)2^{m+1}.$$

♠

Setting $N = 2^m \Leftrightarrow m = \log_2 N$ in Theorem 6.45 implies

$$R(N) \leq N \log_2 N,$$

i.e., the fast Fourier transform reduces the amount of computation required to evaluate an exponential polynomial from $\mathcal{O}(N^2)$ (if done directly) to $\mathcal{O}(N \log_2 N)$. For $N = 1000$ this means roughly 10^4 instead of 10^6 operations.

Remarks:

1. Application of the classical fast Fourier transform is limited to the case where the data is equally spaced, periodic, and of length 2^m. The often suggested strategy of padding with zeros (to get a data vector of length 2^m) or the application of the FFT to non-periodic data may produce unwanted noise.

2. Very recently, a version of the FFT for non-equally spaced data has been proposed by Daniel Potts, Gabriele Steidl and Manfred Tasche.

Remark: Given a vector $a = [a_0, a_1, \ldots, a_{N-1}]^T$ of discrete data, Corollary 6.40 tells us that the discrete Fourier transform $\hat{a}$ of a is computed componentwise as

$$\hat{a}_k = \frac{1}{N} \sum_{j=0}^{N-1} a_j e^{-ik\frac{2\pi j}{N}}, \qquad k = 0, 1, \ldots, N-1.$$

Of course, the FFT can be used to compute these coefficients. There is an analogous formula for the inverse DFT, i.e.,

$$a_j = \sum_{k=0}^{N-1} \hat{a}_k e^{ik\frac{2\pi j}{N}}, \qquad j = 0, 1, \ldots, N-1.$$

We close with a few applications of the FFT. Use of the FFT for many applications benefits from the fact that convolution in the function domain corresponds to multiplication in the transform domain, i.e.,

$$\widehat{f * g} = \hat{f}\hat{g}$$

so that we have

$$f * g = \left(\hat{f}\hat{g}\right)^{\check{}}.$$

Some applications are now

- Filters in signal or image processing.

- Compression of audio of video signals. E.g., the MP3 file format makes use of the discrete cosine transform.

- De-noising of data contaminated with measurement error (see, e.g., the Maple worksheet `578_FFT.mws`).

- Multiplication with cyclic matrices. A cyclic matrix is of the form

$$C = \begin{bmatrix} a_0 & a_1 & \ldots & a_{n-1} & a_n \\ a_n & a_0 & a_1 & \ldots & a_{n-1} \\ a_{n-1} & a_n & a_0 & a_1 & \ldots \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ a_1 & \ldots & a_{n-1} & a_n & a_0 \end{bmatrix}$$

and multiplication by C corresponds to a discrete convolution, i.e.,

$$(Cy)_i = \sum_{j=0}^{n} C_{ij} y_j = \sum_{j=0}^{n} a_{j-i} y_j, \qquad i = 0, \ldots, n,$$

where the subscript on a is to be interpreted cyclically, i.e., $a_{n+1+i} \equiv a_i$ or $a_{-i} \equiv a_{n+1-i}$. Thus, a coupled system of linear equations with cyclic system matrix can be decoupled (and thus trivially solved) by applying the discrete Fourier transform.

- High precision integer arithmetic also makes use of the FFT.
- Numerical solution of differential equations.

Remark: Another transform technique that has some similarities to (but also some significant differences from) the FFT is the fast wavelet transform. Using so-called scaling functions and associated wavelets as orthogonal basis functions one can also perform best approximation of functions (or data). An advantage of wavelets over the trigonometric basis used for the FFT is the fact that not only do we have localization in the transform domain, but also in the function domain.

6.11 Multivariate Interpolation and Approximation

We now briefly turn our attention to some methods that can be used to interpolate or approximate multivariate data, i.e., data of the form

$$(\mathbf{x}_i, f(\mathbf{x}_i)), \; i = 1, \ldots, n, \quad \text{where} \quad \mathbf{x}_i \in \Omega \subset \mathbb{R}^d, \text{ and } f(\mathbf{x}_i) \in \mathbb{R}.$$

Some of the methods discussed will be for the case $d = 2$ only, others will work for arbitrary d. The corresponding section in the textbook is 6.10.

Multivariate interpolation and approximation is still an active research area, and only very few books on the subject are available:

1. C. K. Chui, *Multivariate Splines*, SIAM 1988,
2. P. Lancaster & K. Šalkauskas, *Curve and Surface Fitting*, 1986,
3. W. Cheney & W. Light, *A Course in Approximation Theory*, 1999,
4. K. Höllig, *Finite Element Methods with B-Splines*, SIAM 2002,
5. G. R. Liu, *Mesh Free Methods*, 2003,
6. manuscripts by M. J. Lai & L. L. Schumaker on bivariate splines, and H. Wendland on radial basis functions,
7. as well as several books in the CAGD literature, e.g., by Farin, Hoschek & Lasser.

As in the univariate case, we want to fit the given data with a function s from some approximation space $\mathcal{S}$ with basis $\{B_1, \ldots, B_m\}$, i.e., s will be of the form

$$s(\mathbf{x}) = \sum_{j=1}^{m} c_j B_j(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^d .$$

Depending on whether $m = n$ or $m < n$, the coefficients c_j $(\in \mathbb{R})$ are then determined by interpolation or some form of least squares approximation.

The fundamental difference between univariate and multivariate interpolation is the following theorem on non-interpolation due to Mairhuber and Curtis (1956).

Theorem 6.46 *If $\Omega \subset \mathbb{R}^d$, $d \geq 2$, contains an interior point, then there exist no Haar spaces of continuous functions except for one-dimensional ones.*

In order to understand this theorem we need

Definition 6.47 *Let $\{u_1, \ldots, u_n\}$ be a basis of the function space $\mathcal{U}$. Then $\mathcal{U}$ is a* Haar space *if*

$$\det\left(u_j(\mathbf{x}_i)\right) \neq 0$$

for any set of distinct $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in Ω.

Remarks:

1. Note that existence of a Haar space guarantees invertibility of the interpolation matrix $(u_j(\mathbf{x}_i))$, i.e., existence and uniqueness of an interpolant to data specified at $\mathbf{x}_1, \ldots, \mathbf{x}_n$, from the space $\mathcal{U}$.

2. In the univariate setting we know that polynomials of degree $n-1$ form an n-dimensional Haar space for data given at $x_1, \ldots, x_n$.

3. The Mairhuber-Curtis Theorem implies that in the multivariate setting we can no longer expect this to be the case. E.g., it is not possible to perform unique interpolation with (multivariate) polynomials of degree n to data given at arbitrary locations in $\mathbb{R}^2$.

4. Below we will look at a few constellations of data sites for which unique interpolation with bivariate polynomials is possible.

Proof of Theorem 6.46: Let $d \geq 2$ and suppose $\mathcal{U}$ is a Haar space with basis $\{u_1, \ldots, u_n\}$ with $n \geq 2$. Then, by the definition of a Haar space

$$\det\left(u_j(\mathbf{x}_i)\right) \neq 0 \tag{36}$$

for any distinct $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

Now consider a closed path P in Ω connecting only $\mathbf{x}_1$ and $\mathbf{x}_2$. This is possible since – by assumption – Ω contains an interior point. We can exchange the positions of $\mathbf{x}_1$ and $\mathbf{x}_2$ by moving them continuously along the path P (without interfering with any of the other $\mathbf{x}_j$). This means, however, that rows 1 and 2 of the determinant (36) have been exchanged, and so the determinant has changed sign.

Since the determinant is a continuous function of $\mathbf{x}_1$ and $\mathbf{x}_2$ we must have had $\det = 0$ at some point along P. This is a contradiction. ♠

6.11.1 Gridded Data

Here the domain is usually some rectangular region (or can be decomposed into several such regions). We assume $\Omega = [a,b] \times [c,d]$. The data sites $\mathbf{x}_i$, $i = 1, \ldots, n$, then have components of the form

$$\begin{aligned} x_j &= a + \frac{j-1}{p-1}(b-a), \qquad j = 1, \ldots, p, \\ y_k &= c + \frac{k-1}{q-1}(d-c), \qquad k = 1, \ldots, q, \end{aligned}$$

and $n = pq$. Figure 20 shows the data sites for the case $p = 4$, $q = 3$.

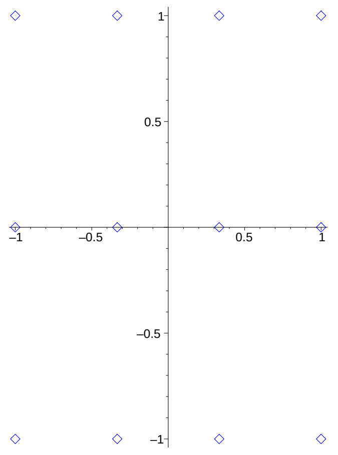

Figure 20: Gridded data with $p = 4$, $q = 3$ on $[-1,1] \times [-1,1]$.

In the case of gridded data it is straightforward to extend univariate methods, and a unique interpolant can be constructed.

Remark: Note that this does not contradict the statement of Theorem 6.46 since we are dealing with a special configuration of data sites.

Tensor-Product Lagrange Interpolation

To obtain an interpolant to the bivariate data $((x_i, y_i), f(x_i, y_i))$, $i = 1, \ldots, n$, we first treat the x-coordinate as a fixed parameter, i.e., we construct a univariate interpolant to data along vertical lines. In this way we obtain

$$s_y(x,y) = \sum_{k=1}^{q} f(x, y_k)\ell_{y,k}(y) \tag{37}$$

with Lagrange functions

$$\ell_{y,k}(y) = \prod_{\substack{\nu=1 \\ \nu \neq k}}^{q} \frac{y - y_\nu}{y_k - y_\nu}, \qquad k = 1, \ldots, q.$$

Note that due to the cardinality property of the Lagrange functions, s_y interpolates data of the type

$$f(x, y_k), \qquad k = 1, \ldots, q.$$

Now we vary x and sample the (partial) interpolant s_y to get the remaining data for a (nested) Lagrange interpolant to all of our data of the form

$$s(x, y) = \sum_{j=1}^{p} s_y(x_j, y)\ell_{x,j}(x) \tag{38}$$

with Lagrange functions

$$\ell_{x,j}(x) = \prod_{\substack{\nu=1 \\ \nu \neq j}}^{p} \frac{x - x_\nu}{x_j - x_\nu}, \qquad j = 1, \ldots, p.$$

Inserting (37) into (38) we obtain

$$s(x, y) = \sum_{j=1}^{p}\sum_{k=1}^{q} f(x_j, y_k)\ell_{x,j}(x)\ell_{y,k}(y),$$

which is one representation of the tensor-product polynomial interpolant to the given data.

Remarks:

1. Of course, we could have first constructed an interpolant s_x to horizontal line data, and then interpolated s_x at the vertical locations – with the same end result.

2. This method is illustrated in the Maple worksheet `578_TP_Lagrange.mws`.

3. Any univariate interpolation (or approximation) method of the form

$$\overline{s}(x) = \sum_{j=1}^{p} c_j B_j(x)$$

   can be used to construct a tensor-product fit

$$s(x, y) = \sum_{j=1}^{p}\sum_{k=1}^{q} c_{jk} B_j(x) B_k(y).$$

   For example, if the data sets are large, then tensor-products of splines are preferred over tensor-products of polynomials.

Boolean Sum Lagrange Interpolation

In the Boolean sum approach we need data given along curves x = const. and y = const. (or we have to generate that data from the given point data). We then create two independent univariate interpolants of the form

$$s_x(x, y) = \sum_{j=1}^{p} f(x_j, y)\ell_{x,j}(x),$$

$$s_y(x,y) = \sum_{k=1}^{q} f(x, y_k)\ell_{y,k}(y),$$

that blend together the given curves. If we want to construct our overall interpolant as the sum of these two partial interpolants, then we need an additional correction term at the data sites of the form

$$s_{xy}(x,y) = \sum_{j=1}^{p}\sum_{k=1}^{q} f(x_j, y_k)\ell_{x,j}(x)\ell_{y,k}(y),$$

so that the final Boolean sum interpolant to our data becomes

$$s(x,y) = s_x(x,y) + s_y(x,y) - s_{xy}(x,y).$$

For more details see homework problem 6.10.6.

Remarks:

1. The Boolean sum idea can be applied to other univariate methods such as splines.
2. In the CAGD literature there are a number of related methods treated under the heading of *blending methods*.

6.11.2 Scattered Data

We again concentrate on the bivariate setting. Now we consider data $((x_i, y_i), f(x_i, y_i))$, $i = 1, \ldots, n$, with no underlying grid structure. Figure 21 shows 100 randomly scattered data sites in the unit square $[0,1] \times [0,1]$.

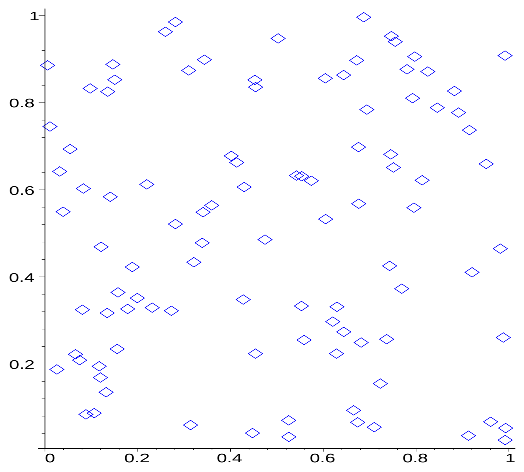

Figure 21: Scattered data with $n = 100$ on $[0,1] \times [0,1]$.

We can again start out by thinking about the use of (multivariate) polynomials. We will use the notation Π_k^d to denote the space of d-variate polynomials of total degree k, i.e.,

$$\Pi_k^d = \text{span}\{x_1^{i_1}x_2^{i_2}\cdots x_d^{i_d},\ 0 \le i_1 + i_2 + \ldots + i_d \le k\}.$$

For example, the space of linear bivariate polynomials Π_1^2 has the standard basis $\{1, x, y\}$, or the space of bivariate polynomials of total degree at most 2 is spanned by $\{1, x, y, x^2, xy, y^2\}$. In general, we have

$$\dim \Pi_k^d = \binom{k+d}{d}$$

(see HW problem 6.10.3 for the trivariate case).

Therefore, if there are n pieces of (bivariate) data to be interpolated we should use a space of (bivariate) polynomials such that $\dim \Pi_k^2 = \binom{k+2}{2} = n$, or $k = (\sqrt{8n+1} - 3)/2$. However, even if we are able to match up these dimensions, the Mairhuber-Curtis Theorem tells us that solvability of the interpolation problem depends on the *locations* of the data sites.

Some (sufficient) conditions are known on the locations of the data sites:

- As we saw in the previous section, if the data is gridded, then tensor-product polynomials can be used.
- If the data sites are arranged in a triangular array of $k+1$ "lines" as in Figure 22, then polynomials of total degree k can be used. Figure 22 shows 10 data points in $\mathbb{R}^2$, so cubic polynomials could be used to uniquely interpolate data at these points (since $\dim \Pi_3^2 = \binom{3+2}{2} = 10$).
- A more general condition is listed as Theorem 3 in the textbook on page 427.

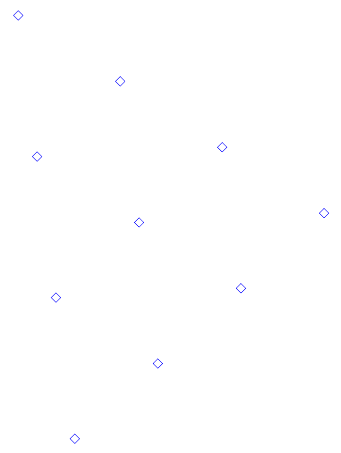

Figure 22: 10 data points arranged in a triangular array of 4 "lines".

We now turn to an overview of methods that are guaranteed to yield a unique interpolant for arbitrary scattered (bivariate) data.

Splines on Triangulations / Finite Elements

The general approach is as follows:

1. Create a triangulation of the convex hull of the data sites. A standard algorithm for doing this is the *Delaunay triangulation* algorithm which aims at maximizing the smallest angle in the triangulation. The reason for this is that long "skinny" triangles quickly lead to numerical instabilities. The resulting triangulation is the dual of the so-called *Voronoi* (or Dirichlet) *tesselation*. Figure 23 shows both the Delaunay triangulation and Voronoi tesselation of 20 scattered points in the unit square.

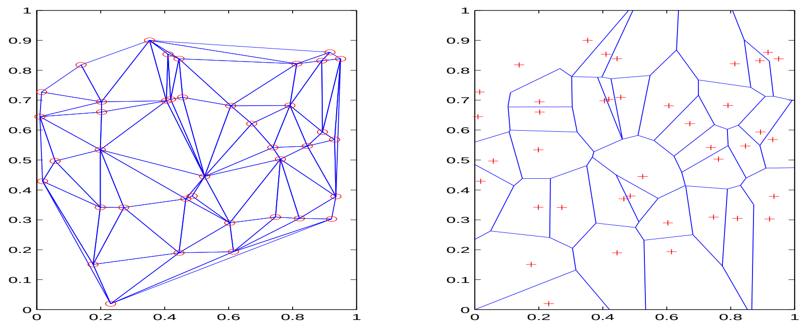

Figure 23: Delaunay triangulation (left) and Voronoi tesselation (right) of 20 scattered points in $[0,1] \times [0,1]$.

2. Define a polynomial p on each triangle that matches the data at the vertices. Depending on what the nature of the data is (e.g., derivative information may also be specified) we may want to enforce smoothness conditions across the triangle boundaries in order to obtain an overall smooth interpolant ($\rightarrow$ spline function).

Constructing general bivariate spline functions is not an easy task. We now discuss two of the simplest constructions.

Example 1: The simplest bivariate spline function is piecewise continuous and linear, and can be seen as the analogue of the "connect-the-dots" approach in the univariate case. After constructing the Delaunay triangulation, we simply fit a plane to the values at the vertices of each triangle. Clearly, the data are interpolated, and two neighboring triangles have a common (linear) boundary so that the overall spline function is continuous. An example of a C^0 piecewise linear spline function is shown in Figure 24. This method is usually the basis for simple rendering procedures in computer graphics.

Example 2: Another popular choice is the use of cubic polynomials on each triangle, and then to construct an overall C^1 interpolating spline function. It is one of the most tantalizing open problems in bivariate spline theory whether a solution to this problem exists in general. It would seem that the availability of 10 parameters per triangle

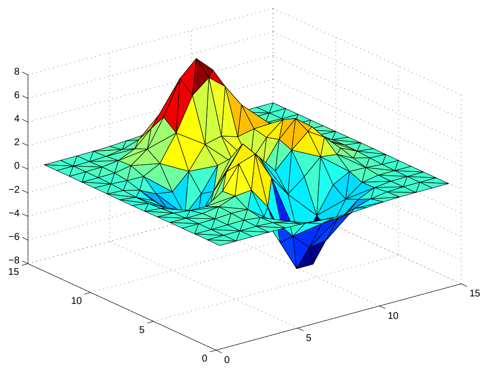

Figure 24: A C^0 piecewise linear spline function.

should be sufficient to satisfy all interpolation and smoothness constraints, but due to the global nature of this problem, an analysis has so far eluded the efforts of researchers.

If, however, we are willing to split each triangle in the original triangulation at its centroid into three subtriangles according to the so-called *Clough-Tocher split* (see Figure 25), then the problem becomes local, and is relatively easy to solve. In fact, now it is possible to specify function value and gradient value at each vertex (of the original triangulation) as well as a cross-derivative at the middle of each interior edge of the original triangulation, and an overall C^1 spline function is obtained. In the finite element literature these kind of elements are referred to as *Clough-Tocher finite elements*.

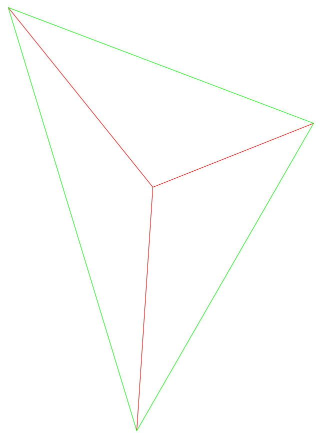

Figure 25: Clough-Tocher split of a (macro) triangle.

Usually one does not represent the polynomial pieces in the standard basis, but instead uses the so-called *Bernstein-Bézier* form, i.e., a polynomial of total degree n is written in the form

$$p(x,y) = \sum_{i+j+k=n} c_{ijk} B^n_{ijk}(x,y),$$

where

$$B^n_{ijk}(x,y) = \frac{n!}{i!j!k!} x^i y^j (1-x-y)^k$$

are the *Bernstein-Bézier basis polynomials*, and the c_{ijk} are coefficients.

Example:

(a) For $n = 1$ we have the space of linear bivariate polynomials Π^2_1, and can represent it as

$$\text{span}\{1, x, y\} \quad \text{or} \quad \text{span}\{x, y, 1-x-y\}.$$

(b) For $n = 2$ we have the space of quadratic bivariate polynomials Π^2_2, and can represent it as

$$\text{span}\{1, x, y, x^2, xy, y^2\} \quad \text{or} \quad \text{span}\{x^2, xy, y^2, x(1-x-y), y(1-x-y), (1-x-y)^2\}.$$

Remark: Besides the literature cited earlier (and many original papers on bivariate splines), Peter Alfeld of the University of Utah maintains an excellent website with information about multivariate splines at `http://www.math.utah.edu/~alfeld/MDS/index.html`.

Radial Basis Functions

The most natural way around the restrictions encountered in higher-dimensional spaces described in the Mairhuber-Curtis Theorem, is to work with interpolation (or approximation) spaces that depend in a natural way on the location of the data sites. This leads to the idea of so-called *radial basis functions* (or RBFs).

We now consider general scattered multivariate data

$$(\mathbf{x}_i, f(\mathbf{x}_i)), \ i = 1, \ldots, n, \quad \text{where} \quad \mathbf{x}_i \in \Omega \subset \mathbb{R}^d, \ \text{and} \ f(\mathbf{x}_i) \in \mathbb{R},$$

with arbitrary $d \geq 1$. The approximation space is chosen to be generated by translations of one single *basic* function φ, i.e.,

$$\mathcal{S} = \text{span}\{\varphi(\|\cdot - \mathbf{z}_1\|), \ldots, \varphi(\|\cdot - \mathbf{z}_m\|)\}.$$

The composition with the (Euclidean) norm is not necessary to satisfy the constraints of the Mairhuber-Curtis Theorem, but it has the advantage that the problem essentially becomes a univariate problem, as φ is now a univariate function. The points $\mathbf{z}_1, \ldots, \mathbf{z}_m$, are usually referred to as *centers* or *knots*. Moreover, in order to find an interpolant to the given data, one usually lets the centers coincide with the data sites. Then the interpolant will be of the form (with $m = n$)

$$s(\mathbf{x}) = \sum_{j=1}^{n} c_j \varphi(\|\mathbf{x} - \mathbf{x}_j\|), \qquad \mathbf{x} \in \mathbb{R}^d, \tag{39}$$

and the coefficients c_j are determined by solving the linear system

$$Ac = f, \tag{40}$$

that arises from the interpolation conditions

$$s(\mathbf{x}_i) = f(\mathbf{x}_i), \qquad i = 1, \ldots, n.$$

Here the matrix A has entries $\varphi(\|\mathbf{x}_i - \mathbf{x}_j\|)$.

Theorem 6.48 *The interpolation matrix A in (40) is nonsingular for any set of distinct data sites $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ provided the d-variate function Φ defined by $\Phi(\mathbf{x}) = \varphi(\|\mathbf{x}\|)$ is strictly positive definite.*

Proof: Next semester. ♠

In order to understand this theorem we need

Definition 6.49 *A function $f : \mathbb{R}^d \to \mathbb{R}$ is* positive definite *if for all n, all sets of pairwise distinct $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, and all $c \in \mathbb{R}^n$ we have*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j f(\mathbf{x}_i - \mathbf{x}_j) \geq 0. \tag{41}$$

If equality in (41) implies $c_1 = \ldots = c_n = 0$, then f is called strictly positive definite*.*

Remarks:

1. The terminology in Definition 6.49 is somewhat "nonstandard" as it does not match the one from linear algebra. However, positive definite functions were first defined in the 1920s by Bochner and others, and at that time there seemed to be no need for the "strict" case. This part of the definition was only added in the 1980s when Charles Micchelli discovered the connection between radial basis functions and positive definite functions and proved (a stronger form of) Theorem 6.48.
2. The generalization of Theorem 6.48 involves so-called *strictly conditionally positive definite* functions.
3. Of course, any kind of function φ that makes A nonsingular will work, but the complete characterization of such functions is still open.

Examples of RBFs

We use the abbreviation $r = \|\mathbf{x} - \mathbf{x}_j\|$. Then some of the most popular radial basic functions are:

1. Multiquadrics:
$$\varphi(r) = \sqrt{r^2 + \alpha^2}, \quad \alpha > 0,$$
with parameter α. These functions have an associated function Φ that is strictly conditionally negative definite of order one.

2. Inverse multiquadrics:
$$\varphi(r) = \frac{1}{\sqrt{r^2+\alpha^2}}, \quad \alpha > 0,$$
with parameter α. These functions give rise to strictly positive definite functions.

3. Gaussians:
$$\varphi(r) = e^{-\alpha^2 r^2}, \quad \alpha > 0,$$
with parameter α. These functions give rise to strictly positive definite functions.

4. Thin plate splines (in $\mathbb{R}^2$):
$$\varphi(r) = r^2 \ln r.$$
These functions give rise to strictly conditionally positive definite functions of order two.

5. Wendland's compactly supported RBFs. There is a whole family of these functions. E.g.,
$$\varphi(r) = (1-r)_+^4(4r+1)$$
gives rise to strictly positive definite functions as long as $d \leq 3$.

To find a scattered data interpolant using RBFs one needs to solve the linear system (40). This system will be a dense system for RBFs 1–4, and sparse for those listed in 5. Presently, one of the major research focusses is efficient solution of the system (40) and fast evaluation of the expansion (39).

Remarks:

1. The functions listed under 1–3 have spectral approximation order, i.e., they can approximate a sufficiently smooth function arbitrarily closely. This approximation is achieved by varying the parameter α. However, there is a trade-off principle involved, i.e., as the approximation quality is improved, the linear system (40) becomes more and more ill-conditioned.

2. RBF interpolation is illustrated in the Maple worksheet `578_RBF.mws`.

Moving Least Squares Approximation

We begin by discussing discrete weighted least squares approximation from the space of multivariate polynomials. Thus, we consider data
$$(\mathbf{x}_i, f(\mathbf{x}_i)),\ i = 1, \ldots, n, \quad \text{where} \quad \mathbf{x}_i \in \Omega \subset \mathbb{R}^d, \text{ and } f(\mathbf{x}_i) \in \mathbb{R},$$
with arbitrary $d \geq 1$. The approximation space is of the form
$$\mathcal{S} = \operatorname{span}\{p_1, \ldots, p_m\}, \qquad m < n,$$
with multivariate polynomials p_m.

We intend to find the best discrete weighted least squares approximation from $\mathcal{S}$ to some given function f, i.e., we need to determine the coefficients c_j in

$$s(\mathbf{x}) = \sum_{j=1}^{m} c_j p_j(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^d,$$

such that

$$\|f - s\|_{2,w} \to \min.$$

Here the norm is defined via the discrete (pseudo) inner product

$$\langle f, g \rangle_w = \sum_{i=1}^{n} f(\mathbf{x}_i) g(\mathbf{x}_i) w_i$$

with scalar weights w_i, $i = 1, \ldots, n$. The norm is then

$$\|f\|_{2,w}^2 = \sum_{i=1}^{n} [f(\mathbf{x}_i)]^2 \, w_i.$$

From Section 6.8 we know that the best approximation s is characterized by

$$\begin{aligned} f - s \perp \mathcal{S} \qquad &\iff \qquad \langle f - s, p_k \rangle_w = 0, \quad k = 1, \ldots, m, \\ &\iff \quad \langle f - \sum_{j=1}^{m} c_j p_j, p_k \rangle_w = 0 \\ &\iff \quad \sum_{j=1}^{m} c_j \langle p_j, p_k \rangle_w = \langle f, p_k \rangle_w, \quad k = 1, \ldots, m. \end{aligned}$$

These are of course the normal equations associated with this problem.

Now we discuss *moving* least squares approximation. The difference to the previous case is that now the weights w_i *move* with the evaluation point $\mathbf{x}$, i.e., we have a new inner product

$$\langle f, g \rangle_{w(\mathbf{x})} = \sum_{i=1}^{n} f(\mathbf{x}_i) g(\mathbf{x}_i) w_i(\mathbf{x}).$$

Therefore, the normal equations become

$$\sum_{j=1}^{m} c_j \langle p_j, p_k \rangle_{w(\mathbf{x})} = \langle f, p_k \rangle_{w(\mathbf{x})}, \qquad k = 1, \ldots, m.$$

Note that now the coefficients c_j have an implicit dependence on the evaluation point $\mathbf{x}$. Therefore, the moving least squares best approximation from $\mathcal{S}$ is given by

$$s(\mathbf{x}) = \sum_{j=1}^{m} c_j(\mathbf{x}) p_j(\mathbf{x}).$$

Remark: This means that for every evaluation of s at a different point $\mathbf{x}$ the system of normal equations needs to be solved anew. This (seemingly) high computational cost was one of the main reasons the moving least squares method was largely ignored after it discovery in the early 1980s.

Example 1: We take $\mathcal{S} = \text{span}\{1\}$ with d arbitrary. Then $m = 1$, and there is only one normal equation

$$c_1 \langle p_1, p_1 \rangle_{w(\mathbf{x})} = \langle f, p_1 \rangle_{w(\mathbf{x})}$$

which we can easily solve for the unknown expansion coefficient, so that

$$c_1 = \frac{\langle f, 1 \rangle_{w(\mathbf{x})}}{\langle 1, 1 \rangle_{w(\mathbf{x})}}.$$

Thus,

$$s(\mathbf{x}) = c_1(\mathbf{x}) p_1(\mathbf{x}) = c_1(\mathbf{x}).$$

It is also easy to determine an explicit formula for $c_1(\mathbf{x})$. Since

$$\begin{aligned} \langle f, 1 \rangle_{w(\mathbf{x})} &= \sum_{i=1}^{n} f(\mathbf{x}_i) 1 w_i(\mathbf{x}), \\ \langle 1, 1 \rangle_{w(\mathbf{x})} &= \sum_{i=1}^{n} 1 \cdot 1 w_i(\mathbf{x}), \end{aligned}$$

we have

$$\begin{aligned} s(\mathbf{x}) &= c_1(\mathbf{x}) = \sum_{i=1}^{n} f(\mathbf{x}_i) \frac{w_i(\mathbf{x})}{\sum\limits_{\ell=1}^{n} w_\ell(\mathbf{x})} \\ &= \sum_{i=1}^{n} f(\mathbf{x}_i) v_i(\mathbf{x}), \qquad (42) \end{aligned}$$

where

$$v_i(\mathbf{x}) = \frac{w_i(\mathbf{x})}{\sum\limits_{\ell=1}^{n} w_\ell(\mathbf{x})}.$$

This method is known as *Shepard's method* (see Computer homework problem 6.10#1).

Remarks:

1. First, notice that the expansion (42) is an explicit formula for the best moving least squares approximation – valid for any evaluation point $\mathbf{x}$. Repeated solution of the normal equations is not required, since we solved the one normal equation once and for all analytically.

2. The functions v_i, $i = 1, \ldots, n$, can be seen as (approximate) cardinal basis functions. They are true cardinal functions if the weight functions are chosen appropriately (see the remarks regarding the choice of weight functions below). Expansion (42) can be viewed as a quasi-interpolation formula for the data given by the function f.

3. Note that the functions v_i, $i = 1, \ldots, n$, form a partition of unity, i.e.,

$$\sum_{i=1}^{n} v_i(\mathbf{x}) = 1, \qquad \mathbf{x} \in \mathbb{R}^d.$$

   Therefore, if the data comes from a constant function, i.e., $f(\mathbf{x}_i) = C$, then

$$s(\mathbf{x}) = \sum_{i=1}^{n} f(\mathbf{x}_i) v_i(\mathbf{x}) = C \sum_{i=1}^{n} v_i(\mathbf{x}) = C,$$

   and Shepard's method reproduces constants. This can be used to show that, if h denotes the "meshsize" of the scattered data, then Shepard's method has approximation order $\mathcal{O}(h)$.

4. By reproducing higher-degree polynomials (and thus solving non-trivial systems of normal equations) one can achieve higher order of approximation with the moving least squares method.

5. We emphasize that the moving least squares method in general does not produce a function that interpolates the given data – only an approximation.

6. The moving least squares method can be generalized by choosing a different approximation space $\mathcal{S}$.

7. The smoothness of the overall approximation is governed by the smoothness of the functions in $\mathcal{S}$ as well as the smoothness of the weight functions.

Example 2:

If we take $\mathcal{S} = \text{span}\{1, x\}$ with $d = 1$, then $m = 2$, and the system of normal equations is

$$\sum_{j=1}^{2} c_j \langle p_j, p_k \rangle_{w(x)} = \langle f, p_k \rangle_{w(x)}, \quad k = 1, 2,$$

or

$$\begin{bmatrix} \langle p_1, p_1 \rangle_{w(x)} & \langle p_2, p_1 \rangle_{w(x)} \\ \langle p_1, p_2 \rangle_{w(x)} & \langle p_2, p_2 \rangle_{w(x)} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \langle f, p_1 \rangle_{w(x)} \\ \langle f, p_2 \rangle_{w(x)} \end{bmatrix}$$

$$\iff \quad \begin{bmatrix} \sum\limits_{i=1}^{n} w_i(x) & \sum\limits_{i=1}^{n} x_i w_i(x) \\ \sum\limits_{i=1}^{n} x_i w_i(x) & \sum\limits_{i=1}^{n} x_i^2 w_i(x) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \sum\limits_{i=1}^{n} f(x_i) w_i(x) \\ \sum\limits_{i=1}^{n} f(x_i) x_i w_i(x) \end{bmatrix}.$$

As mentioned above, this system has to be solved for every evaluation at a different value x. For a fixed value x we have

$$s(x) = \sum_{j=1}^{2} c_j(x) p_j(x) = c_1(x) + x c_2(x). \tag{43}$$

Remarks:

1. We can see that (43) represents the equation of a line. However, depending on the point of evaluation, a different line is chosen.
2. In the statistics literature this idea is known as local polynomial regression.

Choice of Weight Functions

1. In the early discussions of the moving least squares method (in the early 1980s by Lancaster & Šalkauskas) so-called *inverse distance* weight functions of the form

$$w_i(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_i\|^{-\alpha}, \quad \alpha > 0,$$

were suggested. Note that $w_i(\mathbf{x}) \to \infty$ as $\mathbf{x} \to \mathbf{x}_i$. Therefore, using this kind of weight function leads to interpolation of the given data.
2. The RBFs of the previous section can be used as weight functions.
3. B-splines (composed with the Euclidean norm if $d > 1$) have also been suggested as (radial) weight functions.
4. Recent work by Fasshauer suggests that certain polynomials with a radial argument can be used to construct Shepard-like functions with arbitrarily high approximation orders.

Remark: If n is large, then it is more efficient to have local weight functions since not all terms in $\sum_{i=1}^{n} \ldots w_i(\mathbf{x})$ coming from the inner product will be "active" – only those for which $\mathbf{x}_i$ is close to $\mathbf{x}$.