# CHAPTER 8

# Quasi-interpolation methods

In Chapter 5 we considered a number of methods for computing spline approximations. The starting point for the approximation methods is a data set that is usually discrete and in the form of function values given at a set of abscissas. The methods in Chapter 5 roughly fall into two categories: global methods and local methods. A global method is one where any B-spline coefficient depends on all initial data points, whereas a local method is one where a B-spline coefficient only depends on data points taken from the neighbourhood of the support of the corresponding B-spline. Typical global methods are cubic spline interpolation and least squares approximation, while cubic Hermite interpolation and the Schoenberg variation diminishing spline approximation are popular local methods.

In this chapter we are going to describe a general recipe for developing local spline approximation methods. This will enable us to produce an infinite number of approximation schemes that can be tailored to any special needs that we may have or that our given data set dictates. In principle, the methods are local, but by allowing the area of influence for a given B-spline coefficient to grow, our general recipe may even encompass the global methods in Chapter 5.

The recipe we describe produces approximation methods known under the collective term *quasi-interpolation methods*. Their advantage is their flexibility and their simplicity. There is considerable freedom in the recipe to produce tailor-made approximation schemes for initial data sets with special structure. Quasi-interpolants also allow us to establish important properties of B-splines. In the next chapter we will employ them to study how well a given function can be approximated by splines, and to show that B-splines form a stable basis for splines.

## 8.1 A general recipe

A spline approximation method consists of two main steps: First the degree and knot vector are determined, and then the B-spline coefficients of the approximation are computed from given data according to some formula. For some methods like spline interpolation and least squares approximation, this formula corresponds to the solution of a linear system of equations. In other cases, like cubic Hermite interpolation and Schoenberg's Variation Diminishing spline approximation, the formula for the coefficients is given directly in terms of given values of the function to be interpolated.

### 8.1.1 The basic idea

The basic idea behind the construction of quasi-interpolants is very simple. We focus on how to compute the B-spline coefficients of the approximation and assume that the degree and knot vector are known. The procedure depends on two versions of the local support property of B-splines that we know well from earlier chapters: (i) The B-spline $B_j$ is nonzero only within the interval $[t_j, t_{j+d+1}]$, and (ii) on the interval $[t_\mu, t_{\mu+1})$ there are only $d+1$ B-splines in $\mathbb{S}_{d,\boldsymbol{t}}$ that are nonzero so a spline $g$ in $\mathbb{S}_{d,\boldsymbol{t}}$ can be written as $g(x) = \sum_{i=\mu-d}^{\mu} b_i B_i(x)$ when $x$ is restricted to this interval.

Suppose we are to compute an approximation $g = \sum_i c_i B_i$ in $\mathbb{S}_{d,\boldsymbol{t}}$ to a given function $f$. To compute $c_j$ we can select one knot interval $I = [t_\mu, t_{\mu+1}]$ which is a subinterval of $[t_j, t_{j+d+1}]$. We denote the restriction of $f$ to this interval by $f^I$ and determine an approximation $g^I = \sum_{i=\mu-d}^{\mu} b_i B_i$ to $f^I$. One of the coefficients of $g^I$ will be $b_j$ and we fix $c_j$ by setting $c_j = b_j$. The whole procedure is then repeated until all the $c_i$ have been determined.

It is important to note the flexibility of this procedure. In choosing the interval $I$ we will in general have the $d+1$ choices $\mu = j,\, j,\, \ldots,\, j+d$ (fewer if there are multiple knots). As we shall see below we do not necessarily have to restrict $I$ to be one knot interval; all that is required is that $I \cap [t_\mu, t_{\mu+d+1}]$ is nonempty. When approximating $f^I$ by $g^I$ we have a vast number of possibilities. We may use interpolation or least squares approximation, or any other approximation method. Suppose we settle for interpolation, then we have complete freedom in choosing the interpolation points within the interval $I$. In fact, there is so much freedom that we can have no hope of exploring all the possibilities.

It turns out that some of this freedom is only apparent — to produce useful quasi-interpolants we have to enforce certain conditions. With the general setup described above, a useful restriction is that if $f^I$ should happen to be a polynomial of degree $d$ then $g^I$ should reproduce $f^I$, i.e., in this case we should have $g^I = f^I$. This has the important consequence that if $f$ is a spline in $\mathbb{S}_{d,\boldsymbol{t}}$ then the approximation $g$ will reproduce $f$ exactly (apart from rounding errors in the numerical computations). To see why this is the case, suppose that $f = \sum_i \hat{b}_i B_i$ is a spline in $\mathbb{S}_{d,\boldsymbol{t}}$. Then $f^I$ will be a polynomial that can be written as $f^I = \sum_{i=\mu-d}^{\mu} \hat{b}_i B_i$. Since we have assumed that polynomials will be reproduced we know that $g^I = f^I$ so $\sum_{i=\mu-d}^{\mu} b_i B_i = \sum_{i=\mu-d}^{\mu} \hat{b}_i B_i$, and by the linear independence of the B-splines involved we conclude that $b_i = \hat{b}_i$ for $i = \mu - d,\, \ldots,\, \mu$. But then we see that $c_j = b_j = \hat{b}_j$ so $g$ will agree with $f$. An approximation scheme with the property that $Pf = f$ for all $f$ in a space $\mathbb{S}$ is to *reproduce* the space.

### 8.1.2 A more detailed description

Hopefully, the basic idea behind the construction of quasi-interpolants became clear above. In this section we describe the construction in some more detail with the generalisations mentioned before. We first write down the general procedure for determining quasi-interpolants and then comment on the different steps afterwards.

**Algorithm 8.1** (Construction of quasi-interpolants)**.** *Let the spline space $\mathbb{S}_{d,\boldsymbol{t}}$ of dimension $n$ and the real function $f$ defined on the interval $[t_{d+1}, t_{n+1}]$ be given, and suppose that $\boldsymbol{t}$ is a $d+1$-regular knot vector. To approximate $f$ from the space $\mathbb{S}_{d,\boldsymbol{t}}$ perform the following steps for $j = 1,\, 2,\, \ldots,\, n$:*

1. *Choose a subinterval $I = [t_\mu, t_\nu]$ of $[t_{d+1}, t_{n+1}]$ with the property that $I \cap (t_j, t_{j+d+1})$ is nonempty, and let $f^I$ denote the restriction of $f$ to this interval.*

2. *Choose a local approximation method $P^I$ and determine an approximation $g^I$ to $f^I$,*

$$g^I = P^I f^I = \sum_{i=\mu-d}^{\nu-1} b_i B_i, \tag{8.1}$$

   *on the interval $I$.*

3. *Set coefficient $j$ of the global approximation $Pf$ to $b_j$, i.e.,*

$$c_j = b_j.$$

*The spline $Pf = \sum_{j=1}^{n} c_j B_j$ will then be an approximation to $f$.*

The coefficient $c_j$ obviously depends on $f$ and this dependence on $f$ is often indicated by using the notation $\lambda_j f$ for $c_j$. This will be our normal notation in the rest of the chapter.

An important point to note is that the restriction $\mathbb{S}_{d,\boldsymbol{t},I}$ of the spline space $\mathbb{S}_{d,\boldsymbol{t}}$ to the interval $I$ can be written as a linear combination of the B-splines $\{B_i\}_{i=\mu-d}^{\nu-1}$. These are exactly the B-splines whose support intersect the interior of the interval $I$, and by construction, one of them must clearly be $B_j$. This ensures that the coefficient $b_j$ that is needed in step 3 is computed in step 2.

Algorithm 8.1 generalizes the simplified procedure in Section 8.1.1 in that $I$ is no longer required to be a single knot interval in $[t_j, t_{j+d+1}]$. This gives us considerably more flexibility in the choice of local approximation methods. Note in particular that the classical global methods are included as special cases since we may choose $I = [t_{d+1}, t_{n+1}]$.

As we mentioned in Section 8.1.1, we do not get good approximation methods for free. If $Pf$ is going to be a decent approximation to $f$ we must make sure that the local methods used in step 2 reproduce polynomials or splines.

**Lemma 8.2.** *Suppose that all the local methods used in step 2 of Algorithm 8.1 reproduce all polynomials of some degree $d_1 \leq d$. Then the global approximation method $P$ will also reproduce polynomials of degree $d_1$. If all the local methods reproduce all the splines in $\mathbb{S}_{d,\boldsymbol{t},I}$ then $P$ will reproduce the whole spline space $\mathbb{S}_{d,\boldsymbol{t}}$.*

**Proof.** The proof of both claims follow just as in the special case in Section 8.1.1, but let us even so go through the proof of the second claim. We want to prove that if all the local methods $P^I$ reproduce the local spline spaces $\mathbb{S}_{d,\boldsymbol{t},I}$ and $f$ is a spline in $\mathbb{S}_{d,\boldsymbol{t}}$, then $Pf = f$. If $f$ is in $\mathbb{S}_{d,\boldsymbol{t}}$ we clearly have $f = \sum_{i=1}^{n} \hat{b}_i B_i$ for appropriate coefficients $(\hat{b}_i)_{i=1}^n$, and the restriction of $f$ to $I$ can be represented as $f^I = \sum_{i=\mu-d}^{\nu-1} \hat{b}_i B_i$. Since $P^I$ reproduces $\mathbb{S}_{d,\boldsymbol{t},I}$ we will have $P^I f^I = f^I$ or

$$\sum_{i=\mu-d}^{\nu-1} b_i B_i = \sum_{i=\mu-d}^{\nu-1} \hat{b}_i B_i.$$

The linear independence of the B-splines involved over the interval $I$ then allows us to conclude that $b_i = \hat{b}_i$ for all indices $i$ involved in this sum. Since $j$ is one the indices we therefore have $c_j = b_j = \hat{b}_j$. When this holds for all values of $j$ we obviously have $Pf = f$. ∎

The reader should note that if $I$ is a single knot interval, the local spline space $\mathbb{S}_{d,\boldsymbol{t},I}$ reduces to the space of polynomials of degree $d$. Therefore, when $I$ is a single knot interval, local reproduction of polynomials of degree $d$ leads to global reproduction of the whole spline space.

Why does reproduction of splines or polynomials ensure that $P$ will be a good approximation method? We will study this in some detail in Chapter 9, but as is often the case the basic idea is simple: The functions we want to approximate are usually nice and smooth, like the exponential functions or the trigonometric functions. An important property of polynomials is that they approximate such smooth functions well, although if the interval becomes wide we may need to use polynomials of high degree. A quantitative manifestation of this phenomenon is that if we perform a Taylor expansion of a smooth function, then the error term will be small, at least if the degree is high enough. If our approximation method reproduces polynomials it will pick up the essential behaviour of the Taylor polynomial, while the approximation error will pick up the essence of the error in the Taylor expansion. The approximation method will therefore perform well whenever the error in the Taylor expansion is small. If we reproduce spline functions we can essentially reproduce Taylor expansions on each knot interval as long as the function we approximate has at least the same smoothness as the splines in the spline space we are using. So instead of increasing the polynomial degree because we are approximating over a wide interval, we can keep the spacing in the knot vector small and thereby keep the polynomial degree of the spline low. Another way to view this is that by using splines we can split our function into suitable pieces that each can be approximated well by polynomials of relatively low degree, even though this is not possible for the complete function. By constructing quasi-interpolants as outlined above we obtain approximation methods that actually utilise this approximation power of polynomials on each subinterval. In this way we can produce good approximations even to functions that are only piecewise smooth.

## 8.2 Some quasi-interpolants

It is high time to try out our new tool for constructing approximation methods. Let us see how some simple methods can be obtained from Algorithm 8.1.

### 8.2.1 Piecewise linear interpolation

Perhaps the simplest, local approximation method is piecewise linear interpolation. We assume that our $n$-dimensional spline space $\mathbb{S}_{1,\boldsymbol{t}}$ is given and that $\boldsymbol{t}$ is a 2-regular knot vector. For simplicity we also assume that all the interior knots are simple. The function $f$ is given on the interval $[t_2, t_{n+1}]$. To determine $c_j$ we choose the local interval to be $I = [t_j, t_{j+1}]$. In this case, we have no interior knots in $I$ so $\mathbb{S}_{1,\boldsymbol{t},I}$ is the two dimensional space of linear polynomials. A basis for this space is given by the two linear B-splines $B_{j-1}$ and $B_j$, restricted to the interval $I$. A natural candidate for our local approximation method is interpolation at $t_j$ and$t_{j+1}$. On the interval $I$, the B-spline $B_{j-1}$ is a straight line with value 1 at $t_j$ and value 0 at $t_{j+1}$, while $B_j$ is a straight line with value 0 at $t_j$ and value 1 at $t_{j+1}$. The local interpolant can therefore be written

$$P_1^I f(x) = f(t_j)B_{j-1}(x) + f(t_{j+1})B_j(x).$$

From Algorithm 8.1 we know that the coefficient multiplying $B_j$ is the one that should multiply $B_j$ also in our global approximation, in other words $c_j = \lambda_j f = f(t_{j+1})$. The

global approximation is therefore

$$P_1 f(x) = \sum_{j=1}^{n} f(t_{j+1}) B_j(x).$$

Since a straight line is completely characterized by its value at two points, the local approximation will always give zero error and therefore reproduce all linear polynomials. Then we know from Lemma 8.2 that $P_1$ will reproduce all splines $\mathbb{S}_{1,\boldsymbol{t}}$.

This may seem like unnecessary formalism in this simple case where the conclusions are almost obvious, but it illustrates how the construction works in a very transparent situation.

### 8.2.2 A 3-point quadratic quasi-interpolant

In our repertoire of approximation methods, we only have one local, quadratic method, Schoenberg's variation diminishing spline. With the quasi-interpolant construction it is easy to construct alternative, local methods. Our starting point is a quadratic spline space $\mathbb{S}_{2,\boldsymbol{t}}$ based on a 3-regular knot vector with distinct interior knots, and a function $f$ to be approximated by a scheme which we denote $P_2$. The support of the B-spline $B_j$ is $[t_j, t_{j+3}]$, and we choose our local interval as $I = [t_{j+1}, t_{j+2}]$. Since $I$ is one knot interval, we need a local approximation method that reproduces quadratic polynomials. One such method is interpolation at three distinct points. We therefore choose three distinct points $x_{j,0}$, $x_{j,1}$ and $x_{j,2}$ in $I$. Some degree of symmetry is always a good guide so we choose

$$x_{j,0} = t_{j+1}, \quad x_{j,1} = \frac{t_{j+1} + t_{j+2}}{2}, \quad x_{j,2} = t_{j+2}.$$

To determine $P_2^I f$ we have to solve the linear system of three equations in the three unknowns $b_{j-1}$, $b_j$ and $b_{j+1}$ given by

$$P_2^I f(x_{j,k}) = \sum_{i=j-1}^{j+1} b_i B_i(x_{j,k}) = f(x_{j,k}), \quad \text{for } k = 0,\, 1,\, 2.$$

With the aid of a tool like Mathematica we can solve these equations symbolically. The result is that

$$b_j = \frac{1}{2}\big(-f(t_{j+1}) + 4f(t_{j+3/2}) - f(t_{j+2})\big),$$

where $t_{j+3/2} = (t_{j+1} + t_{j+2})/2$. The expressions for $b_{j-1}$ and $b_{j+1}$ are much more complicated and involve the knots $t_j$ and $t_{j+3}$ as well. The simplicity of the expression for $b_j$ stems from the fact that $x_{j,1}$ was chosen as the midpoint between $t_{j+1}$ and $t_{j+2}$.

The expression for $b_j$ is valid whenever $t_{j+1} < t_{j+2}$ which is not the case for $j = 1$ and $j = n$ since $t_1 = t_2 = t_3$ and $t_{n+1} = t_{n+2} = t_{n+3}$. But from Lemma 2.12 we know that any spline $g$ in $\mathbb{S}_{3,\boldsymbol{t}}$ will interpolate its first and last B-spline coefficient at these points so we simply set $c_1 = f(t_1)$ and $c_n = f(t_{n+1})$.

Having constructed the local interpolants, we have all the ingredients necessary to

construct the quasi-interpolant $P_2 f = \sum_{j=1}^{n} \lambda_j f B_j$, namely

$$\lambda_j f = \begin{cases} f(t_1), & \text{when } j = 1; \\ \frac{1}{2}(-f(x_{j,0}) + 4f(x_{j,1}) - f(x_{j,2}), & \text{when } 1 < j < n; \\ f(t_{n+1}), & \text{when } j = n. \end{cases}$$

Since the local approximation reproduced the local spline space (the space of quadratic polynomials in this case), the complete quasi-interpolant will reproduce the whole spline space $\mathbb{S}_{2,\boldsymbol{t}}$.

### 8.2.3 A 5-point cubic quasi-interpolant

The most commonly used splines are cubic, so let us construct a cubic quasi-interpolant. We assume that the knot vector is 4-regular and that the interior knots are all distinct. As usual we focus on the coefficient $c_j = \lambda_j f$. It turns out that the choice $I = [t_{j+1}, t_{j+3}]$ is convenient. The local spline space $\mathbb{S}_{3,\boldsymbol{t},I}$ has dimension 5 and is spanned by the (restriction of the) B-splines $\{B_i\}_{i=j-2}^{j+2}$. We want the quasi-interpolant to reproduce the whole spline space and therefore need $P^I$ to reproduce $\mathbb{S}_{3,\boldsymbol{t},I}$. We want to use interpolation as our local approximation method, and we know from Chapter 5 that spline interpolation reproduces the spline space as long as it has a unique solution. The solution is unique if the coefficient matrix of the resulting linear system is nonsingular, and from Theorem 5.18 we know that a B-spline coefficient matrix is nonsingular if and only if its diagonal is positive. Since the dimension of $\mathbb{S}_{3,\boldsymbol{t},I}$ is 5 we need 5 interpolation points. We use the three knots $t_{j+1}$, $t_{j+2}$ and $t_{j+3}$ and one point from each of the knot intervals in $I$,

$$x_{j,0} = t_{j+1}, \quad x_{j,1} \in (t_{j+1}, t_{j+2}), \quad x_{j,2} = t_{j+2}, \quad x_{j,3} \in (t_{j+2}, t_{j+3}), \quad x_{j,4} = t_{j+3}.$$

Our local interpolation problem is

$$\sum_{i=j-2}^{j+2} b_i B_i(x_{j,k}) = f(x_{j,k}), \quad \text{for } k = 0, 1, \ldots, 4.$$

In matrix-vector form this becomes

$$\begin{pmatrix} B_{j-2}(x_{j,0}) & B_{j-1}(x_{j,0}) & 0 & 0 & 0 \\ B_{j-2}(x_{j,1}) & B_{j-1}(x_{j,1}) & B_j(x_{j,1}) & B_j(x_{j,1}) & 0 \\ B_{j-2}(x_{j,2}) & B_{j-1}(x_{j,2}) & B_j(x_{j,2}) & B_j(x_{j,2}) & B_j(x_{j,2}) \\ 0 & B_{j-1}(x_{j,3}) & B_j(x_{j,3}) & B_j(x_{j,3}) & B_j(x_{j,3}) \\ 0 & 0 & 0 & B_j(x_{j,4}) & B_j(x_{j,4}) \end{pmatrix} \begin{pmatrix} b_{j-2} \\ b_{j-1} \\ b_j \\ b_{j+1} \\ b_{j+2} \end{pmatrix} = \begin{pmatrix} f(x_{j,0}) \\ f(x_{j,1}) \\ f(x_{j,2}) \\ f(x_{j,3}) \\ f(x_{j,4}) \end{pmatrix}$$

when we insert the matrix entries that are zero. Because of the way we have chosen the interpolation points we see that all the entries on the diagonal of the coefficient matrix will be positive so the matrix is nonsingular. The local problem therefore has a unique solution and will reproduce $\mathbb{S}_{3,\boldsymbol{t},I}$. The expression for $\lambda_j f$ is in general rather complicated, but in the special case where the width of the two knot intervals is equal and $x_{j,2}$ and $x_{j,4}$ are chosen as the midpoints of the two intervals we end up with

$$\lambda_j f = \frac{1}{6}\big(f(t_{j+1}) - 8f(t_{j+3/2}) + 20f(t_{j+2}) - 8f(t_{j+5/2}) + f(t_{j+3})\big)$$

where $t_{j+3/2} = (t_{j+1} + t_{j+2})/2$ and $t_{j+5/2} = (t_{j+2} + t_{j+3})/2$. Unfortunately, this formula is not valid when $j = 1$, 2, $n-1$ or $n$ since then one or both of the knot intervals in $I$ collapse to one point. However, our procedure is sufficiently general to derive alternative formulas for computing the first two coefficients. The first value of $j$ for which the general procedure works is $j = 3$. In this case $I = [t_4, t_6]$ and our interpolation problem involves the B-splines $\{B_i\}_{i=1}^5$. This means that when we solve the local interpolation problem we obtain B-spline coefficients multiplying all of these B-splines, including $B_1$ and $B_2$. There is nothing stopping us from using the same interval $I$ for computation of several coefficients, so in addition to obtaining $\lambda_3 f$ from this local interpolant, we also use it as our source for the first two coefficients. In the special case when the interior knots are uniformly distributed and $x_{3,1} = t_{9/2}$ and $x_{3,3} = t_{11/2}$, we find

$$\begin{aligned}\lambda_1 f &= f(t_4),\\ \lambda_2 f &= \frac{1}{18}\big(-5f(t_4) + 40f(t_{9/2}) - 36f(t_5) + 18f(t_{11/2}) - f(t_6)\big).\end{aligned}$$

In general, the second coefficient will be much more complicated, but the first one will not change.

This same procedure can obviously be used to determine values for the last two coefficients, and under the same conditions of uniformly distributed knots and interpolation points we find

$$\begin{aligned}\lambda_{n-1} f &= \frac{1}{18}\big(-f(t_{n-1}) + 18f(t_{n-1/2}) - 36f(t_n) + 40f(t_{n+1/2}) - 5f(t_{n+1})\big),\\ \lambda_n f &= f(t_{n+1}).\end{aligned}$$

### 8.2.4 Some remarks on the constructions

In all our constructions, we have derived specific formulas for the B-spline coefficients of the quasi-interpolants in terms of the function $f$ to be approximated, which makes it natural to use the notation $c_j = \lambda_j f$. To do this, we had to solve the local linear system of equations symbolically. When the systems are small this can be done quite easily with a computer algebra system like Maple or Mathematica, but the solutions quickly become complicated and useless unless the knots and interpolation points are nicely structured, preferably with uniform spacing. The advantage of solving the equations symbolically is of course that we obtain explicit formulas for the coefficients once and for all and can avoid solving equations when we approximate a particular function.

For general knots, the local systems of equations usually have to be solved numerically, but quasi-interpolants can nevertheless prove very useful. One situation is real-time processing of data. Suppose we are in a situation where data are measured and need to be fitted with a spline in real time. With a global approximation method we would have to recompute the whole spline each time we receive new data. This would be acceptable at the beginning, but as the data set grows, we would not be able to compute the new approximation quickly enough. We could split the approximation into smaller pieces at regular intervals, but quasi-interpolants seem to be a perfect tool for this kind of application. In a real-time application the data will often be measured at fixed time intervals, and as we have seen it is then easy to construct quasi-interpolants with explicit formulas for the coefficients. Even if this is not practicable because the explicit expressions are not

available or become too complicated, we just have to solve a simple, linear set of equations to determine each new coefficient. The important fact is that the size of the system is constant so that we can handle almost arbitrarily large data sets, the only limitation being available storage space.

Another important feature of quasi-interpolants is their flexibility. In our constructions we have assumed that the function we approximate can be evaluated at any point that we need. This may sometimes be the case, but often the function is only partially known by a few discrete, measured values at specific abscissas. The procedure for constructing quasi-interpolants has so much inherent freedom that it can be adapted in a number of ways to virtually any specific situation, whether the whole data set is available a priori or the approximation has to be produced in real-time as the data is generated.

## 8.3 Quasi-interpolants are linear operators

Now that we have seen some examples of quasi-interpolants, let us examine them from a more general point of view. The basic ingredient of quasi-interpolants is the computation of each B-spline coefficient, and we have have used the notation $c_j = \lambda_j f = \lambda_j(f)$ to indicate that each coefficient depends on $f$. It is useful to think of $\lambda_j$ as a 'function' that takes an ordinary function as input and gives a real number as output; such 'functions' are usually called functionals. If we go back and look at our examples, we notice that in each case the dependency of our coefficient functionals on $f$ is quite simple: The function values occur explicitly in the coefficient expressions and are not multiplied or operated on in any way other than being added together and multiplied by real numbers. This is familiar from linear algebra.

**Definition 8.3.** *In the construction of quasi-interpolants, each B-spline coefficient is computed by evaluating a* linear *functional. A linear functional $\lambda$ is a mapping from a suitable space of functions $\mathbb{S}$ into the real numbers $\mathbb{R}$ with the property that if $f$ and $g$ are two arbitrary functions in $\mathbb{S}$ and $\alpha$ and $\beta$ are two real numbers then*

$$\lambda(\alpha f + \beta g) = \alpha\lambda f + \beta\lambda g.$$

Linearity is a necessary property of a functional that is being used to compute B-spline coefficients in the construction of quasi-interpolants. If one of the coefficient functionals are not linear, then the resulting approximation method is not a quasi-interpolant. Linearity of the coefficient functionals leads to linearity of the approximation scheme.

**Lemma 8.4.** *Any quasi-interpolant $P$ is a linear operator, i.e., for any two admissible functions $f$ and $g$ and any real numbers $\alpha$ and $\beta$,*

$$P(\alpha f + \beta g) = \alpha Pf + \beta Pg.$$

**Proof.** Suppose that the linear coefficient functionals are $(\lambda_j)_{j=1}^n$. Then we have

$$P(\alpha f + \beta g) = \sum_{i=1}^{n} \lambda_j(\alpha f + \beta g)B_i = \alpha \sum_{i=1}^{n} \lambda_j f B_i + \beta \sum_{i=1}^{n} \lambda_j g B_i = \alpha Pf + \beta Pg$$

which demonstrates the linearity of $P$. ∎

This lemma is simple, but very important since there are so many powerful mathematical tools available to analyse linear operators. In Chapter 9 we are going to see how well a given function can be approximated by splines. We will do this by applying basic tools in the analysis of linear operators to some specific quasi-interpolants.

## 8.4 Different kinds of linear functionals and their uses

In our examples of quasi-interpolants in Section 8.2 the coefficient functionals were all linear combinations of function values, but there are other functionals that can be useful. In this section we will consider some of these and how they turn up in approximation problems.

### 8.4.1 Point functionals

Let us start by recording the form of the functionals that we have already encountered. The coefficient functionals in Section 8.2 were all in the form

$$\lambda f = \sum_{i=0}^{\ell} w_i f(x_i) \tag{8.2}$$

for suitable numbers $(w_i)_{i=0}^{\ell}$ and $(x_i)_{i=0}^{\ell}$. Functionals of this kind can be used if a procedure is available to compute values of the function $f$ or if measured values of $f$ at specific points are known. Most of our quasi-interpolants will be of this kind.

Point functionals of this type occur naturally in at least two situations. The first is when the local approximation method is interpolation, as in our examples above. The second is when the local approximation method is discrete least squares approximation. As a simple example, suppose our spline space is $\mathbb{S}_{2,\boldsymbol{t}}$ and that in determining $c_j$ we consider the single knot interval $I = [t_{j+1}, t_{j+2}]$. Suppose also that we have 10 function values at the points $(x_{j,k})_{k=0}^{9}$ in this interval. Since the dimension of $\mathbb{S}_{2,\boldsymbol{t},I}$ is 3, we cannot interpolate all 10 points. The solution is to perform a local least squares approximation and determine the local approximation by minimising the sum of the squares of the errors,

$$\min_{g \in \mathbb{S}_{2,\boldsymbol{t},I}} \sum_{k=0}^{9} \big(g(x_{j,k}) - f(x_{j,k})\big)^2.$$

The result is that $c_j$ will be a linear combination of the 10 function values,

$$c_j = \lambda_j f = \sum_{k=0}^{9} w_{j,k} f(x_{j,k}).$$

### 8.4.2 Derivative functionals

In addition to function values, we can also compute derivatives of a function at a point. Since differentiation is a linear operator it is easy to check that a functional like $\lambda f = f''(x_i)$ is linear. The most general form of a derivative functional based at a point that we will consider is

$$\lambda f = \sum_{k=0}^{r} w_k f^{(k)}(x)$$

where $x$ is a suitable point in the domain of $f$. We will construct a quasi-interpolant based on this kind of coefficient functionals in Section 8.6.1. By combining derivative functionals based at different points we obtain

$$\lambda f = \sum_{i=0}^{\ell} \sum_{k=0}^{r_i} w_{i,k} f^{(k)}(x_i)$$

where each $r_i$ is a nonnegative integer. A typical functional of this kind is the divided difference of a function when some of the arguments are repeated. Such functionals are fundamental in interpolation with polynomials. Recall that if the same argument occurs $r+1$ times in a divided difference, this signifies that all derivatives of order 0, 1, ..., $r$ are to be interpolated at the point. Note that the point functionals above are derivative functionals with $r_i = 0$ for all $i$.

### 8.4.3 Integral functionals

The final kind of linear functionals that we will consider are based on integration. A typical functional of this kind is

$$\lambda f = \int_a^b f(x)\phi(x)\,dx \tag{8.3}$$

where $\phi$ is some fixed function. Because of basic properties of integration, it is easy to check that this is a linear functional. Just as with point functionals, we can combine several functionals like the one in (8.3) together,

$$\lambda f = w_0 \int_a^b f(x)\phi_0(x)\,dx + w_1 \int_a^b f(x)\phi_1(x)\,dx + \cdots + w_\ell \int_a^b f(x)\phi_\ell(x)\,dx,$$

where $(w_i)_{i=0}^{\ell}$ are real numbers and $\{\phi_i\}_{i=0}^{\ell}$ are suitable functions. Note that the right-hand side of this equation can be written in the form (8.3) if we define $\phi$ by

$$\phi(x) = w_0\phi_0(x) + w_1\phi_1(x) + \cdots + w_\ell\phi_\ell(x).$$

Point functionals can be considered a special case of integral functionals. For if $\phi_\epsilon$ is a function that is positive on the interval $I_\epsilon = (x_i - \epsilon, x_i + \epsilon)$ and $\int_{I_\epsilon} \phi_\epsilon = 1$, then we know from the mean value theorem that

$$\int_{I_\epsilon} f(x)\phi_\epsilon(x)\,dx = f(\xi)$$

for some $\xi$ in $I_\epsilon$, as long as $f$ is a nicely behaved (for example continuous) function. If we let $\epsilon$ tend to 0 we clearly have

$$\lim_{\epsilon \to 0} \int_{I_\epsilon} f(x)\phi_\epsilon(x)\,dx = f(x_i), \tag{8.4}$$

so by letting $\phi$ in (8.3) be a nonnegative function with small support around $x$ and unit integral we can come as close to point interpolation as we wish.

If we include the condition that $\int_a^b \phi\,dx = 1$, then the natural interpretation of (8.3) is that $\lambda f$ gives a weighted average of the function $f$, with $\phi(x)$ giving the weight of the

function value $f(x)$. A special case of this is when $\phi$ is the constant $\phi(x) = 1/(b-a)$; then $\lambda f$ is the traditional average of $f$. From this point of view the limit (8.4) is quite obvious: if we take the average of $f$ over ever smaller intervals around $x_i$, the limit must be $f(x_i)$.

The functional $\int_a^b f(x)\,dx$ is often referred to as the *first moment* of $f$. As the name suggests there are more moments. The $i+1$st moment of $f$ is given by

$$\int_a^b f(x)x^i\,dx.$$

Moments of a function occur in many applications of mathematics like physics and the theory of probability.

### 8.4.4 Preservation of moments and interpolation of linear functionals

Interpolation of function values is a popular approximation method, and we have used it repeatedly in this book. However, is it a good way to approximate a given function $f$? Is it not a bit haphazard to pick out a few, rather arbitrary, points on the graph of $f$ and insist that our approximation should reproduce these points exactly and then ignore all other information about $f$? As an example of what can happen, suppose that we are given a set of function values $\big(x_i, f(x_i)\big)_{i=1}^m$ and that we use piecewise linear interpolation to approximate the underlying function. If $f$ has been sampled densely and we interpolate all the values, we would expect the approximation to be good, but consider what happens if we interpolate only two of the values. In this case we cannot expect the resulting straight line to be a good approximation. If we are only allowed to reproduce two pieces of information about $f$ we would generally do much better by reproducing its first two moments, i.e., the two integrals $\int f(x)\,dx$ and $\int f(x)x\,dx$, since this would ensure that the approximation would reproduce some of the *average* behaviour of $f$.

Reproduction of moments is quite easy to accomplish. If our approximation is $g$, we just have to ensure that the conditions

$$\int_a^b g(x)x^i\,dx = \int_a^b f(x)x^i\,dx, \quad i = 0,\, 1,\, \ldots,\, n-1$$

are enforced if we want to reproduce $n$ moments. In fact, this can be viewed as a generalisation of interpolation if we view interpolation to be preservation of the values of a set of linear functionals $(\rho_i)_{i=1}^n$,

$$\rho_i g = \rho_i f, \quad \text{for } i = 1,\, 2,\, \ldots,\, n. \tag{8.5}$$

When $\rho_i f = \int_a^b f(x)x^{i-1}\,dx$ for $i = 1, \ldots, n$ we preserve moments, while if $\rho_i f = f(x_i)$ for $i = 1, \ldots, n$ we preserve function values. Suppose for example that $g$ is required to lie in the linear space spanned by the basis $\{\psi_j\}_{j=1}^n$. Then we can determine coefficients $(c_j)_{j=1}^n$ so that $g(x) = \sum_{j=1}^n c_j\psi_j(x)$ satisfies the interpolation conditions (8.5) by inserting this expression for $g$ into (8.5). By exploiting the linearity of the functionals, we end up with the $n$ linear equations

$$c_1\rho_i(\psi_1) + c_2\rho_i(\psi_2) + \cdots + c_n\rho_i(\psi_n) = \rho_i(f), \quad i = 1, \ldots, n$$

in the $n$ unknown coefficients $(c_i)_{i=1}^n$. In matrix-vector form this becomes

$$\begin{pmatrix} \rho_1(\psi_1) & \rho_1(\psi_2) & \cdots & \rho_1(\psi_n) \\ \rho_2(\psi_1) & \rho_2(\psi_2) & \cdots & \rho_1(\psi_n) \\ \vdots & \vdots & \ddots & \vdots \\ \rho_n(\psi_1) & \rho_n(\psi_2) & \cdots & \rho_n(\psi_n) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} \rho_1(f) \\ \rho_2(f) \\ \vdots \\ \rho_n(f) \end{pmatrix}. \tag{8.6}$$

A fundamental property of interpolation by point functionals is that the only polynomial of degree $d$ that interpolates the value 0 at $d+1$ points is the zero polynomial. This corresponds to the fact that when $\rho_i f = f(x_i)$ and $\psi_i(x) = x^i$ for $i = 0, \ldots, d$, the matrix in (8.6) is nonsingular. Similarly, it turns out that the only polynomial of degree $d$ whose $d+1$ first moments vanish is the zero polynomial, which corresponds to the fact that the matrix in (8.6) is nonsingular when $\rho_i f = \int_a^b f(x)x^i\,dx$ and $\psi_i(x) = x^i$ for $i = 0, \ldots, d$.

If the equations (8.6) can be solved, each coefficient will be a linear combination of the entries on the right-hand side,

$$c_j = \lambda_j f = w_{j,1}\rho_1(f) + w_{j,2}\rho_2(f) + \cdots + w_{j,n}\rho_n(f).$$

We recognise this as (8.2) when the $\rho_i$ correspond to point functionals, whereas we have

$$\begin{aligned} c_j = \lambda_j f &= w_{j,1}\int_a^b f(x)\,dx + w_{j,2}\int_a^b f(x)x\,dx + \cdots + w_{j,n}\int_a^b f(x)x^{n-1}\,dx \\ &= \int_a^b f(x)\big(w_{j,1} + w_{j,2}x + \cdots + w_{j,n}x^{n-1}\big)\,dx \end{aligned}$$

when the $\rho_i$ correspond to preservation of moments.

### 8.4.5 Least squares approximation

In the discussion of point functionals, we mentioned that least squares approximation leads to coefficients that are linear combinations of point functionals when the error is measured by summing up the squares of the errors at a given set of data points. This is naturally termed *discrete* least squares approximation. In *continuous* least squares approximation we minimise the integral of the square of the error. If the function to be approximated is $f$ and the approximation $g$ is required to lie in a linear space $\mathbb{S}$, we solve the minimisation problem

$$\min_{g\in\mathbb{S}} \int_a^b \big(f(x) - g(x)\big)^2\,dx.$$

If $\mathbb{S}$ is spanned by $(\psi_i)_{i=1}^n$, we can write $g$ as $g = \sum_{i=1}^n c_i\psi$ and the minimisation problem becomes

$$\min_{(c_1,\ldots,c_n)\in\mathbb{R}^n} \int_a^b \Big(f(x) - \sum_{i=1}^n c_i\psi(x)\Big)^2\,dx.$$

To determine the minimum we differentiate with respect to each coefficient and set the derivatives to zero which leads to the so-called *normal equations*

$$\sum_{i=1}^n c_i \int_a^b \psi_i(x)\psi_j(x)\,dx = \int_a^b \psi_j(x)f(x)\,dx, \quad \text{for } j = 1, \ldots, n.$$

If we use the notation above and introduce the linear functionals $\rho_i f = \int_a^b \psi_i(x) f(x)$ represented by the basis functions, we recognise this linear system as an instance of (8.6). In other words, least squares approximation is nothing but interpolation of the linear functionals represented by the basis functions. In particular, preservation of moments corresponds to least squares approximation by polynomials.

### 8.4.6 Computation of integral functionals

In our discussions involving integral functionals we have tacitly assumed that the values of integrals like $\int_a^b f(x)\psi(x)\,dx$ are readily available. This is certainly true if both $f$ and $\psi$ are polynomials, and it turns out that it is also true if both $f$ and $\psi$ are splines. However, if $f$ is some general function, then the integral cannot usually be determined exactly, even when $\psi_i$ is a polynomial. In such situations we have to resort to numerical integration methods. Numerical integration amounts to computing an approximation to an integral by evaluating the function to be integrated at certain points, multiplying the function values by suitable weights, and then adding up to obtain the approximate value of the integral,

$$\int_a^b f(x)\,dx \approx w_0 f(x_0) + w_1 f(x_1) + \cdots + w_\ell f(x_\ell).$$

In other words, when it comes to practical implementation of integral functionals we have to resort to point functionals. In spite of this, integral functionals and continuous least squares approximation are such important concepts that it is well worth while to have an exact mathematical description. And it is important to remember that we do have exact formulas for the integrals of polynomials and splines.

## 8.5 Alternative ways to construct coefficient functionals

In Section 8.2 we constructed three quasi-interpolants by following the general procedure in Section 8.1. In this section we will deduce two alternative ways to construct quasi-interpolants.

### 8.5.1 Computation via evaluation of linear functionals

Let us use the 3-point, quadratic quasi-interpolant in subsection 8.2.2 as an example. In this case we used $I = [t_{j+1}, t_{j+2}]$ as the local interval for determining $c_j = \lambda_j f$. This meant that the local spline space $\mathbb{S}_{2,\boldsymbol{t},I}$ become the space of quadratic polynomials on $I$ which has dimension three. This space is spanned by the three B-splines $\{B_i\}_{i=j-1}^{j+1}$ and interpolation at the three points

$$t_{j+1}, \quad t_{j+3/2} = \frac{t_{j+1} + t_{j+2}}{2}, \quad t_{j+2}$$

allowed us to determine a local interpolant $g^I = \sum_{i=j-1}^{j+1} b_i B_i$ whose middle coefficient $b_j$ we used as $\lambda_j f$.

An alternative way to do this is as follows. Since $g^I$ is constructed by interpolation at the three points $t_{j+1}$, $t_{j+3/2}$ and $t_{j+2}$, we know that $\lambda_j f$ can be written in the form

$$\lambda_j f = w_1 f(t_{j+1}) + w_2 f(t_{j+3/2}) + w_3 f(t_{j+2}). \tag{8.7}$$

We want to reproduce the local spline space which in this case is just the space of quadratic polynomials. This means that (8.7) should be valid for all quadratic polynomials. Reproduction of quadratic polynomials can be accomplished by demanding that (8.7) should be exact when $f$ is replaced by the three elements of a basis for $\mathbb{S}_{2,\boldsymbol{t},I}$. The natural basis to use in our situation is the B-spline basis $\{B_i\}_{i=j-1}^{j+1}$. Inserting this, we obtain the system

$$\begin{aligned}\lambda_j B_{j-1} &= w_1 B_{j-1}(t_{j+1}) + w_2 B_{j-1}(t_{j+3/2}) + w_3 B_{j-1}(t_{j+2}),\\ \lambda_j B_j &= w_1 B_j(t_{j+1}) + w_2 B_j(t_{j+3/2}) + w_3 B_j(t_{j+2}),\\ \lambda_j B_{j+1} &= w_1 B_{j+1}(t_{j+1}) + w_2 B_{j+1}(t_{j+3/2}) + w_3 B_{j+1}(t_{j+2}).\end{aligned}$$

in the three unknowns $w_1$, $w_2$ and $w_3$. The left-hand sides of these equations are easy to determine. Since $\lambda_j f$ denotes the $j$th B-spline coefficient, it is clear that $\lambda_j B_i = \delta_{i,j}$, i.e., it is 1 when $i = j$ and 0 otherwise.

To determine the right-hand sides we have to compute the values of the B-splines. For this it is useful to note that the $w_j$'s in equation (8.7) cannot involve any of the knots other than $t_{j+1}$ and $t_{j+2}$ since a general polynomial knows nothing about these knots. This means that we can choose the other knots so as to make life simple for ourselves. The easiest option is to choose the first three knots equal to $t_{j+1}$ and the last three equal to $t_{j+2}$. But then we are in the Bézier setting, and we know that the B-splines in this case will have the same values if we choose $t_{j+1} = 0$ and $t_{j+2} = 1$. The knots are then $(0, 0, 0, 1, 1, 1)$ which means that $t_{j+3/2} = 1/2$. If we denote the B-splines on these knots by $\{\tilde{B}_i\}_{i=1}^3$, we can replace $B_i$ in (8.5.1) by $\tilde{B}_{i-j+2}$ for $i = 1$, 2, 3. We can now simplify (8.5.1) to

$$\begin{aligned}0 &= w_1 \tilde{B}_1(0) + w_2 \tilde{B}_1(1/2) + w_3 \tilde{B}_1(1),\\ 1 &= w_1 \tilde{B}_2(0) + w_2 \tilde{B}_2(1/2) + w_3 \tilde{B}_2(1),\\ 0 &= w_1 \tilde{B}_3(0) + w_2 \tilde{B}_3(1/2) + w_3 \tilde{B}_3(1).\end{aligned}$$

If we insert the values of the B-splines we end up with the system

$$\begin{aligned}w_1 + w_2/4 &= 0,\\ w_2/2 &= 1,\\ w_2/4 + w_3 &= 0,\end{aligned}$$

which has the solution $w_1 = -1/2$, $w_2 = 2$ and $w_3 = -1/2$. In conclusion we have

$$\lambda_j f = \frac{-f(t_{j+1}) + 4f(t_{j+3/2}) - f(t_{j+2})}{2},$$

as we found in Section 8.2.2.

This approach to determining the linear functional works quite generally and is often the easiest way to compute the weights $(w_i)$.

### 8.5.2 Computation via explicit representation of the local approximation

There is a third way to determine the expression for $\lambda_j f$. For this we write down an explicit expression for the approximation $g^I$. Using the 3-point quadratic quasi-interpolant as our

example again, we introduce the abbreviations $a = t_{j+1}$, $b = t_{j+3/2}$ and $c = t_{j+2}$. We can write the local interpolant $g^I$ as

$$g^I(x) = \frac{(x-b)(x-c)}{(a-b)(a-c)}f(a) + \frac{(x-a)(x-c)}{(b-a)(b-c)}f(b) + \frac{(x-a)(x-b)}{(c-a)(c-b)}f(c),$$

as it is easily verified that $g^I$ then satisfies the three interpolation conditions $g^I(a) = f(a)$, $g^I(b) = f(b)$ and $g^I(c) = f(c)$. What remains is to write this in terms of the B-spline basis $\{B_i\}_{i=j-1}^{j+1}$ and pick out coefficient number $j$. Recall that we have the notation $\gamma_j(f)$ for the $j$th B-spline coefficient of a spline $f$. Coefficient number $j$ on the left-hand side is $\lambda_j f$. On the right, we find the B-spline coefficients of each of the three polynomials and add up. The numerator of the first polynomial is $(x-b)(x-c) = x^2 - (b+c)x + bc$. To find the $j$th B-spline of this polynomial, we make use of Corollary 3.5 which tells that, when $d = 2$, we have $\gamma_j(x^2) = t_{j+1}t_{j+2} = ac$ and $\gamma_j(x) = (t_{j+1} + t_{j+2})/2 = (a+c)/2 = b$, respectively. The $j$th B-spline coefficient of the first polynomial is therefore

$$\gamma_j\Big(\frac{ac - (b+c)b + bc}{(a-b)(a-c)}\Big) = \frac{ac - b^2}{(a-b)(a-c)} \tag{8.8}$$

which simplifies to $-1/2$ since $b = (a+c)/2$. Similarly, we find that the $j$th B-spline coefficient of the second and third polynomials are 2 and $-1/2$, respectively. The complete $j$th B-spline coefficient of the right-hand side of (8.8) is therefore $-f(a)/2 + 2f(b) - f(c)/2$. In total, we have therefore obtained

$$\lambda_j f = \gamma_j(g^I) = -\frac{f(t_{j+1})}{2} + 2f(t_{j+3/2}) - \frac{f(t_{j+2})}{2},$$

as required.

This general procedure also works generally, and we will see another example of it in Section 8.6.1.

## 8.6 Two quasi-interpolants based on point functionals

In this section we consider two particular quasi-interpolants that can be constructed for any polynomial degree. They may be useful for practical approximation problems, but we are going to use them to prove special properties of spline functions in Chapters 9 and 10. Both quasi-interpolants are based on point functionals: In the first case all the points are identical which leads to derivative functionals, in the second case all the points are distinct.

### 8.6.1 A quasi-interpolant based on the Taylor polynomial

A very simple local, polynomial approximation is the Taylor polynomial. This leads to a quasi-interpolant based on derivative functionals. Even though we use splines of degree $d$, our local approximation can be of lower degree; in Theorem 8.5 this degree is given by $r$.

**Theorem 8.5** (de Boor-Fix)**.** *Let $r$ be an integer with $0 \le r \le d$ and let $x_j$ be a number in $[t_j, t_{j+d+1}]$ for $j = 1, \ldots, n$. Consider the quasi-interpolant*

$$Q_{d,r}f = \sum_{j=1}^{n} \lambda_j(f) B_{j,d}, \quad \text{where} \quad \lambda_j(f) = \frac{1}{d!}\sum_{k=0}^{r} (-1)^k D^{d-k}\rho_{j,d}(x_j) D^k f(x_j), \tag{8.9}$$

*and $\rho_{j,d}(y) = (y - t_{j+1}) \cdots (y - t_{j+d})$. Then $Q_{d,r}$ reproduces all polynomials of degree $r$ and $Q_{d,d}$ reproduces all splines in $\mathbb{S}_{d,\boldsymbol{t}}$.*

**Proof.** To construct $Q_{d,r}$ we let $I$ be the knot interval that contains $x_j$ and let the local approximation $g^I = P_r^I f$ be the Taylor polynomial of degree $r$ at the point $x_j$,

$$g^I(x) = P_r^I f(x) = \sum_{k=0}^{r} \frac{(x-x_j)^k}{k!} D^k f(x_j).$$

To construct the linear functional $\lambda_j f$, we have to find the B-spline coefficients of this polynomial. We use the same approach as in Section 8.5.2. For this Marsden's identity,

$$(y-x)^d = \sum_{j=1}^{n} \rho_{j,d}(y) B_{j,d}(x),$$

will be useful. Setting $y = x_j$, we see that the $j$th B-spline coefficient of $(x_j - x)^d$ is $\rho_{j,d}(x_j)$. Differentiating Marsden's identity $d-k$ times with respect to $y$, setting $y = x_i$ and rearranging, we obtain the $j$th B-spline coefficient of $(x-x_j)^k/k!$ as

$$\gamma_j\big((x-x_j)^k/k!\big) = (-1)^k D^{d-k} \rho_{j,d}(x_j)/d! \qquad \text{for } k = 0, \ldots, r.$$

Summing up, we find that

$$\lambda_j(f) = \frac{1}{d!} \sum_{k=0}^{r} (-1)^k D^{d-k} \rho_{j,d}(x_j) D^k f(x_j).$$

Since the Taylor polynomial of degree $r$ reproduces polynomials of degree $r$, we know that the quasi-interpolant will do the same. If $r = d$, we reproduce polynomials of degree $d$ which agree with the local spline space $\mathbb{S}_{d,\boldsymbol{t},I}$ since $I$ is a single knot interval. The quasi-interpolant therefore reproduces the whole spline space $\mathbb{S}_{d,\boldsymbol{t}}$ in this case. ∎

**Example 8.6.** We find

$$D^d \rho_{j,d}(y)/d! = 1, \quad D^{d-1}\rho_{j,d}(y)/d! = y - t_j^*, \quad \text{where} \quad t_j^* = \frac{t_{j+1} + \cdots + t_{j+d}}{d}. \tag{8.10}$$

For $r = 1$ and $x_j = t_j^*$ we therefore obtain

$$Q_{d,r} f = \sum_{j=1}^{n} f(t_j^*) B_{j,d}$$

which is the Variation Diminishing spline approximation. For $d = r = 2$ we obtain

$$Q_{2,2} f = \sum_{j=1}^{n} \big[ f(x_j) - (x_j - t_{j+3/2}) D f(x_j) + \frac{1}{2}(x_j - t_{j+1})(x_j - t_{j+2}) D^2 f(x_j) \big] B_{j,2}. \tag{8.11}$$

while for $d = r = 3$ and $x_j = t_{j+2}$ we obtain

$$Q_{3,3} f = \sum_{j=1}^{n} \big[ f(t_{j+2}) + \frac{1}{3}(t_{j+3} - 2t_{j+2} + t_{j+1}) D f(t_{j+2}) - \frac{1}{6}(t_{j+3} - t_{j+2})(t_{j+2} - t_{j+1}) D^2 f(t_{j+2}) \big] B_{j,3}. \tag{8.12}$$

We leave the detailed derivation as a problem for the reader.

Since $Q_{d,d} f = f$ for all $f \in \mathbb{S}_{d,\boldsymbol{t}}$ it follows that the coefficients of a spline $f = \sum_{j=1}^{n} c_j B_{j,d}$ can be written in the form

$$c_j = \frac{1}{d!} \sum_{k=0}^{d} (-1)^k D^{d-k} \rho_{j,d}(x_j) D^k f(x_j), \qquad \text{for } j = 1, \ldots, n, \tag{8.13}$$

where $x_j$ is any number in $[t_j, t_{j+d+1}]$.

### 8.6.2 Quasi-interpolants based on evaluation

Another natural class of linear functionals is the one where each $\lambda_j$ used to define $Q$ is constructed by evaluating the data at $r+1$ distinct points

$$t_j \le x_{j,0} < x_{j,1} < \cdots < x_{j,r} \le t_{j+d+1} \tag{8.14}$$

located in the support $[t_j, t_{j+d+1}]$ of the B-spline $B_{j,d}$ for $j = 1, \ldots, n$. We consider the quasi-interpolant

$$P_{d,r}f = \sum_{j=1}^{n} \lambda_{j,r}(f) B_{j,d}, \tag{8.15}$$

where

$$\lambda_{j,r}(f) = \sum_{k=0}^{r} w_{j,k} f(x_{j,k}). \tag{8.16}$$

From the preceding theory we know how to choose the constants $w_{j,k}$ so that $P_{d,r}f = f$ for all $f \in \pi_r$.

**Theorem 8.7.** *Let $\mathbb{S}_{d,\boldsymbol{t}}$ be a spline space with a $d+1$-regular knot vector $\boldsymbol{t} = (t_i)_{i=1}^{n+d+1}$. Let $(x_{j,k})_{k=0}^{r}$ be $r+1$ distinct points in $[t_j, t_{j+d+1}]$ for $j = 1, \ldots, n$, and let $w_{j,k}$ be the $j$th B-spline coefficient of the polynomial*

$$p_{j,k}(x) = \prod_{\substack{s=0 \\ s\neq k}}^{r} \frac{x - x_{j,s}}{x_{j,k} - x_{j,s}}.$$

*Then $P_{d,r}f = f$ for all $f \in \pi_r$, and if $r = d$ and all the numbers $(x_{j,k})_{k=0}^{r}$ lie in one subinterval*

$$t_j \le t_{\ell_j} \le x_{j,0} < x_{j,1} < \cdots < x_{j,r} \le t_{\ell_j+1} \le t_{j+d+1} \tag{8.17}$$

*then $P_{d,d}f = f$ for all $f \in \mathbb{S}_{d,\boldsymbol{t}}$.*

**Proof.** It is not hard to see that

$$p_{j,k}(x_{j,i}) = \delta_{k,i}, \quad k, i = 0, \ldots, r$$

so that the polynomial

$$P_{d,r}^{I} f(x) = \sum_{k=0}^{r} p_{j,k}(x) f(x_{j,k})$$

satisfies the interpolation conditions $P_{d,r}^{I} f(x_{j,r}) = f(x_{j,r})$ for all $j$ and $r$. The result therefore follows from the general recipe. ∎

In order to give examples of quasi-interpolants based on evaluation we need to know the B-spline coefficients of the polynomials $p_{j,k}$. We will return to this in more detail in Chapter 9, see (9.15) in the case $r = d$. A similar formula can be given for $r < d$.

**Example 8.8.** For $r = 1$ we have

$$p_{j,0}(x) = \frac{x_{j,1} - x}{x_{j,1} - x_{j,0}}, \quad p_{j,1}(x) = \frac{x - x_{j,0}}{x_{j,1} - x_{j,0}}$$

and (8.15) takes the form

$$P_{d,1}f = \sum_{j=1}^{n} \left[ \frac{x_{j,1} - t_j^*}{x_{j,1} - x_{j,0}} f(x_{j,0}) + \frac{t_j^* - x_{j,0}}{x_{j,1} - x_{j,0}} f(x_{j,1}) \right] B_{j,d}. \tag{8.18}$$

This quasi-interpolant reproduces straight lines for any choice of $t_j \le x_{j,0} < x_{j,1} \le t_{j+d+1}$. If we choose $x_{j,0} = t_j^*$ the method simplifies to

$$\tilde{P}_{d,1}f = \sum_{j=1}^{n} f(t_j^*) B_{j,d}. \tag{8.19}$$

This is again the *Variation diminishing method of Schoenberg.*

## Exercises for Chapter 8

8.1 In this exercise we assume that the points $(x_{i,k})$ and the spline space $\mathbb{S}_{d,\boldsymbol{t}}$ are as in Theorem 8.7.

a) Show that for $r = d = 2$

$$\begin{aligned} P_{2,2}f = \sum_{j=1}^{n} \Bigg[ & \frac{(t_{j+1} - x_{j,1})(t_{j+2} - x_{j,2}) + (t_{j+2} - x_{j,1})(t_{j+1} - x_{j,2})}{2(x_{j,0} - x_{j,1})(x_{j,0} - x_{j,2})} f(x_{j,0}) \\ & + \frac{(t_{j+1} - x_{j,0})(t_{j+2} - x_{j,2}) + (t_{j+2} - x_{j,0})(t_{j+1} - x_{j,2})}{2(x_{j,1} - x_{j,0})(x_{j,1} - x_{j,2})} f(x_{j,1}) \\ & + \frac{(t_{j+1} - x_{j,0})(t_{j+2} - x_{j,1}) + (t_{j+2} - x_{j,0})(t_{j+1} - x_{j,1})}{2(x_{j,2} - x_{j,0})(x_{j,2} - x_{j,1})} f(x_{j,2}) \Bigg] B_{j,2} \end{aligned} \tag{8.20}$$

b) Show that (8.20) reduces to the operator (9.6) for a suitable choice of $(x_{j,k})_{k=0}^{2}$.

8.2 Derive the following operators $Q_{d,l}$ and show that they are exact for $\pi_r$ for the indicated $r$. Again we the points $(x_{j,k})$ and the spline space $\mathbb{S}_{d,\boldsymbol{t}}$ are is in Theorem 8.7. Which of the operators reproduce the whole spline space?

a) $Q_{d,0}f = \sum_{j=1}^{n} f(x_j) B_{j,d}, \quad (r = 0).$

b) $Q_{d,1}f = \sum_{j=1}^{n} \big[ f(x_j) + (t_j - x_j) Df(x_j) \big] B_{j,d}, \quad (r = 1).$

c) $\tilde{Q}_{d,1}f = \sum_{j=1}^{n} f(t_j^*) B_{j,d}, \quad (r = 1).$

d)

$$\begin{aligned} Q_{2,2}f = \sum_{j=1}^{n} \big[ & f(x_j) - (x_j - t_{j+3/2}) Df(x_j) \\ & + \frac{1}{2}(x_j - t_{j+1})(x_j - t_{j+2}) D^2 f(x_j) \big] B_{j,2}, \text{(r=2)}. \end{aligned}$$

e) $\tilde{Q}_{2,2}f = \sum_{j=1}^{n} \big[ f(t_{j+3/2}) - \frac{1}{2}(t_{j+2} - t_{j+1})^2 D^2 f(t_{j+3/2}) \big] B_{j,2}, \quad (r = 2).$

f)

$$Q_{3,3}f = \sum_{j=1}^{n} \big[ f(t_{j+2}) + \frac{1}{3}(t_{j+3} - 2t_{j+2} + t_{j+1}) Df(t_{j+2}) \\ - \frac{1}{6}(t_{j+3} - t_{j+2})(t_{j+2} - t_{j+1}) D^2 f(t_{j+2}) \big] B_{j,3}, \quad (r = 3).$$