Splines in Industrial Applications

Lecture Notes, Zaragoza, February 2007

Tomas Sauer
Lehrstuhl für Numerische Mathematik
Justus–Liebig–Universität Gießen
Heinrich–Buff–Ring 44
D-35392 Gießen

Version 0.0
Version 28.2.2007

Avoiding empty pages . . . | 0

Links were electronic now, not narrative . . . Until the advent of hyperlinks, only God had been able to see simultaneously into past, present and future alike; human beings were imprisoned in the calendar of their days.

S. Rushdie, *Fury*

"Money don't buy happiness [. . .]" – "I only wanted to rent it for a few weeks"

T. Pratchett, *Maskerade*

Nature is not embarrassed by difficulties of analysis.

A. Fresnel

To isolate mathematics from the practical demands of the sciences is to invite the sterility of a cow shut away from the bulls.

P. Chebyshev

Contents | 0

1 Basics **2**
1.1 Knots and B–Splines 2
1.2 The de Boor algorithm 5
1.3 Curry, Schoenberg and other bases 7
1.4 How to represent a spline? 9

2 Manipulating Splines **10**
2.1 Simple geometric transformations 10
2.2 Differentiation and integration 10
2.3 Numerical integration of spline functions 13

3 Interpolation and minimality **15**
3.1 Interpolation at knots and the natural spline 15
3.2 Minimality of the natural spline 16
3.3 Schoenberg, Whitney, Greville 19
3.4 Parametric interpolation 20

4 What is wrong with interpolation? **24**
4.1 Dependency on the parameterization 24
4.2 Overshooting – the curse of interpolation 25
4.3 Handling corners 27
4.4 Why approximation can be better 28

5 Smoothing splines **32**
5.1 Interpolation and minimization 32
5.2 Minimizing the sup–norm and linear programming 34
5.3 Least squares and energy functionals 36
5.4 Efficient implementation of Gramians 38
5.5 An example of smoothing splines 39
5.6 A sup–smoothing spline 40

6 Knot insertion and removal **44**
6.1 Refinement of knot sequences 44
6.2 Knot insertion 45
6.3 First applications of knot insertion 48
6.4 Zeros of spline functions 50
6.5 Knot removal 52

I disapprove of certainties [. . .] They limit one's range of vision. Doubt is one aspect of width.

S. Rushdie, Grimus

1 Basics

Splines – at least in our context here – are piecewise polynomials used for the description and manipulation of curves. These curves are usually **not** graphs of a function but *parametric* ones, i.e., $\boldsymbol{f} : [a, b] \to \mathbb{R}^d$, where the parameter interval $[a, b]$ is usually irrelevant[1] and the dimension d is 2 or 3. In order to store an manipulate such a curve efficiently on the computer, we require two fundamental properties:

1. The curve must be represented by a finite number of *coefficients* $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n \in \mathbb{R}^d$.
2. The coefficients must reflect geometric properties of the curve.

There are various methods to construct such curves and the discussion of the advantages and disadvantages of such representations are a topic all by itself. But since we have to make a choice we simply choose splines here and will not regret it.

1.1 Knots and B–Splines

The basic concept underlying splines is that of a *knot sequence* as it defines the regions restricted to which splines are polynomials.

Definition 1.1 *A finite, indexed set $T = T_{m,n} = \{t_1, \ldots, t_{m+n+1}\}$ is called a **knot sequence** of order m if*

1. $t_1 \leq \cdots \leq t_{n+m+1}$.
2. $t_j < t_{j+m+1}$.

Obviously, the order of the knot sequence is only relevant for the second condition in the above definition which says that the **multiplicity** of any knot must not exceed $m + 1$; the multiplicity

[1] We will, however, see examples where this matters, namely, when we want the curve to be arc–length parameterized, at least approximately.

$\mu = \mu_j$ of the knot t_j is simply the number that says how often this not is repeated within T:

$$t_1 \leq \cdots \leq t_{j-1} < \underbrace{t_j = \cdots = t_{j+\mu-1}}_{\mu} < t_{j+\mu} \leq \cdots \leq t_{m+n+1}. \tag{1.1}$$

If $\mu_j = 1$, for $j = 1, \ldots, m+n+1$, then we say that the knot sequence consists of **simple knots**.

Definition 1.2 *For $k = 0, \ldots, m$, the **B–splines** N_j^k, $j = 1, \ldots, n+m-k$, of order k are defined as*

$$N_j^0(\cdot \,|\, T) = \chi_{[t_j, t_{j+1})}, \tag{1.2}$$

$$N_j^k(\cdot \,|\, T) = \frac{\cdot - t_j}{t_{j+k} - t_j} N_j^{k-1}(\cdot \,|\, T) + \frac{t_{j+k+1} - \cdot}{t_{j+k+1} - t_{j+1}} N_{j+1}^{k-1}(\cdot \,|\, T). \tag{1.3}$$

Almost directly from this definition, we can conclude some fundamental properties of the B–Splines.

The B–Splines are

1. nonnegative, $N_j^k \geq 0$,
2. compactly supported, $N_j^k(x \,|\, T) = 0$, $x \notin [t_j, t_{j+k+1}]$,
3. piecewise polynomials of degree at most k, $N_j^k\big|_{(t_\ell, t_{\ell+1})} \in \Pi_k$.

It is very easy to prove these properties simultaneously by induction on k: first, they are immediately verified for $k = 0$ and then the inductive step consists of noting that the recurrence (1.3) implies that

$$N_j^k(x \,|\, T) = 0, \qquad x \notin ([t_j, t_{j+k}] \cup [t_{j+1}, t_{j+k+1}]) = [t_j, t_{j+k+1}]$$

and that for $x \in [t_{j+1}, t_{j+k}]$ all the terms in (1.3) are nonnegative while for $x \in [t_j, t_{j+1}] \cup [t_{j+k}, t_{j+k+1}]$ the negative linear term is rendered irrelevant by a zero spline function.

Remark 1.3 *While for simple knots the recursion of Definition 1.2 works smoothly and flawless, it has to be implemented with a little bit of care in the case of multiple knots! At some point of the iteration, at least in the computation of N_j^1 for some j, we will encounter the case of a "division by zero". But in fact, we will not! The associated support interval of the respective is spline is the empty set and this tells us that we need not take this spline into account in the recurrence.*

The evaluation of a B–Spline at a vector `X` of points will be done by the `Octave`–function[2] `BSplEval`, so let us use this function to make ourselves acquainted to the system and plot

[2]The functions are designed in such a way that they should (!) also run under `Matlab`, but since `Octave` is Open Source software and thus freely and generally available, we will continue to refer to `Octave` exclusively.

some B–splines of order m. A look at Definition 1.2 convinces us that the associated B–splines are $N_1^m, \ldots, N_n^m$, so in order to have n of them, our knot vector has to have $n + m + 1$ entries. OK, let's try the famous *cubic* case and let us just use equidistant knots, say

```
octave> T = (0:10)
T =

   0   1   2   3   4   5   6   7   8   9  10
```

So we have $n = 11 - 4 = 7$ B–splines which we will evaluate at a fine grid and plot them:

```
octave> hold on; X = (0:.01:10);
octave> for j=1:7 plot( X,BSplEval( j,3,T,X ) ); end
```

That was easy. So let us get curious and try a non–equidistant spacing:

```
octave> T = sqrt(0:10); clearplot; X = (0:.01:max(T));
octave> for j=1:7 plot( X,BSplEval( j,3,T,X ) ); end
```

The results can be seen in Fig. 1.1, but they can only be a motivation to perform "independent" experiments with B–splines relative to varying knot distributions. But let us stop playing and

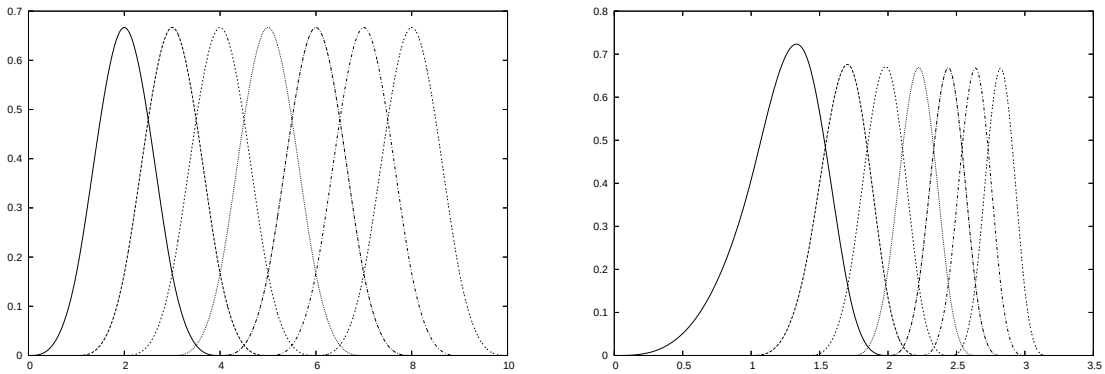

Figure 1.1: The B–splines with equidistant *(left)* and square root *(right)* knots.

return to theory.

Definition 1.4 *A **spline curve** in $\mathbb{R}^d$ is a curve of the form*

$$S_m \boldsymbol{d} = S_m\left(\cdot \,|\, T\right) \boldsymbol{d} := \sum_{j=1}^{n} \boldsymbol{d}_j \, N_j^m\left(\cdot \,|\, T\right). \tag{1.4}$$

A spline curve is represented by the **control points** $\boldsymbol{d}_j$, $j = 1, \ldots, n$, and the **knot sequence** T, so all manipulations of spline curves have to be performed on either the control points or the knots or both.

1.2 The de Boor algorithm

On our way to a better understanding of splines, we next consider a geometric algorithm for the evaluation of spline curves which is due to de Boor.

Algorithm 1.5 (de Boor)
Input:

- *knot sequence* $T = T_{m,n}$,
- *control points* $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n \in \mathbb{R}^d$,
- $x \in [t_m, \ldots, t_{n+1}]$.

Procedure:

1. *Determine* $\ell \in \{m+1, \ldots, n\}$ *such that*
$$x \in [t_\ell, t_{\ell+1}).$$
2. *Initialize:*
$$\boldsymbol{d}_j^0(x) = \boldsymbol{d}_j, \qquad j = i-m, \ldots, i.$$
3. *For* $k = 1, \ldots, m$
   (a) *For* $j = i-m+k, \ldots, i$ *compute*
$$\alpha_j(x) = \frac{t_{j+m-k+1} - x}{t_{j+m-k+1} - t_j}$$
   and
$$\boldsymbol{d}_j^k(x) = \alpha(x)\, \boldsymbol{d}_{j-1}^{k-1}(x) + (1 - \alpha(x))\, \boldsymbol{d}_j^{k-1}(x). \tag{1.5}$$

Result: $S_m \boldsymbol{d}(x) = \boldsymbol{d}_\ell^m(x)$.

The correctness of the algorithm is proved rather easily: we just read the recurrence relation of the B–Splines "backwards":

$$\begin{aligned}
S_m \boldsymbol{d}(x) &= \sum_{j=1}^{n} \boldsymbol{d}_j\, N_j^m\,(x \,|\, T) \\
&= \sum_{j=1}^{n} \boldsymbol{d}_j \left[\frac{x - t_j}{t_{j+k} - t_j}\, N_j^{m-1}\,(x \,|\, T) + \frac{t_{j+k+1} - x}{t_{j+k+1} - t_{j+1}}\, N_{j+1}^{m-1}\,(x \,|\, T) \right] \\
&= \sum_{j=1}^{n} \boldsymbol{d}_j \frac{x - t_j}{t_{j+k} - t_j}\, N_j^{m-1}\,(x \,|\, T) + \sum_{j=2}^{n+1} \boldsymbol{d}_{j-1}\, \frac{t_{j+k} - x}{t_{j+k} - t_j}\, N_j^{m-1}\,(x \,|\, T) \\
&= \sum_{j=1}^{n+1} \left[\frac{t_{j+k} - x}{t_{j+k} - t_j}\, \boldsymbol{d}_{j-1} + \frac{x - t_j}{t_{j+k} - t_j}\, \boldsymbol{d}_j \right]\, N_j^{m-1}\,(x \,|\, T)\,.
\end{aligned}$$

The rest of the proof consists of carefully monitoring the spline's support regions, thus extracting the proper part from the set of control points.

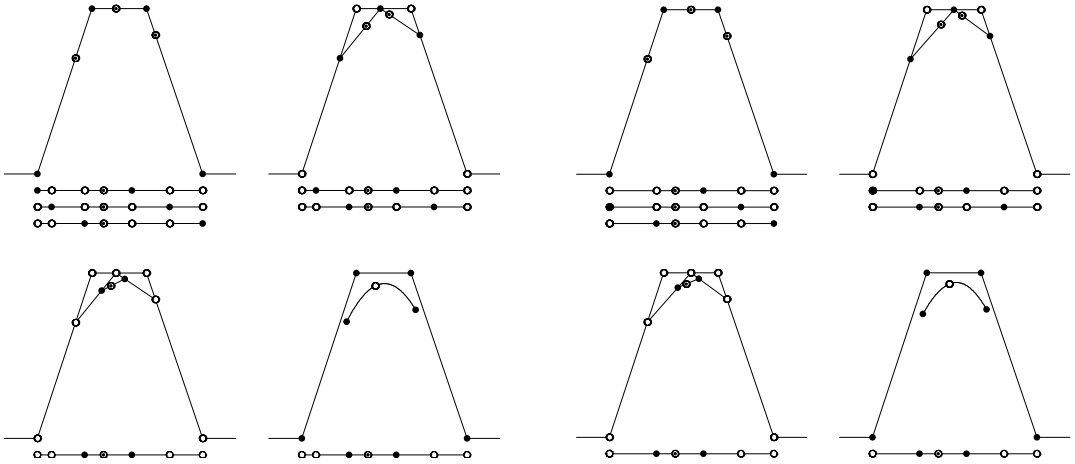

Figure 1.2: The cubic ($m = 3$) de Boor algorithm for a simple *(left)* and a double *(right)* knot.

Remark 1.6 (Further properties)

1. *The spline curve according to the de Boor algorithm is a curve from* $[t_{m+1}, t_{n+1}] \to \mathbb{R}^d$. *Consequently, the* ***boundary knots*** $t_1, \ldots, t_{m+1}$ *and* $t_{n+1}, \ldots, t_{m+n+1}$ *must play a special role.*

2. *This is a little bit strange since the B–Splines are defined everywhere on* $\mathbb{R}$, *not just on* $[t_{m+1}, t_{n+1}]$. *The algorithm, on the other hand, seems to work only on this interval.*

3. *In many applications, the boundary knots are taken as* $m + 1$*–fold knots, that is,*

$$t_1 = \cdots = t_{m+1}, \qquad t_{n+1} = \cdots = t_{n+m+1}.$$

   This guarantees that at the boundary the spline passes through the first and last control points and that its derivatives are given by the differences of boundary control points.

4. *It is usually hard to remember all the index details of the algorithm, but simple to keep in mind the geometric idea: the ratio in which* x *divides certain knot intervals, first generated by* $m + 1$, *then by* m *and finally by* 2 *knots, is used to divide appropriate faces of the* ***control polygon****, which is the piecewise linear function connecting the control points.*

5. *The de Boor algorithm uses* ***convex combinations*** *and therefore the spline curve is always contained in the convex hull of the control polygon – a property that eases the computation of intersections.*

Let us cast some light on the apparently contradictionary situation that the spline curve is defined globally in terms of the B–splines but only locally in terms of the algorithm. To that end,

we consider the simple fact that for the "constant curve" $\boldsymbol{d}_j = 1$, we have that

$$\boldsymbol{d}_j^k(x) = \alpha(x)\, \boldsymbol{d}_{j-1}^k(x) + (1 - \alpha(x))\ \boldsymbol{d}_j^k(x) = \alpha(x) + 1 - \alpha(x) = 1,$$

hence

$$\sum_{j=1}^{n} N_j^k\,(\cdot \,|\, T) = 1. \tag{1.6}$$

However, the above identity only holds on the interval $[t_{m+1}, t_{n+1}]$, see Fig. 1.3, so that the

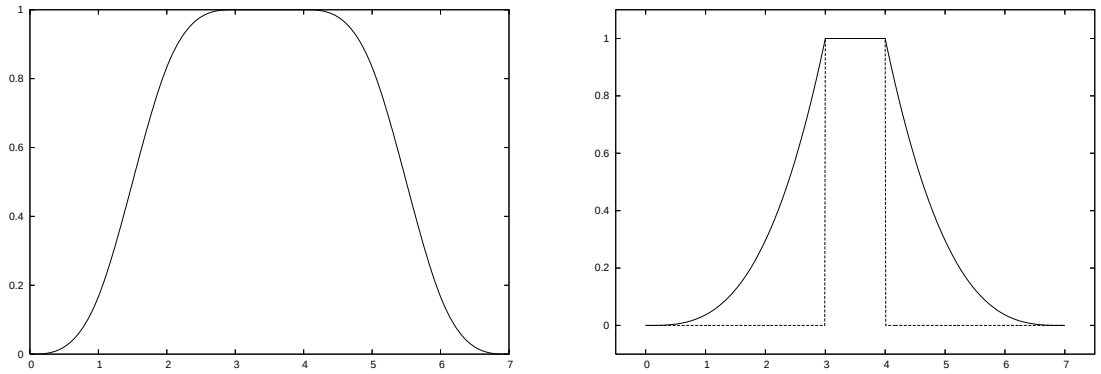

Figure 1.3: Plot of the function from (1.6) for the knot sequence $\{0, 1, 2, 3, 4, 5, 6, 7\}$ *(left)* – the identity is indeed valid only on the interval $[3, 4]$ formed by the innermost boundary knots.

With triple boundary knots there is a non–differentiable "break" at 3 and 4 while the knot sequence $\{3, 3, 3, 3, 4, 4, 4, 4\}$ offers a sharp cut–off *(right)*.

answer to the dilemma is in fact a very simple one:

> Splines are well–defined everywhere, but they are "nice" curves only on the interval $I_m(T) := [t_{m+1}, t_{n+1}]$.

This gives a good argument in favor of $m + 1$–fold boundary knots as in this case

1.3 Curry, Schoenberg and other bases

The most fundamental result of spline theory says that the B–splines are not only special piecewise polynomial functions, but that they form a basis of a most peculiar space.

Definition 1.7 (Spline space) *The **spline space** of order m associated to a knot sequence $T = T_{m,n}$, denoted by $\mathbb{S}_m(T)$ consists of all functions f with the following two properties:*

1. *f is a piecewise polynomial function, $f|_{(t_j, t_{j+1})} \in \Pi_m$.*

2. f has a certain global smoothness:

$$t_{j-1} < t_j = \cdots = t_{j+\mu-1} < t_{j+\mu} \qquad \Rightarrow \qquad f \in C^{m-\mu}\left(t_{j-1}, t_{j+\mu}\right). \tag{1.7}$$

In other words,

> The spline space $\mathbb{S}_m(T)$ consists of all functions that are polynomials when restricted to knot intervals and have continuous $(m-\mu)$th derivatives at a knot of multiplicity μ.

Theorem 1.8 (Curry & Schoenberg) *The B–splines $N_j^m\left(\cdot \,|\, T\right)$ are a **basis** of the spline space $\mathbb{S}_m(T)$.*

One way to prove this result is to use the derivative formula

$$\frac{d}{dx} N_j^m\left(x|T\right) = \frac{m}{t_{j+m} - t_j} N_j^{m-1}\left(x \,|\, T\right) - \frac{m}{t_{j+m+1} - t_{j+1}} N_{j+1}^{m-1}\left(x \,|\, T\right), \qquad j = 1, \ldots, n, \tag{1.8}$$

which can be verified by straightforward though tedious computations: take derivatives of the recurrence (1.3), use (1.8) inductively for the lower order splines and recombine terms appropriately. Since the B–splines are piecewise polynomials, (1.8) holds for any $x \in \mathbb{R} \setminus T$, i.e., away from the knots, and another inductive argument can be used to obtain the continuity properties of the derivatives at knots of some given multiplicity.

But the main consequence of Theorem 1.8 is the following:

> Any function $f \in \mathbb{S}_m(T)$ can be uniquely written as
>
> $$f = \sum_{j=1}^{n} f_j \, N_j^m\left(\cdot \,|\, T\right).$$
>
> In particular, $\dim \mathbb{S}_m(T) = n$.

There is another basis for the $\mathbb{S}_m$, namely the one based on the **truncated powers** which, for simplicity, we will only consider for simple knots. Here, the basis functions are the monomials $1, x, \ldots, x^m$ as well as the functions

$$(x - t_j)_+^m = \begin{cases} (x - t_j)^m, & x \geq t_j, \\ 0, & x \leq t_j, \end{cases} \qquad j = m+1, \ldots, n.$$

These are again n linearly independent piecewise polynomial functions of global differentiability $m-1$, hence a basis for the spline space. Again, they are basically defined on $[t_{m+1}, t_{n+1}]$, but most of this functions have a very large support – quite in contrast to the B–spline.

1.4 How to represent a spline?

Let us conclude the basics on splines by reviewing how a spline curve is represented on the computer. The easiest way is to arrange the control points $\boldsymbol{d}_j$ into a matrix:

$$\boldsymbol{d} = [\boldsymbol{d}_1 \cdots \boldsymbol{d}_n] = \begin{bmatrix} d_{1,1} & \dots & d_{n,1} \\ \vdots & \ddots & \vdots \\ d_{1,d} & \dots & d_{n,d} \end{bmatrix} \in \mathbb{R}^{d\times n}, \qquad \boldsymbol{d}_j = \begin{bmatrix} d_{j,1} \\ \vdots \\ d_{j,d} \end{bmatrix}.$$

To evaluate a spline at a (finite) point set $X \subset \mathbb{R}$, we then compute the matrix

$$\boldsymbol{N}_m(X) = \boldsymbol{N}_m\left(X \,|\, T\right) := \left[N_j^m\left(x \,|\, T\right) \;:\; \begin{matrix} j = 1,\dots,n \\ x \in X \end{matrix} \right],$$

and the value of the spline $S_m\boldsymbol{d}$ at the point set X is

$$S_m\boldsymbol{d}(X) = \boldsymbol{d}\,\boldsymbol{N}_m(X). \tag{1.9}$$

This simple piece of linear algebra will turn out to be very useful in the sequel. The function to compute the matrix $\boldsymbol{N}_m\left(X \,|\, T\right)$, given m, T and X, is `BSplVander` and the apparently strange name will become clear fairly soon.

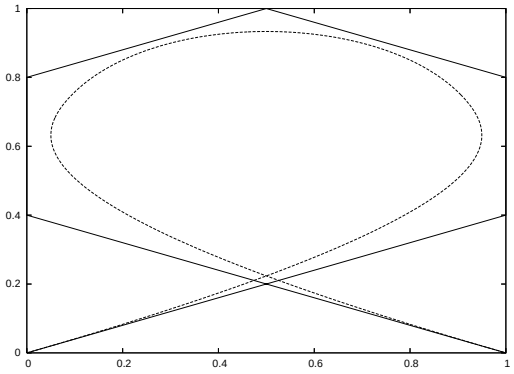

Figure 1.4: A spline curve and its control polygon. It's all quite easy ...

Let us finally show how a spline curve can now be set up and plotted kike in Fig 1.4. First, we define the knot sequence (with four–fold boundary knots) and the control matrix

```
octave> T = [ 0,0,0,0,1,2,3,4,4,4,4 ];
octave> d = [ 0 1 1 .5 0 0 1 ; 0 .4 .8 1 .8 .4 0 ];
```

To evaluate the spline at a fine grid, we simply call

```
octave> X = (0:.01:4); y = d * BSplVander( 3,T,X );
```

and plotting means to plot `d` and `y`:

```
octave> hold on; plot( d(1,:),d(2,:) ); plot ( y(1,:), y(2,:) );
```

And that's it!

Infinities and indivisibles transcend our finite understanding, the former on account of their magnitude, the latter because of their smallness; Imagine what they are when combined.

G. Galilei

2 Manipulating Splines

In this section we review some of the basic procedures needed in elementary computations of splines.

2.1 Simple geometric transformations

If one wants to geometrically manipulate a spline curve, this is rather easy: for $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{y} \in \mathbb{R}^d$ we just have

$$\boldsymbol{A}S_m\boldsymbol{d} + \boldsymbol{y} = \sum_{j=1}^{n} \left(\boldsymbol{A}\boldsymbol{d}_j + \boldsymbol{y}\right)\, N_j^m\left(\cdot \,|\, T\right) - \boldsymbol{y}\underbrace{\sum_{j=1}^{n} N_j^m\left(\cdot \,|\, T\right)}_{=1} + \boldsymbol{y},$$

so that any affine transformation is performed by transforming the control polygon. This allows us to easily rotate and translate curves.

2.2 Differentiation and integration

In many cases it is important to compute derivatives or antiderivatives of spline curves; we will see applications for this later. To derive a formula, we make use of the derivative formula (1.8) for the B–splines and get

$$\begin{aligned} S_m\boldsymbol{d}'(x) &= \sum_{j=1}^{n} \boldsymbol{d}_j \left[\frac{m}{t_{j+m}-t_j} N_j^{m-1}\left(x \,|\, T\right) - \frac{m}{t_{j+m+1}-t_{j+1}} N_{j+1}^{m-1}\left(x \,|\, T\right)\right] \\ &= m\sum_{j=1}^{n} \frac{\boldsymbol{d}_j}{t_{j+m}-t_j} N_j^{m-1}\left(x \,|\, T\right) - m\sum_{j=2}^{n+1} \frac{\boldsymbol{d}_{j-1}}{t_{j+m}-t_j} N_j^{m-1}\left(x \,|\, T\right) \\ &= m\sum_{j=1}^{n+1} \frac{\boldsymbol{d}_j - \boldsymbol{d}_{j-1}}{t_{j+m}-t_j}\, N_j^{m-1}\left(x \,|\, T\right) \end{aligned}$$

$$=: \quad S_{m-1}\boldsymbol{d}^1(x),$$

where we made use of the permanent convention that $\boldsymbol{d}_0 = \boldsymbol{d}_{n+1} = 0$. Therefore, the coefficient matrix $\boldsymbol{d}^1$ of the "derivative curve" $S_{m-1}\boldsymbol{d}^1$ is obtained as

$$\boldsymbol{d}^1 = m\,\boldsymbol{d}\,\boldsymbol{D}_n \begin{bmatrix} t_{m+1} - t_1 & & \\ & \ddots & \\ & & t_{m+n+1} - t_{n+1} \end{bmatrix}^{-1} := m\,\boldsymbol{d}\,\boldsymbol{D}_n\,\Delta_m T \tag{2.1}$$

where the **difference matrix** $\boldsymbol{D}$ is defined as

$$\boldsymbol{D}_n = \begin{bmatrix} 1 & -1 & & \\ & \ddots & \ddots & \\ & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{n\times n+1}$$

and $\Delta_m T$ is the diagonal matrix formed by the inverse m–differences between the knots:

$$\Delta_m T = \text{diag}\,\left[(t_{j+m} - t_j)^{-1} \;:\; j = 1, \ldots, n+1\right].$$

By iteration of (2.1), we can also compute higher order derivatives as

$$\begin{aligned} \boldsymbol{d}^2 &= \boldsymbol{d}\,m(m-1)\boldsymbol{D}_n\Delta_m T\,\boldsymbol{D}_{n+1}\Delta_{m-1}T, \\ &\vdots \\ \boldsymbol{d}^k &= \boldsymbol{d}\,\frac{m!}{(m-k)!}\prod_{j=0}^{k-1}\boldsymbol{D}_{n+j}\Delta_{m-j}T =: \boldsymbol{d}\,\boldsymbol{G}_k. \end{aligned}$$

Since for any x and any $\boldsymbol{d}$

$$\boldsymbol{d}\,\boldsymbol{N}_m^{(k)}(x) = \boldsymbol{d}^k\,\boldsymbol{N}_{m-k}\,(x\,|\,T) = \boldsymbol{d}\,\boldsymbol{G}_k\,\boldsymbol{N}_{m-k}\,(x\,|\,T)$$

it also follows that the spline vectors satisfy the relationship

$$\boldsymbol{N}_m^{(k)} = \boldsymbol{G}_k\,\boldsymbol{N}_{m-k}\,(\cdot\,|\,T)\,, \tag{2.2}$$

which allows us to directly connect these column vectors.

However, we missed a little problem at the moment! If we have boundary knots of multiplicity $m+1$, then $t_{m+1} = t_1$ and $t_{n+m+1} = t_{n+1}$ so that $\Delta_m T$ takes the form

$$\Delta_m T = \begin{bmatrix} 0 & & & & \\ & * & & & \\ & & \ddots & & \\ & & & * & \\ & & & & 0 \end{bmatrix}^{-1}$$

which, of course, tries the impossible task of inverting a singular matrix. Nevertheless, this is not really a problem if we take into account that the first B–spline, N_1^{m-1} is always supported on $[t_1, t_{m+1}]$ and thus cannot contribute to the behavior of the derivative on $I_m(T) = [t_{m+1}, t_{m+1}]$. The same also holds true for the rightmost boundary knot t_{m+n+1}.

Consequently, as long as we are interested in the behavior on $I_m(T)$ only[3], we can simply ignore the first and the last B–spline and thus get that

$$S_m \boldsymbol{d}' = S_{m-1} \widehat{\boldsymbol{d}}^1 \left(\cdot \,|\, \widehat{T} \right)$$

where $\widehat{T} = \widehat{T}_{m-1,n-1} = \{t_2, \ldots, t_{n+m}\}$ and

$$\widehat{\boldsymbol{d}}^1 = m\, \boldsymbol{d}\, \widehat{\boldsymbol{D}}_n \Delta_m \widehat{T}, \qquad \widehat{\boldsymbol{D}}_n = \begin{bmatrix} -1 & & \\ 1 & \ddots & \\ & \ddots & -1 \\ & & 1 \end{bmatrix} \in \mathbb{R}^{n \times n-1}.$$

The respective matrices $\widehat{\boldsymbol{G}}_k \in \mathbb{R}^{n \times n-k}$ can thus be recursively computed as

$$\widehat{\boldsymbol{G}}_0 = \boldsymbol{I}, \qquad \widehat{\boldsymbol{G}}_k = \widehat{\boldsymbol{G}}_{k-1}\, \widehat{\boldsymbol{D}}_{n-k+1} \Delta_{m-k+1} \widehat{T}_k, \qquad \widehat{T}_k = \{t_{k+1}, \ldots, t_{m+n+1-k}\},$$

and by means of

$$\widehat{\boldsymbol{d}}^k \boldsymbol{N}_{m-k} \left(\cdot \,|\, \widehat{T}_k \right) = \boldsymbol{d}\, \boldsymbol{N}_m^{(k)} \left(\cdot \,|\, T \right),$$

we obtain the counterpiece of (2.2):

$$\boldsymbol{N}_m^{(k)} \left(\cdot \,|\, T \right) = \widehat{\boldsymbol{G}}_k\, \boldsymbol{N}_{m-k} \left(\cdot \,|\, \widehat{T}_k \right). \tag{2.3}$$

Let us summarize.

> Depending on which concept of derivatives one wants to consider, there are two ways of determining the derivative of a spline curve, one that successively *removes* outmost boundary knots, one that preserves them. Nevertheless:
>
> 1. On the "crucial" interval $I_m(T)$ both concepts are equivalent.
> 2. Boundary knots of multiplicity $m + 1$ have to use the second concept.

We can also use the derivative formula to compute the integral or antiderivative of a spline curve as follows:

$$\int_{-\infty}^{x} S_m \boldsymbol{d}(t)\, dt = S_{m+1} \boldsymbol{d}^*(x), \qquad x \in [t_m, t_n], \tag{2.4}$$

[3] Which is automatically indicated by giving the boundary knots multiplicity $m + 1$!

where

$$\boldsymbol{d}_j^* = \sum_{k=0}^{j} \frac{t_{k+m+1} - t_k}{m+1} \boldsymbol{d}_k, \qquad j = 0, \ldots, n-1. \tag{2.5}$$

This formula is easily verified by applying the above derivative formula on (2.4) – this is a nice and simple exercise.

2.3 Numerical integration of spline functions

Even if the formula (2.4) gives us a convenient way to determine the formal antiderivative of a spline curve, it is only of limit use in computing spline integrals since mostly we will need integrals of the form

$$\int_{\mathbb{R}} N_j^m\,(x\,|\,T)\;N_k^m\,(x\,|\,T)\;dx$$

which will appear naturally in the concept of least squares approximations. To that end, we will make use of composite **Gauss quadrature formula**, cf. [7, 8]. First note that the product of two splines is again a piecewise polynomial function on T, just the polynomial degree is $2m$ instead of m. So it suffices to use a Gaussian quadrature of order m, that is, with $m+1$ knots and weights, on each interval since such a quadrature integrates all polynomials of degree at most $2m+1$ *exactly*.

So, let $\xi_0, \ldots, \xi_m \in [-1, 1]$ the Gauss nodes of order m for $\int_{-1}^{1} dx$ and $w_0, \ldots, w_m$ the associated weights which can be found e.g. in [1]. Since

$$\int_{t_j}^{t_{j+1}} f(x)\,dx = \frac{2}{t_{j+1} - t_j} \int_{-1}^{1} f\left(t_j + (x+1)\frac{t_{j+1} - t_j}{2}\right)\,dx,$$

so that the associated quadrature takes the form

$$\int_{t_j}^{t_{j+1}} f(x)\,dx \simeq \frac{2}{t_{j+1} - t_j} \sum_{k=0}^{m} w_k\, f\left(\frac{t_{j+1} + t_j}{2} + \frac{t_{j+1} - t_j}{2}\,\xi_k\right). \tag{2.6}$$

The integral over the "complete" curve can then easily be obtained by summing from $j = m+1, \ldots, n$ or $j = 1, \ldots, n+m$, respectively.

Again, this can be implemented efficiently in terms of linear algebra[4] by defining the evaluation vector

$$\Xi_j := \left[t_j^* + \frac{t_{j+1} - t_j}{2}\,\xi_k \;:\; k = 0, \ldots, m\right], \qquad t_j^* = \frac{t_{j+1} + t_j}{2}, \qquad j = m, \ldots, n,$$

the matrix

$$\boldsymbol{F} = \text{diag}\left[\frac{2}{t_{j+1} - t_j} \;:\; j = m+1, \ldots, n\right]\,[f\,(\Xi_j) \;:\; j = m+1, \ldots, n] \tag{2.7}$$

[4] In `Octave` as well as in `Matlab` the users are encouraged to replace any type of loop by vectorization. However, it would be more honest to state that loops are discouraged!

and compute the integral as

$$\int_{I_m(T)} f(x)\,dx \;=\; \sum_{j=m}^{n} \int_{t_j}^{t_{j+1}} f(x)\,dx = \mathbf{1}^T \boldsymbol{F}\boldsymbol{w}$$

where $\boldsymbol{w} = [w_j \;:\; j = 0, \ldots, m]$ – the Gauss weights guarantee the validity of the equality sign here.

Of course, (2.7) is only valid for single knots but since knot "intervals" between multiple knots degenerate to a point and thus are irrelevant for the integration process, there is not much to take care of.

God does arithmetic

C. F. Gauss

3 Interpolation and minimality

It's common belief that splines are made for interpolation. There are arguments in favor of this statement and arguments that do not support it. So let us begin with the "positive" side.

3.1 Interpolation at knots and the natural spline

The **interpolation problem** consists of finding, for given points or **sites** $X \subset \mathbb{R}$ and given data $y \in \mathbb{R}^X$, a function f such that

$$f(X) = y, \qquad \text{i.e.} \qquad f(x) = y_x, \qquad x \in X. \tag{3.1}$$

Interpolation from a linear space like $\mathbb{S}_m(T)$ can always be transfered into a linear algebra problem: if we write a spline in its B–spline representation then the interpolation problem looks

$$\sum_{j=1}^{n} \boldsymbol{d}_j \, N_j^m \, (x \,|\, T) = \boldsymbol{y}_x \qquad x \in X$$

which can be rewritten into matrix form as

$$\boldsymbol{d}\, \boldsymbol{N}_m(X) = \boldsymbol{y}.$$

The matrix $\boldsymbol{N}_m(X)$ (or its transpose, depending on the "school") is called **Vandermonde matrix** of the interpolation problem; this also explains the name of the function `BSplVander`.

The interpolation problem has a unique solution for all prescribed data if and only if the matrix $\boldsymbol{N}_m(X)$ is nonsingular.

If we want to interpolate with splines, there are some "important" or even natural sites present, namely, the knots T. So it is apparently not a bad idea to interpolate at the **relevant knots** $t_{m+1}, \ldots, t_{n+1}$. However, these are only $n-m+1$ points so that the matrix $\boldsymbol{N}_m\left(\{t_{m+1}, \ldots, t_{n+1}\}\right)$ is not even a square one and thus cannot be invertible – we are facing an *underdetermined* system if we consider the interpolation problem

$$S_m \boldsymbol{d}\,(t_{m+j}) = \boldsymbol{y}_j, \qquad j = 1, \ldots, n-m+1.$$

Do we explicitly have to say that this only works for simple knots?

To make the interpolating spline unique, we will have to add further linear conditions on the spline, preferably at the end points. If m is an odd number, then $m-1$ is even and one could add $(m-1)/2$ conditions *symmetrically* to both end points of the interval. Some typical choices would be

Natural conditions: Make derivatives of *high order* vanish,

$$S_m^{(k)}\boldsymbol{d}\,(t_j) = 0, \qquad k = \frac{m-1}{2}+1,\ldots,m-1, \quad j = m+1, n+1.$$

Hermite conditions: Prescribe values for the *derivatives*

$$S_m^{(k)}\boldsymbol{d}\,(t_j)\,, \qquad k = 1,\ldots,\frac{m-1}{2}, \quad j = m+1, n+1.$$

Periodic conditions: "Close" the curve periodically,

$$S_m^{(k)}\boldsymbol{d}\,(t_{m+1}) = S_m^{(k)}\boldsymbol{d}\,(t_{n+1})\,, \qquad k = 1,\ldots,m-1.$$

Except the natural conditions[5] these additional conditions require additional assumptions as well: in the Hermite case one has to "guess" derivatives at the end points, for example by taking differences of the data to be interpolated, while the periodic conditions only make sense if the curve is periodic itself, i.e., if $\boldsymbol{y}_1 = \boldsymbol{y}_{n-m+1}$.

3.2 Minimality of the natural spline

There is another method to solve underdetermined systems, which is by *minimization*. Instead of looking for a solution to (3.1), one minimizes (or maximizes) a certain functional subject to (3.1). We will return to this issue later in wider generality. Here we simply realize that the **natural spline** defined just above is indeed the solution to a minimization problem.

To that end, we define the energy seminorms

$$|f|_k := \int_{I_m(T)} \left(f^{(k)}(x)\right)^2\,dx$$

and define the **natural spline interpolant** $S_{m,T}f$ of order m with respect to T at a function $f \in C(\mathbb{R})$ as the unique natural spline which satisfies

$$S_{m,T}f\,(t_j) = f\,(t_j)\,, \qquad j = m+1,\ldots,n+1.$$

Theorem 3.1 *Let $m = 2r+1$ be an odd number and $f \in C^{r+1}(\mathbb{R})$. Then*

$$|S_{m,T}f|_{r+1} \le |f|_{r+1}\,. \tag{3.2}$$

[5] Though it is not clear from this definition what is so "natural" about it.

In more prosaic words, we can express Theorem 3.1 as

> Among all $C^{(m+1)/2}$–solutions of an interpolation problem at $t_{m+1}, \ldots, t_{n+1}$, the natural spline minimizes the energy functional $|\cdot|_{r+1}$.

This finally explains the name "natural spline". Indeed, the mathematical object is named after a tool to draw interpolating curves by means of a flexible ruler (Fig 3.1) – which bends into a shape that minimizes the bending energy of the curve. Now, an approximation, but only an approximation, of the bending energy is $|\cdot|_2$ so that the natural cubic spline is indeed an approximation to the "real" object. Nevertheless,w we have to record that

Figure 3.1: A "real" spline. The weights (and they rightfully carry that name) fix the flexible ruler at the interpolation points which takes an energy minimal interpolating shape. Thanks to Dr. Hollenhorst of the University Giessen Computing Center for permitting me to use the device

> The cubic natural spline is not a spline!

Since the proof of Theorem 3.1 is elementary and fairly instructive, let us have a look at it.

Proof of Theorem 3.1: We use the abbreviations $Sf = S_{m,T}f$ and $I = I_m(T)$ and begin with

$$\begin{aligned} |f - Sf|_{r+1}^2 &= \int_I \left(f^{(r+1)}(x) - (Sf)^{(r+1)}(x) \right)^2 dx \\ &= \int_I \left(f^{(r+1)}(x) \right)^2 - 2 f^{(r+1)}(x)(Sf)^{(r+1)}(x) + \left((Sf)^{(r+1)}(x) \right)^2 dx \\ &= |f|_{r+1,I}^2 - 2 \int_I \left(f^{(r+1)}(x) - (Sf)^{(r+1)}(x) \right) (Sf)^{(r+1)}(x)\, dx - |Sf|_{r+1}^2 . \end{aligned} \tag{3.3}$$

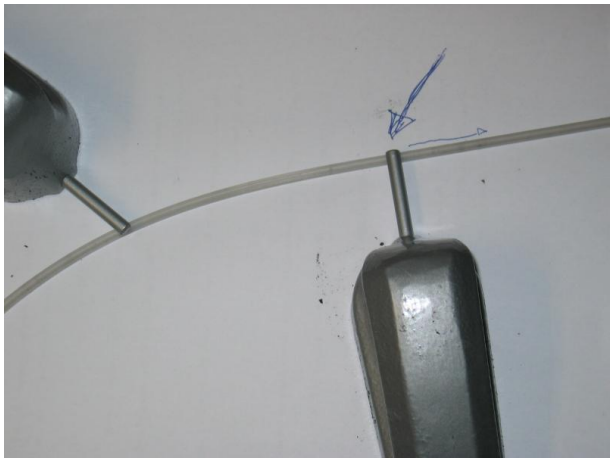

Figure 3.2: The natural side conditions of a cubic spline: The free ends outside the boundary assume linear shape.

For $j = m+1, \ldots, n$ partial integration gives us

$$\begin{aligned}
&\int_{t_j}^{t_{j+1}} \left(f^{(r+1)}(x) - Sf^{(r+1)}(x)\right) Sf^{(r+1)}(x)\, dx \\
&= \left(f^{(r)}(x) - Sf^{(r)}(x)\right) Sf^{(r+1)}(x)\Big|_{t_j}^{t_{j+1}} - \int_{t_j}^{t_{j+1}} \left(f^{(r)}(x) - Sf^{(r)}(x)\right) Sf^{(r+2)}(x)\, dx \\
&= \sum_{l=0}^{k} (-1)^{r-l} \left(f^{(r-l)}(x) - Sf^{(r-l)}(x)\right) Sf^{(r+l+1)}(x)\Big|_{t_j}^{t_{j+1}} \\
&\qquad + (-1)^{k+1} \int_{t_j}^{t_{j+1}} \left(f^{(r-k)}(x) - S_f^{(r-k)}(x)\right) \underbrace{Sf^{(r+k+2)}(x)}_{=0 \text{ for } k=r}\, dx, \qquad k = 1, \ldots, r \\
&= \sum_{l=0}^{r} (-1)^{r-l} \left(f^{(r-l)}(x) - Sf^{(r-l)}(x)\right) Sf^{(r+l+1)}(x)\Big|_{t_j}^{t_{j+1}},
\end{aligned}$$

Summing over j all the interior term cancel and we get

$$\begin{aligned}
&\int_{I} \left(f^{(r+1)}(x) - Sf^{(r+1)}(x)\right) Sf^{(r+1)}(x)\, dx \\
&\quad = \sum_{l=0}^{r} (-1)^{r-l} \left(f^{(r-l)}(x) - Sf^{(r-l)}(x)\right) S_f^{(r+l+1)}(x)\Big|_{t_m}^{t_{n+1}} = 0.
\end{aligned}$$

If we substitute this into (3.3) we end up with

$$|Sf|_{r+1}^2 = |f|_{r+1}^2 - |f - Sf|_{r+1}^2 \leq |f|_{r+1}^2$$

with equality iff $f - Sf \in \Pi_r$. □

3.3 Schoenberg, Whitney, Greville

In the preceding section we insisted on interpolation at the "relevant" knots and had to pay a price for that by being forced to add further conditions that made the interpolation problem uniquely solvable. However, it is an interesting question how **interpolation sites** $x_1, \ldots, x_n$ have to be placed such that we can interpolate from the n–dimensional vector space $\mathbb{S}_m(T_{m,n})$.

It is intuitively clear that the interpolation points cannot be chosen arbitrarily: since any spline is a polynomial of degree at most m when restricted to a (nontrivial) knot interval $[t_j, t_{j+1}]$, such an interval cannot contain more than $m+1$ sites as then the associated *local* interpolation problem would become unsolvable. Hence, there must be a relationship between the sites and the knots and this relationship is as simple as one may imagine.

Theorem 3.2 (Schoenberg & Whitney) *Interpolation at the sites $X = \{x_j : j = 1, \ldots, n\}$, $x_1 < \cdots < x_n$ has a unique solution in $\mathbb{S}_m(T)$ if and only if*[6]

$$t_j < x_j < t_{j+m+1}, \qquad j = 1, \ldots, m. \tag{3.4}$$

Geometrically, we can slightly rephrase condition (3.4):

> Each site has to be at a position where the B–spline with the same index is positive.

A complete proof of Theorem 3.2 is beyond our scope here[7], but it is rather easy and illustrative to see why this condition is necessary. Suppose, for example, that there is some index j such that $x_j \leq t_j$, and therefore also $x_j \leq t_k$, $k \geq j$, because the knots are in increasing order. Since the B–splines N_k^m are supported on $[t_k, t_{k+m+1}]$, it follows that

$$N_k^m(x_j \,|\, T) = 0, \qquad k = j, \ldots, n.$$

Now look at the first j columns of the Vandermonde matrix $\boldsymbol{N}_m(X)$, that is,

$$\begin{bmatrix} N_1(x_1\,|,T) & \ldots & N_1(x_j\,|,T) \\ \vdots & \ddots & \vdots \\ N_{j-1}(x_1\,|,T) & \ldots & N_{j-1}(x_j\,|,T) \\ N_j(x_1\,|,T) & \ldots & N_j(x_j\,|,T) \\ \vdots & \ddots & \vdots \\ N_n(x_1\,|,T) & \ldots & N_n(x_j\,|,T) \end{bmatrix} = \begin{bmatrix} N_1(x_1\,|,T) & \ldots & N_1(x_j\,|,T) \\ \vdots & \ddots & \vdots \\ N_{j-1}(x_1\,|,T) & \ldots & N_{j-1}(x_j\,|,T) \\ 0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{bmatrix}$$

[6] To be precise, this only holds when the multiplicity of the knots is $\leq m$ so that the B–splines are at least continuous.

[7] But it is not **sooooooo** difficult either.

and these columns are linearly dependent as they consist of j vectors which are different from zero at most in the first $j-1$ components. Hence, the matrix cannot be invertible. If, on the other hand, there is some j such that $x_j \geq t_{j+m+1}$ then an analogous reasoning with the first j rows of the matrix leads to the same result.

So, Theorem 3.2 tells precisely where to put the knots in order to achieve unique interpolation. What it does not tell us is how to *systematically* chose the sites from the knots. But there is a "canonical" choice, called the **Greville abscissae**:

$$x_j = \frac{1}{m} \sum_{k=1}^{m} t_{j+k}, \qquad j = 1, \ldots, n. \tag{3.5}$$

Note that the two "outmost" knots, t_1 and t_{n+m+1} do not enter the definition of the Greville abscissae. It is easy to see that they are indeed suitable interpolation points since

$$t_j \leq t_{j+1} \leq \frac{1}{m} \left(t_{j+1} + \cdots + t_{j+m} \right) \leq t_{j+m} \leq t_{j+m+1}$$

with equality in the leftmost inequality if and only if $t_j = \cdots = t_{j+m}$ and equality in the rightmost inequality if and only if $t_{j+1} = \cdots = t_{j+m+1}$, hence the **Schoenberg–Whitney condition** (3.4) can only be violated in the appearance of inner knots of multiplicity $m+1$. But that is forbidden for interpolation anyway.

At a knot of multiplicity $m+1$ the spline curve has a discontinuity and thus an interpolation value there cannot be determined – is it the left or the right limit?

Another justification of the Greville abscissae comes from looking at the **Schoenberg operator**

$$\mathscr{S}_m f := \sum_{j=1}^{n} f\left(x_j\right) \, N_j^m \left(\cdot \,|\, T \right), \tag{3.6}$$

which does not interpolate but uses the values of f at the Greville abscissae as "control points". We will return to this operator later.

3.4 Parametric interpolation

In applications, the abscissae of the interpolation are seldom given. The only available information are a sequence of points, $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$, which come, for example from scanning or sampling a workpiece. Therefore, the sites as well as the knots can, shall and have to be determined from the data! There are many strategies to find suitable parameter values, some of them listed in [6], but practically all of them motivated by heuristic considerations.

Intuitively[8], the distance between the interpolation sites x_j should match the distance between the data points, that is,

$$x_1 \;=\; 0$$

[8] So we are at heuristics here as well.

$$\begin{aligned} x_2 &= \|\boldsymbol{y}_2 - \boldsymbol{y}_1\|_2 \\ &\vdots \\ x_k &= x_{k-1} + \left\|\boldsymbol{y}_k - \boldsymbol{y}_{k-1}\right\|_2 = \sum_{j=2}^{k} \left\|\boldsymbol{y}_j - \boldsymbol{y}_{j-1}\right\|, \qquad k = 2, \ldots, n. \end{aligned}$$

Next, we can try to determine the knots T such that $\boldsymbol{x} = [x_j \,:\, j = 1, \ldots, n]$ are the associated Greville abscissae and, recalling our initial considerations, we can also insist on multiplicity $m+1$ on the boundary, that is $t_1 = \cdots = t_{m+1}$. Then we could solve successively from (3.5):

$$\begin{aligned} t_1 = \cdots = t_{m+1} &= x_1 \\ t_{m+2} &= m\, x_2 - \sum_{k=1}^{m-1} t_{k+2} \\ &\vdots \\ t_{m+j} &= m\, x_j - \sum_{k=1}^{m-1} t_{k+j}, \qquad j = 2, \ldots, n \end{aligned} \tag{3.7}$$

Note, however, that we have no multiplicity $m+1$ at the right boundary then! Moreover, (3.7) is not a good strategy as the following numerical example shows: we sample the circumference of the circle,

```
octave> y = [ cos( 2*pi*(0:6)/7 ); sin(  2*pi*(0:6)/7 ) ]
y =

   1.00000   0.62349  -0.22252  -0.90097  -0.90097  -0.22252   0.62349
   0.00000   0.78183   0.97493   0.43388  -0.43388  -0.97493  -0.78183
```

and compute the knot sequence

```
octave> T=GrevilleKnots( 3,y )
ans =

 Columns 1 through 8:

  0.00000  0.00000  0.00000  0.00000  2.60330  2.60330  2.60330  5.20660

 Columns 9 through 11:

  5.20660  5.20660  5.20660
```

which shows that we get a triple knot in the interior. The associated Greville abscissae are

```
octave> T=GrevilleKnots( 3,y )
ans =

 Columns 1 through 8:

  0.00000  0.00000  0.00000  0.00000  2.60330  2.60330  2.60330  5.20660
```

```
 Columns 9 through 11:

  5.20660  5.20660  5.20660
```

And indeed, the solution is disappointing – though the function in Fig 3.3 interpolates at the corners of the polygon, it does not really look exciting because of the triple knot. In other

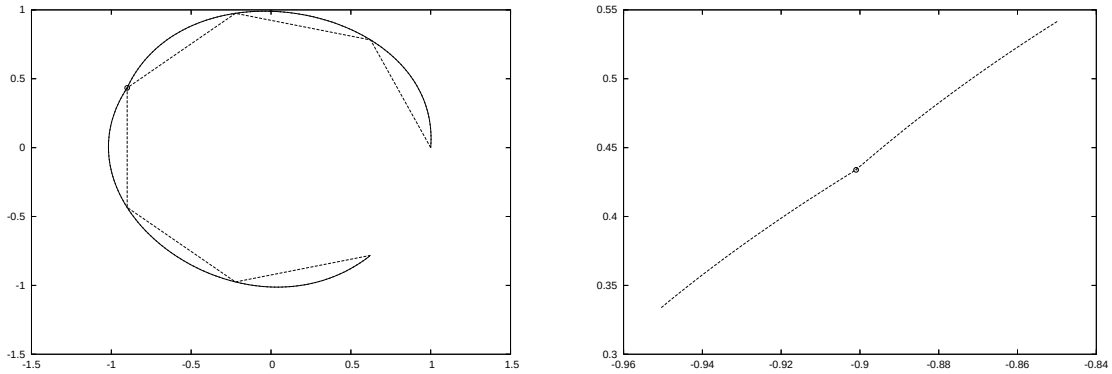

Figure 3.3: Interpolant whose knots are computed by the formula (3.7). The triple knot cause the result to be non–differentiable at the marked knot, see detail *(right)*.

words, the naive strategy is not a working one. It even gets worse as there are examples when the computed knot sequence is not increasing!

In general, any valid knot sequence $\boldsymbol{t} = [t_j \ : \ j = 2, \ldots, m+n]$ must be a solution of

$$\frac{1}{m}\begin{bmatrix} 1 & \ldots & 1 & & \\ & \ddots & & \ddots & \\ & & 1 & \ldots & 1 \end{bmatrix} \boldsymbol{t} =: \boldsymbol{M}\boldsymbol{t} = \boldsymbol{x}, \qquad \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \boldsymbol{t} = \boldsymbol{D}^T\boldsymbol{t} \geq 0. \tag{3.8}$$

Note that (3.8) is a system of linear inequalities which can be solved together with the minimization of an arbitrary linear functional by means of linear programming, see [5, 16] and also [9] for the connection to the theory of games. In fact, we could try to maximize the distance between the knots, i.e., solve the minimization problem

$$\min_{\boldsymbol{t},s,\boldsymbol{s}} -s, \qquad \begin{bmatrix} \boldsymbol{M} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{D}^T & -\boldsymbol{1} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{t} \\ s \\ \boldsymbol{s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{0} \end{bmatrix}, \qquad \boldsymbol{t}, s, \boldsymbol{s} \geq 0, \tag{3.9}$$

which can be solved by means of linear programming. It turns out, however, that for many configurations of Greville abscissae this optimization problem has *no* solution, the associated feasible set is empty.

Therefore, the “standard” approach is to define knots whose distance coincides with the distance between the interpolation points and then use a natural spline interpolant. Alternatively, one could use knots which are spaced according to the distances of the data points and then use their respective Greville abscissae.

A mathematician, like a painter or poet, is a maker of patterns. If his patterns are more permanent than theirs, it is because they are made with ideas.

G. H. Hardy, *A mathematician's apology*

What is wrong with interpolation? 4

So, splines can interpolate, either at Greville abscissae or at the knots, but there are some unavoidable problems that simply come from interpolating with smooth functions.

4.1 Dependency on the parameterization

The first problem with interpolation is that the outcome depends strongly on the parameterization of the interpolation sites. Let us illustrate this with the seven points on the circle from above and cubic splines. Thus we need $n+m+1 = 7+3+1 = 11$ knots with boundary knots of multiplicity 4. Let us start equally spaced, that is,

```
octave> T = [ 0 0 0 0 1 2 3 4 4 4 4 ];
octave> X = Greville( 3,T )
X =

  0.00000  0.33333  1.00000  2.00000  3.00000  3.66667  4.00000
```

where the function `Greville` clearly computes the Greville abscissae of given order for a given knot sequence. Next, we set up the Vandermonde matrix and compute the control points via

```
octave> V = BSplVander( 3,T,X )'; d = ( V \ y' )';
```

The little bit of transposition is for technical reasons only; and indeed, the result in Fig 4.1.

OK, that is not too bad, but now we increase the distance between the knots and therefore between the Greville abscissae in a non–proportional fashion:

```
octave> T = [ 0 0 0 0 1 4 9 16 16 16 16 ];X = Greville( 3,T );
```

Not surprisingly, the respective interpolant has a larger deviation in the "later" part of the curve while the interpolant with decreasing differences between the knots, generated by

```
octave> T = [ 0 0 0 0 1.3 1.7 1.9 2 2 2 2 ];X = Greville( 3,T );
```

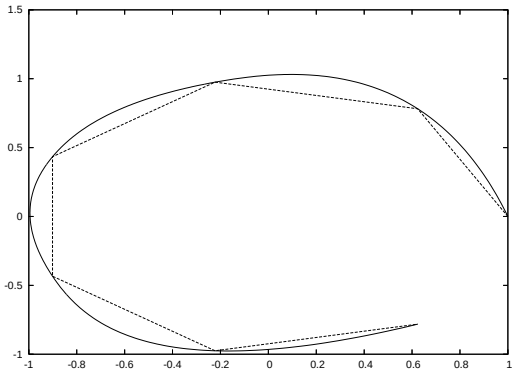

Figure 4.1: The spline interpolant with equidistant knots and associated equidistant Greville abscissae. It can be seen that the deviation of the spline from the data polygon becomes larger where the polygon has "short" pieces.

takes its maximal deviation in its "early" stages, see Fig 4.2. This shows that the outcome is really sensitive to the chosen parameterization of the interpolation sites; unfortunately, as the example with the Greville abscissae shows, it is not so easy to choose these values appropriately.

Proper parameterization of the interpolating curve is crucial for the quality of the result.

4.2 Overshooting – the curse of interpolation

The really nasty side of interpolation by smooth functions becomes apparent when interpolating on lines that enter corners. Let us, for example, sample the lines $[0,0] \rightarrow [1,0]$ and $[1,0] \rightarrow [1,1]$ and put an interpolant through these points. So, let us try four points on each line

```
octave> y = [ (0:1/3:1 ); zeros( 1,4 ) ];
octave> y = [ y, [ ones( 1,3 ); ( 1/3:1/3:1 ) ] ];
```

choose the necessary 11 equidistributed knots,

```
octave> T = [ 0 0 0 (0:4) 4 4 4 ];
```

and compute the coefficients of the interpolant by our "standard" methods:

```
octave> d = ( BSplVander( 3,T,Greville( 3,T ) )'\y' )';
```

Fig. 4.3 shows the result of this interpolation and it the usually unwanted overshooting. Of course, the effect becomes smaller if the sampling density is higher, but it still persists.

What is worse than the fact that at the corner there is a large deviation from the "ideal" shape of the curve is the loss of **shape information**: the original curve is convex, the interpolant is not. This, however, is a general principle:

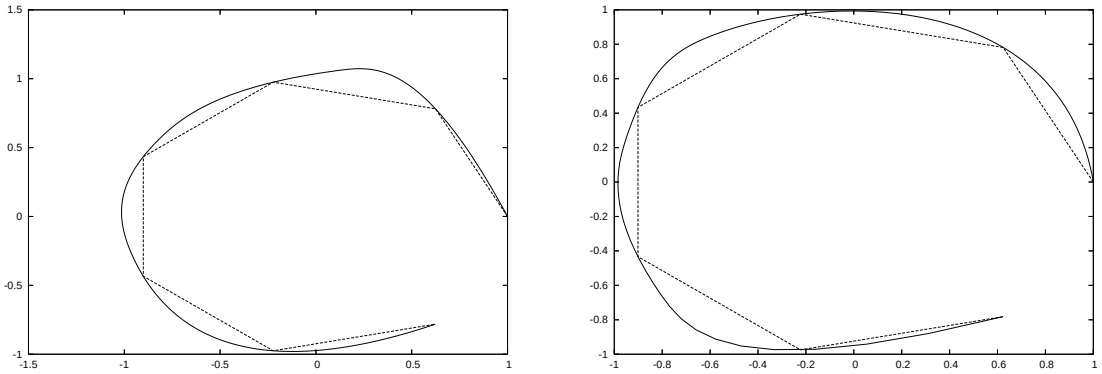

Figure 4.2: The interpolants with increasing *(left)* and decreasing *(right)* differences between the knots.

Differentiable interpolants cannot preserve convexity of the data to be interpolated.

The simplest example that shows that convex differentiable interpolation of convex data is impossible, interpolates $f(x) = |x|$ at the points 0, $\pm\frac{1}{2}$ and ± 1. We will show that any convex C^1–interpolant must be linear on $[-1, 0]$ as well as on $[0, 1]$ and thus cannot be differentiable at 0.

Denote the interpolant by g. Convexity of g implies that for $x \in [0, 1]$ we have

$$g(x) = g\left(x\,1 + (1-x)\,0\right) \leq (1-x)\;g(0) + x\,g(1) = (1-x)\;f(0) + x\,f(1) = f(x),$$

hence $g(x) \leq f(x)$, $x \in [0, 1]$. Moreover,

$$g'\left(\frac{1}{2}\right) = \lim_{h\to 0^+} \frac{g\left(\frac{1}{2}+h\right) - g\left(\frac{1}{2}\right)}{h} \leq \frac{f\left(\frac{1}{2}+h\right) - f\left(\frac{1}{2}\right)}{h} = 1$$

as well as

$$g'\left(\frac{1}{2}\right) = \lim_{h\to 0^+} \frac{g\left(\frac{1}{2}\right) - g\left(\frac{1}{2}-h\right)}{h} \geq \frac{f\left(\frac{1}{2}\right) - f\left(\frac{1}{2}-h\right)}{h} = 1,$$

so that $g'\left(\frac{1}{2}\right) = 1$. Now suppose that there exists $x^* \in \left[0, \frac{1}{2}\right]$ such that $g\left(x^*\right) < f\left(x^*\right)$ and write $x \in \left[x^*, \frac{1}{2}\right]$ as

$$x = \lambda x^* + (1-\lambda)\frac{1}{2}, \qquad \lambda = \frac{\frac{1}{2} - x}{\frac{1}{2} - x^*} \in [0, 1].$$

Then, again by convexity,

$$g(x) \;\leq\; \lambda\, g\left(x^*\right) + (1-\lambda)\, g\left(\frac{1}{2}\right) = \lambda\, f\left(x^*\right) + (1-\lambda)\, f\left(\frac{1}{2}\right) + \lambda\left(g\left(x^*\right) - f\left(x^*\right)\right)$$

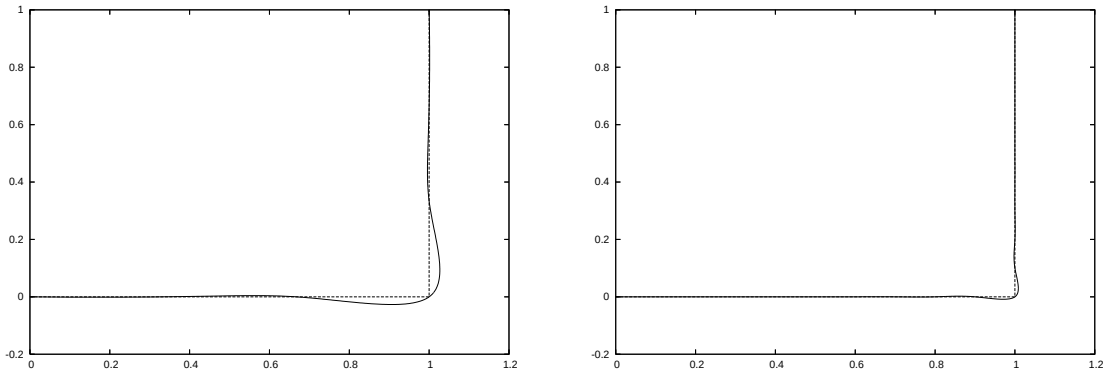

Figure 4.3: Overshooting of the spline interpolant in corners, with four *(left)* and ten *(right)* sampling points on the piecewise linear curve.

$$
\begin{aligned}
&= f\left(\frac{1}{2}\right) - \left(\frac{1}{2} - x\right) + \lambda\left(g\left(x^*\right) - f\left(x^*\right)\right) \\
&= x + \frac{\frac{1}{2} - x}{\frac{1}{2} - x^*}\left(g\left(x^*\right) - f\left(x^*\right)\right).
\end{aligned}
$$

Hence, for $h > 0$,

$$
g\left(\frac{1}{2}\right) - g\left(\frac{1}{2} - h\right) \;\geq\; h\left(1 - \frac{g\left(x^*\right) - f\left(x^*\right)}{\frac{1}{2} - x^*}\right)
$$

leading to the contradiction that

$$
g'\left(\frac{1}{2}\right) \geq 1 - \frac{g\left(x^*\right) - f\left(x^*\right)}{\frac{1}{2} - x^*} > 1.
$$

In the same way the existence of $x^* \in \left[\frac{1}{2}, 1\right]$ with $g\left(x^*\right) < f\left(x^*\right)$ would imply that $g'\left(\frac{1}{2}\right) < 1$, again a contradiction. Consequently, we must have that $g = f$ but then g cannot be differentiable at $x = 0$.

To close this section, we show one more picture, Fig. 4.4 which discourages the use of interpolation.

4.3 Handling corners

But what to do if the data stems from extremely precise measurements so that interpolation is necessary as deviation from the measurements cannot be tolerated. This situation occurs, for example in coordinate measurement technology. There is a simple trick to handle such configurations:

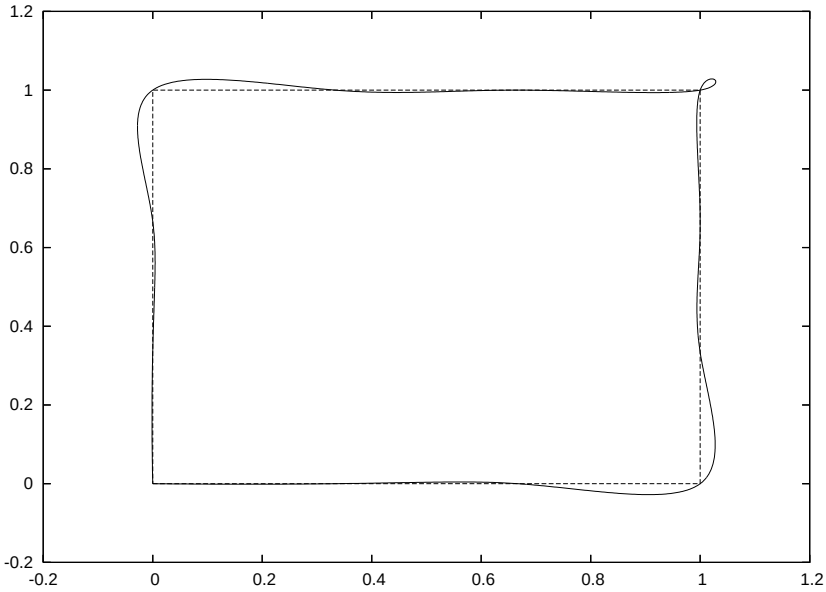

Figure 4.4: Running around a square by means of interpolation shows both possible effects: Overshooting and looping.

Apply edge detection methods and interpolate corners with knots of multiplicity m – these knots are always in the set of Greville abscissae.

There remains the question of how the edge detection can be performed. Some possible methods are

- Use the second (divided) differences of the data points – where these differences are large, corner can be assumed.
- Use wavelet methods. As illustrated in Fig. 4.5, the wavelet coefficients indicate the locations of the corners, one can even read information off their amplitude and the way in which they change signs.

In fact, a combination of the two approaches above has given very good results in a coordinate measurement application, [11]. It is, however, reasonable to do a finer piecewise linear resampling of the curve in order to get equidistant points on the piecewise linear curve.

4.4 Why approximation can be better

So, interpolation has its limits and they are due to principal reasons and not to technical limitations. In this respect, the reconstruction of a function f from sampled values by means of

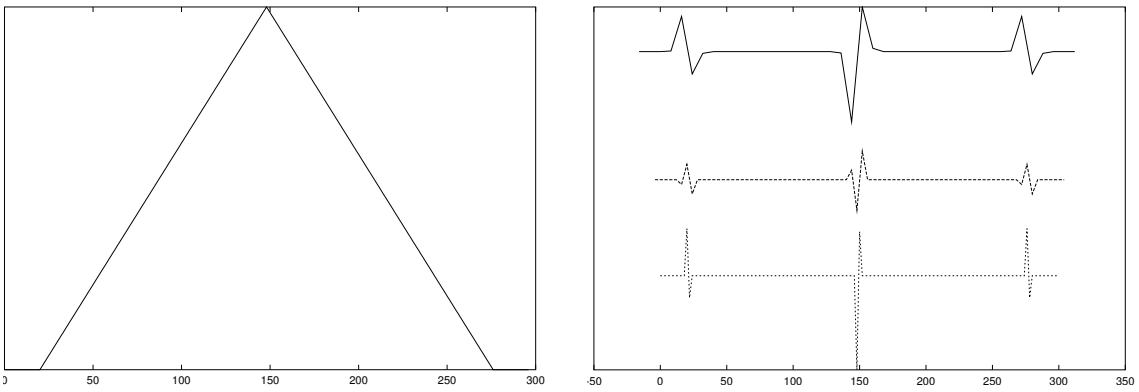

Figure 4.5: The "D_3" Daubechies wavelet coefficients *(right)* of a function *(left)*. Significant coefficients "point" to the edges.

interpolation can be tricky and can lead to artifacts. Though this can be overcome to some extent by edge detection, but even then there are quite a few detail problems that have to be taken into account.

A completely different approach to the **reconstruction** of functions from sampling values is the aforementioned **Schoenberg operator**

$$\mathscr{S}_m f = \sum_{j=1}^{n} f(x_j)\ N_j^m(\cdot\,|\,T), \qquad x_j = \frac{1}{m}\sum_{k=1}^{m} t_{j+k}, \quad j = 1, \ldots, n. \tag{4.1}$$

which is a so–called **quasi–interpolant** at the **Greville abscissae**. The use of Greville abscissae is no accident here!

Proposition 4.1 *The Schoenberg operator has **linear precision**, that is, $\mathscr{S}_m \ell = \ell$ for $\ell \in \Pi_1$ and $m \geq 1$.*

Proof: The case $m = 1$ is obvious, as there the quasi–interpolant is a piecewise linear interpolant of a linear function and thus linear again. For $m \geq 2$ we have that

$$\mathscr{S}_m \ell \;=\; \sum_{j=1}^{n} \ell\left(\frac{1}{m}\sum_{k=1}^{m} t_{j+k}\right)\ N_j^m(\cdot\,|\,T) = \frac{1}{m}\sum_{j=1}^{n}\sum_{k=1}^{m} \ell(t_{j+k})\ N_j^m(\cdot\,|\,T).$$

By (2.1),

$$\begin{aligned}(\mathscr{S}_m \ell)' &= \sum_{j=1}^{n} \frac{1}{t_{j+m} - t_j} \sum_{k=1}^{m} \left(\ell(t_{j+k} - t_{j-1+k})\right)\ N_j^{m-1}(\cdot\,|\,T) \\ &= \sum_{j=1}^{n} \frac{\ell(t_{j+m}) - \ell(t_j)}{t_{j+m} - t_j}\, N_j^{m-1}(\cdot\,|\,T) = \ell'\end{aligned}$$

which is a constant. Hence, $\mathscr{S}_m\ell$ is a linear function with the same slope and if at least one of the end knots has multiplicity m or $m+1$ then it interpolates there and the two functions coincide as claimed. □

The reason for the specific choice of Greville abscissae is the preservation of linear functions

But why linear precision? The reason is that we can prove a quantitative result for the overall error of approximation which ensures us that for sufficiently dense sampling (more precisely, for a sufficiently dense choice of knots) we can approximate any function as good as we want. We will also look at the proof as it is quite illuminating and serves as a prototype for more general results on **approximation order** of function spaces. For more information see [3, 17, 19]. So, here is the result.

Theorem 4.2 *Suppose that $f \in C^2[t_{m+1}, t_{n+1}]$, then*

$$\|f - \mathscr{S}_m f\|_\infty := \max_{x\in[t_{m+1},t_{n+1}]} |f(x) - \mathscr{S}_m f(x)| \le m^2 \|f''\|_\infty h^2, \tag{4.2}$$

where

$$h = \max_{j=1,\dots,m+n} |t_{j+1} - t_j| \tag{4.3}$$

*is the **knot spacing** distance.*

Proof: The proof is surprisingly simple. Consider a nontrivial knot interval $I_j = [t_j, t_{j+1})$, then

$$\mathscr{S}_m f|_{I_j} = \sum_{k=j-m}^{j} f\left(\frac{t_{k+1}+\cdots+t_{k+m}}{m}\right) N_k^m(\cdot|T), \tag{4.4}$$

and choosing $t^* = \frac{1}{2}(t_{j-m+1} + t_{j+m})$ as the midpoint of the interval spanned by the "relevant" knots, a Taylor expansion of f around t^* yields that

$$f(x) = \underbrace{f(t^*) + (x-t^*) f'(t^*)}_{=:T_1 f} + \frac{(x-t^*)^2}{2} f''(\xi), \qquad \xi \in (x, t^*).$$

Hence, for $x \in I_j$,

$$|f(x) - T_1 f(x)| = \frac{|x-t^*|^2}{2} |f''(\xi)| \le \frac{\frac{1}{4}(t_{j-m+1}+t_{j+m})^2}{2} \max_{x\in I_j} |f''(x)| \le \frac{m^2h^2}{2} \|f''\|_\infty \tag{4.5}$$

and moreover, for $k = j-m, \dots, j$,

$$|f - T_1 f|\left(\frac{t_{k+1}+\cdots+t_{k+m}}{m}\right) \le \frac{m^2h^2}{2} \|f''\|_\infty. \tag{4.6}$$

Therefore

$$\mathscr{S}_m\left(f-T_1f\right)|_{I_j} \leq \frac{m^2h^2}{2}\left\|f''\right\|_\infty \underbrace{\sum_{k=j-m}^{j} N_k^m\left(\cdot|T\right)}_{\leq 1} \leq \frac{m^2h^2}{2}\left\|f''\right\|_\infty$$

and so

$$\begin{aligned}\max_{x\in I_j}\left|\mathscr{S}_m f-f\right|(x) &\leq \max_{x\in I_j}\left|\mathscr{S}_m\left(f-T_1f\right)\right|(x)+\max_{x\in I_j}\left|T_1f-f\right|(x)\\ &\quad+\max_{x\in I_j}\underbrace{\left|\mathscr{S}_m\left(f-T_1f\right)\right|(x)}_{=0}\\ &\leq m^2\,h^2\,\left\|f''\right\|_\infty\end{aligned}$$

but since the right hand side is independent of j, the estimate holds globally which is nothing but (4.2). □

So what is the essence of the proof? It is composed of two main ingredients:

Locality: On a given knot interval I_j only a finite number of splines is "active" and the part of the Schoenberg interpolant that is affected by the behavior of f on I_j has size at most mh.

Polynomial preservation: The Greville abscissae are chosen such that the linear part of the Taylor polynomial becomes irrelevant in the difference between the function and the spline approximant.

This type of argument that probably dates back even before [18] indeed works in much more general circumstances, for example also in the analysis of integer translates of compactly supported functions in the framework of translation invariant spaces and wavelet analysis.

Also note that the above proof can be refined to a more "local" flavor: The approximation error is bounded by the **local** knot spacing and the **local** maximum of the second derivative so that a good approximation can still be obtained as long as the product remains small.

To obtain a good approximation, the Schoenberg operator needs a high density of knots where the function has a large second derivative ("curvature"). Where the second derivative is small, however, the knot spacing can be selected larger.

Finally, it should be mentioned that the Schoenberg operator is a **variation diminishing** approximation operator, see [12] which can also be found in [13]. There one also finds the famous **Marsden identity** which allows a generalization of evaluation at Greville abscissae to functionals[9] that allow a reproduction of polynomials of degree higher than 1 as well as a quantitative convergence proof for the Schoenberg operator.

[9] Involving derivatives of higher order, however.

In mathematics you don't understand things. You just get used to them.

J. von Neumann

5 Smoothing splines

We have already learned that the natural splines is the solution to a certain minimization problem. In this chapter we will consider interpolants or approximants which are defined as the solution of certain minimization problems. Typically, the functional to be minimized will be a weighted combination of an *approximation functional* ("How far is the spline from the given data or curve") and a *smoothness functional* ("How well–behaved is the spline") where we can vary the norms in which these functionals are taken as well as the properties of the spline to be taken into account.

5.1 Interpolation and minimization

The first idea is to choose, among all interpolants available, the one which minimizes a certain functional – this concept of **optimal interpolation**[10] has been very popular in statistical estimation and is famous under the name of **kriging**, cf [4].

Let us consider two examples in terms of splines. To that end, we use interpolation sites $X = \{x_j \,:\, j = 1, \ldots, n'\}$, $n' \leq n$, to obtain an *underdetermined* interpolation problem, and find a unique solution by minimizing an "additional" functional – in the context of linear models this is usually a covariance estimate. For example, our minimization problem could take the form

$$\min_{f \in \mathbb{S}_m(T)} \int_{\mathbb{R}} |f''(x)|^2 \, dx, \qquad f(X) = \boldsymbol{y}, \quad y \in \mathbb{R}^X. \tag{5.1}$$

Now, our matrix–vector notations turn out to be quite handy. Writing the spline as $f = S_m \boldsymbol{d} = \boldsymbol{d}\boldsymbol{N}_m$, the minimization problem (5.1) can be written in terms of the coefficient (row) vector $\boldsymbol{d}$ as

$$\min_{\boldsymbol{d}} \int_{\mathbb{R}} \boldsymbol{d}\boldsymbol{N}_m''(x)\boldsymbol{N}_m''(x)^T\boldsymbol{d}\, dx =: \boldsymbol{d}\boldsymbol{A}\boldsymbol{d}^T, \qquad \boldsymbol{d}\boldsymbol{N}_m(X) = \boldsymbol{y}, \tag{5.2}$$

which is a quadratic optimization problem with linear equality side conditions in $\boldsymbol{d}$. By means of **Lagrange multipliers**[11] this is turned into the linear system the minimum is determined by

[10] See, for example, the references in [14].

[11] Optimizers would speak of Kuhn–Tucker conditions

solving the linear system

$$\begin{bmatrix} 2\boldsymbol{A} & \boldsymbol{N}_m(X) \\ \boldsymbol{N}_m^T(X) & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}^T \\ \lambda \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{y}^T \end{bmatrix}, \tag{5.3}$$

where, according to (2.2)

$$\begin{aligned} \boldsymbol{A} &= \left[\int_{\mathbb{R}} N_j^m(x)'' \, N_k^m(x)'' \, dx \ : \ j,k = 1,\ldots,n \right] \\ &= \int_{\mathbb{R}} \boldsymbol{G}_2 \boldsymbol{N}_m(x) \boldsymbol{N}_m^T(x) \boldsymbol{G}_2^T \, dx = \boldsymbol{G}_2 \left(\int_{\mathbb{R}} \boldsymbol{N}_m(x) \boldsymbol{N}_m^T(x) \, dx \right) \boldsymbol{G}_2^T \end{aligned}$$

is a square positive semidefinite matrix. Moreover, since the **B–spline Gramian**

$$\boldsymbol{B}_m = \boldsymbol{B}_m(T) := \int_{\mathbb{R}} \boldsymbol{N}_m(x) \boldsymbol{N}_m^T(x) \, dx \tag{5.4}$$

is (strictly) positive definite, we have $\boldsymbol{A}\boldsymbol{x} = 0$ if and only if $\boldsymbol{G}_2\boldsymbol{x} = 0$ which is the case if and only if $\boldsymbol{x}$ is a segment of a linear sequence. Consequently, the linear system in (5.3) has a unique solution if and only if the interpolation problem does not have a solution by linear functions, that is, if the data is not located on a straight line.

This approach of minimal interpolation works in the same way for any **quadratic functional** to be minimized and always leads to a linear equation. More complicated but smooth functionals would result in *nonlinear* equations, but those require more intricate nonlinear methods.

Other functionals to be minimized can be obtained by *discretizing* the energy functional, i.e., by computing the vector

$$\boldsymbol{f} := f''(Z) = \boldsymbol{d}\boldsymbol{N}_m(Z)'', \qquad Z \subset \mathbb{R},$$

and then minimizing a vector norm of $\boldsymbol{f}$, for example

$$\|\boldsymbol{f}\|_1 = \sum_{z \in Z} |f''(z)| \, .$$

Suppose that $\#Z = N$, i.e., $Z = \{z_1, \ldots, z_N\}$, then the minimization problem would be

$$\min_{\boldsymbol{d},\boldsymbol{u},\boldsymbol{v}_1,\boldsymbol{v}_2 \geq 0} \boldsymbol{1}^T \boldsymbol{u} = \sum_{j=1}^{N} u_j, \qquad \begin{bmatrix} \boldsymbol{N}_m^T(X) & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{N}_m^T(Z)'' & \boldsymbol{I} & -\boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{N}_m^T(Z)'' & \boldsymbol{I} & \boldsymbol{0} & -\boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}^T \\ \boldsymbol{u} \\ \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}^T \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix},$$

and can be solved again by linear programming. Minimizing with respect to the ℓ_1–norm is a common principle in image processing and the associated inverse problems, cf. [10].

5.2 Minimizing the sup–norm and linear programming

Instead of minimizing "average" properties of a certain derivative, we can also use as criterion the minimization of the **global maximum** of a certain derivative, that is,

$$\min_{\boldsymbol{d}} \left\| S_m^{(k)} \boldsymbol{d} \right\|_\infty = \min_{\boldsymbol{d}} \max_{t\in[t_{m+1},t_{n+1}]} \left| S_m^{(k)} \boldsymbol{d} \right|, \qquad \boldsymbol{d}\boldsymbol{N}_m(X) = \boldsymbol{y}.$$

Let us consider this problem first for $k = m-1$, i.e., the maximal order of differentiation[12] since then the spline curve

$$S_m^{(m-1)} \boldsymbol{d} = \boldsymbol{d}\, \boldsymbol{N}_m^{(m-1)} = \boldsymbol{d}\widehat{\boldsymbol{G}}_{m-1}\, \boldsymbol{N}_1\left(\cdot \,|\, \widehat{T}\right)$$

is a **piecewise linear curve** which takes its maximum at one of the knots of $\widehat{T}$ and the value there is the respective control point. As coefficient norm on $\mathbb{R}^d$ we also choose the ∞–norm and define

$$v := \max_{t\in[t_{m+1},t_{n+1}]} \left\| S_m^{(m-1)} \boldsymbol{d} \right\|_\infty = \max_{j=1,\ldots,n-m+1} \left\| \left(\boldsymbol{d}\widehat{\boldsymbol{G}}_{m-1} \right)_j \right\|_\infty .$$

Thus, v is the smallest positive number such that

$$\left| \left(\boldsymbol{d}\widehat{\boldsymbol{G}}_{m-1} \right)_j \right| \le v\,\mathbf{1}, \qquad j = 1,\ldots,n-m+1,$$

that is,

$$-v\mathbf{1} \le \left(\boldsymbol{d}\widehat{\boldsymbol{G}}_{m-1} \right)_j \le v\mathbf{1} \qquad \text{or} \qquad \pm \left(\boldsymbol{d}\widehat{\boldsymbol{G}}_{m-1} \right)_j \le v\mathbf{1}.$$

Using the **slack variables** $\boldsymbol{s}_{j,+}, \boldsymbol{s}_{j,-} \in \mathbb{R}^d_+$, we thus get the identities

$$\pm \left(\boldsymbol{d}\widehat{\boldsymbol{G}}_{m-1} \right)_j - v\mathbf{1} + \boldsymbol{s}_{j,\pm} = 0, \qquad j = 1,\ldots,n-m+1, \tag{5.5}$$

which have to be satisfied together with $\boldsymbol{d}\,\boldsymbol{N}_m(X) = \boldsymbol{y}$. With the notation

$$\boldsymbol{s}_\pm = \left[\boldsymbol{s}_j^\pm \;:\; j = 1,\ldots,n-m+1 \right],$$

the equations (5.5) take the matrix form

$$\pm \boldsymbol{d}\widehat{\boldsymbol{G}}_{m-1} - v \underbrace{\mathbf{1}_d \mathbf{1}_{n-m+1}^T}_{=:\mathbf{1}_{d\times n-m+1}} + \boldsymbol{s}_\pm = \mathbf{0}.$$

Then the solution of the linear program

$$\min_{\boldsymbol{d},v,\boldsymbol{s}^\pm} v, \qquad \begin{bmatrix} \boldsymbol{N}_m^T(X) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \widehat{\boldsymbol{G}}_{m-1}^T & -\mathbf{1}_{n-m+1\times d} & \boldsymbol{I} & \mathbf{0} \\ -\widehat{\boldsymbol{G}}_{m-1}^T & -\mathbf{1}_{n-m+1\times d} & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}^T \\ v \\ \boldsymbol{s}_+^T \\ \boldsymbol{s}_-^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}^T \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{5.7}$$

gives the control polygon such that *globally* the respective spline curve has minimal norm of the $(m-1)$st derivative. There are however, several remarks to be made.

[12] Note that for $m = 3$ this problem corresponds to minimizing the largest value that $S_m'' \boldsymbol{d}$ can assume.

1. Since most implementations of the simplex algorithm can only handle *vector valued* right hand side in (5.7), the columns of the right–hand side matrix and the respective pieces of the "big" matrix have to be "stacked" on top of each other. This job is easily done by using the `reshape` command.

2. The form (5.7) is usually **not** proper to be entered directly into an implementation of the simplex algorithm as most of them use the implicit assumption that all variables have to be positive and there is no justification to assume that $\boldsymbol{d}$ has this property. However, any $\boldsymbol{d}$ can obviously be written as $\boldsymbol{d} = \boldsymbol{d}_+ - \boldsymbol{d}_-$ with $\boldsymbol{d}_\pm \geq \mathbf{0}$ so that formally we just have to introduce a new variable to be "on the safe side". Nevertheless, this approach is not recommended.

3. The unfortunate reason for this problem is that $\boldsymbol{d}$ is a **free variable** and is not bounded a priori, the bound for $\boldsymbol{d}$ just comes from the minimization. This "degeneracy" can cause trouble in many simple implementations of the simplex method which can be inappropriate for our slightly more sophisticated optimization problem.

4. Also note that (5.7) can **not** be considered *componentwise*, that is, each row of $\boldsymbol{d}$ would be determined by a scalar linear program as in (5.7) – the components are all coupled by the number v which is a maximum over **all** components.

5. If we limit each component by v_j, $j = 1, \ldots, d$, and minimize over $v_1 + \cdots + v_d$, then this corresponds to using the 1–norm as coefficient norm on $\mathbb{R}^d$.

6. General keep in mind that this approximation problems result in very **large** linear programs that may be hard to store and hard to solve. After all, the **worst case complexity** of a linear program with n variables is $O\left(2^n\right)$, though this happens only for very special and academic examples, and it is reported that in average and computational practice [16, 20] the complexity seems to be more on the $O(n)$–side. But there is no guarantee ...

But what to do in the case that $k < m - 1$? Here the maxima are not assumed at the knots, but nevertheless we can use the idea of (5.7) and just replace $\widehat{\boldsymbol{G}}_{m-1}$ by $\widehat{\boldsymbol{G}}_k$ there. What we minimize then is not the norm of the derivative but the norm of the largest coefficient of the derivative curve. However, since

$$\|S_m \boldsymbol{d}(x)\| \leq \sum_{j=1}^{n} \|\boldsymbol{d}_j\| \; N_j^m\left(x \,|\, T\right) \leq \max_{j=1,\ldots,n} \|\boldsymbol{d}_j\| \underbrace{\sum_{j=1}^{n} N_j^m\left(x \,|\, T\right)}_{=1} = \max_{j=1,\ldots,n} \|\boldsymbol{d}_j\|,$$

"small" control points also lead to globally "small" splines curves and so the target of minimization still make sense. But more important, this minimization problem can be solved by readily available efficient algorithms, in this case, the simplex method.

5.3 Least squares and energy functionals

The minimizing approach from the preceding subsection still insisted on interpolation. Of course, there are situations where interpolation is a reasonable approach, but in many real world application at least one of the following two situations occurs:

1. The data can be contaminated by noise, so that interpolation always interpolates the noise as well, causing, for example, unnecessary oscillation in the spline.
2. Interpolation may not even be necessary and it would be sufficient to reproduce the data with a certain tolerance. Hence, this tolerance could be used to obtain "smoother" solutions of the approximation problem.

The **smoothing splines** is defined as the solution of the optimization problem

$$\min_{\boldsymbol{f}\in\mathbb{S}_m(T)} \sum_{x\in X} \|\boldsymbol{f}\,(x) - \boldsymbol{y}_x\|_2^2 + \lambda \int_{\mathbb{R}} \|\boldsymbol{f}''(t)\|_2^2 \, dt \tag{5.8}$$

for a parameter $\lambda > 0$ to be chosen appropriately.

The idea in the smoothing spline is to "balance" accuracy to the data against smoothness (in the "geometric" sense of little oscillation) with the help of the parameter λ.

Of course, there are a lot of things that can be varied in (5.8):

1. One could pick a smoothness measure different from $|\cdot|_2$, the integral over the second derivative, for example integrate over higher order derivatives or even linear combinations of derivatives of different orders.
2. Both norms, the one in the "approximation term" as well as the one in the "smoothness term", can also be endowed with weights.

A fairly general form of the smoothing spline problem can be stated with **weights** $w_x \geq 0$, $x \in X$, and parameters[13] $\lambda_1, \ldots, \lambda_{m-1}$ as

$$\min_{f\in\mathbb{S}_m(T)} \sum_{x\in X} w_x \left|f(x) - y_x\right|^2 + \sum_{r=1}^{m-1} \lambda_r \int_{\mathbb{R}} \left|f^{(r)}(x)\right|^2 \, dx. \tag{5.9}$$

To determine the solution of this problem, we first write use the (row) vectors

$$f(X) = [f(x) \ : \ x \in X] = \boldsymbol{d}\,\boldsymbol{N}_m(x) \qquad \text{and} \qquad \boldsymbol{y} = [y_x \ : \ x \in X]$$

[13] Why at most $m-1$ such terms? They will correspond to derivatives and a spline of order m can only guarantee at most $m-1$ continuous derivatives.

and note that, with $\boldsymbol{W} = \text{diag}\,[w_x \,:\, x \in X]$,

$$\begin{aligned}\sum_{x\in X} w_x \left|f(x) - y_x\right|^2 &= \left(\boldsymbol{d}\boldsymbol{N}_m(X) - \boldsymbol{y}\right)\boldsymbol{W}\left(\boldsymbol{d}\boldsymbol{N}_m(X) - \boldsymbol{y}\right)^T \\ &= \boldsymbol{d}\boldsymbol{N}_m(X)\boldsymbol{W}\boldsymbol{N}_m^T(X)\boldsymbol{d}^T - 2\,\boldsymbol{d}\boldsymbol{N}_m(X)\boldsymbol{W}\boldsymbol{y}^T + \boldsymbol{y}^T\boldsymbol{W}\boldsymbol{y}\end{aligned}$$

as well as, recalling (5.4)

$$\int_{\mathbb{R}} \left|f^{(r)}(x)\right|^2 dx = \boldsymbol{d}\boldsymbol{G}_r\boldsymbol{B}_{m-r}\boldsymbol{G}_r^T\boldsymbol{d}^T.$$

Consequently, the functional to be minimized in terms of the coefficient vector $\boldsymbol{d}$ takes the form

$$\Phi_\lambda(\boldsymbol{d}) = \boldsymbol{d}\left(\boldsymbol{N}_m(X)\boldsymbol{W}\boldsymbol{N}_m^T(X) + \sum_{r=1}^{m-1}\boldsymbol{G}_r\boldsymbol{B}_{m-r}\boldsymbol{G}_r^T\right)\boldsymbol{d}^T - 2\,\boldsymbol{d}\boldsymbol{N}_m(X)\boldsymbol{W}\boldsymbol{y}^T + \boldsymbol{y}^T\boldsymbol{W}\boldsymbol{y}$$

which is a quadratic form with positive semidefinite[14] quadratic coefficient whose minimum can be located by solving $\nabla_{\boldsymbol{d}}\Phi_\lambda(\boldsymbol{d}) = 0$, that is,

$$\left(\boldsymbol{N}_m(X)\boldsymbol{W}\boldsymbol{N}_m^T(X) + \sum_{r=1}^{m-1}\lambda_r\boldsymbol{G}_r\boldsymbol{B}_{m-r}\boldsymbol{G}_r^T\right)\boldsymbol{d}^T = \boldsymbol{N}_m(X)\boldsymbol{W}\boldsymbol{y}^T. \tag{5.10}$$

As we will see later, these normal equations take a rather nice form for splines – the matrices appearing on the right hand side are banded ones and permit a solution in $O(n)$.

Note that for smoothing splines there is no connection needed any more between the dimension of the spline space and the number of "interpolation" conditions:

1. If the dimension of the spline space is too small, however, then this may result in a severe defect in approximation, i.e. the approximation term will always be rather large and may even dominate the smoothness term.

2. If the dimension of the spline space is large enough or maybe too large, then we will obtain an interpolating or at least "almost interpolating" function for small parameters of λ.

It is clear that there is a large number of degrees of freedom, from choosing the dimension of the spline space, the location of the sites X up to the choice of the parameters λ. For the latter ones there exists the concept of **cross validation** which fixes the parameters such that they are optimal in a certain statistical context.

In **fitting problems** where a smoothest possible spline is to be found which is "sufficiently close" to given data, a different strategy can be applied:

[14] And in most cases strictly positive definite.

1. Choose the spline space large enough to enable interpolation[15].
2. Solve the problem for $\lambda = 0$, thus computing an interpolant whose coefficients even satisfy $\Phi_0(\boldsymbol{d}) = 0$.
3. Enlarge λ to improve the smoothness of the function as long as the approximation property is still maintained.

This procedure leads to a choice of λ which gives the "smoothest" approximant within a given tolerance.

5.4 Efficient implementation of Gramians

Finally, there is the need to compute the B–spline Gramian

$$\boldsymbol{B}_m(T) = \left[\int_{\mathbb{R}} N_j^m(x)\, N_k^m(x)\, dx \;:\; \begin{array}{l} j = 1, \ldots, n \\ k = 1, \ldots, n \end{array} \right] \tag{5.11}$$

and we better do this in an efficient way. The matrix is obviously symmetric, so it suffices to compute the entries of the upper triagonal part, i.e., for $j \le k$. Moreover, since N_j^m is supported on $[t_j, t_{j+m+1}]$ and N_k^m on $[t_k, t_{k+m+1}]$, respectively, the integral is nonzero if and only if $t_k \in [t_j, t_{j+m}]$, hence if and only if $k \le j + m$. In other words, all we need to compute for given $j \in \{1, \ldots, n\}$ are the integrals

$$\int_{\mathbb{R}} N_j^m(x)\, N_k^m(x)\, dx = \int_{t_j}^{t_{j+m}} N_j^m(x)\, N_k^m(x)\, dx, \qquad k = j, \ldots, j+m.$$

This already gives us an important but not too surprising observation:

> The Gramian $\boldsymbol{B}_m(T)$ is a symmetric banded matrix with m sub- and superdiagonals.

This method is implemented in the function `BSplGram`. To obtain the Gramian of a derivative, in other words, the matrix

$$\boldsymbol{B}_m^{(r)} := \int_{\mathbb{R}} \boldsymbol{N}_m^{(r)}(x) \boldsymbol{N}_m^{(r)}(x)^T\, dx = \left[\int_{\mathbb{R}} N_j^m(x)^{(r)}\, N_k^m(x)^{(r)}\, dx \;:\; \begin{array}{l} j = 1, \ldots, n \\ k = 1, \ldots, n \end{array} \right],$$

we substitute (2.3) and obtain that

$$\begin{aligned} \boldsymbol{B}_m^{(r)} &= \int_{\mathbb{R}} \widehat{\boldsymbol{G}}_r\, \boldsymbol{N}_m^{(r)}\left(x \,|\, \widehat{T}_k\right) \boldsymbol{N}_m^{(r)}\left(x \,|\, \widehat{T}_k\right)^T \widehat{\boldsymbol{G}}_r^T\, dx \\ &= \widehat{\boldsymbol{G}}_r \left(\int_{\mathbb{R}} \boldsymbol{N}_m^{(r)}\left(x \,|\, \widehat{T}_k\right) \boldsymbol{N}_m^{(r)}\left(x \,|\, \widehat{T}_k\right)^T dx \right) \widehat{\boldsymbol{G}}_r^T = \widehat{\boldsymbol{G}}_r\, \boldsymbol{B}_{m-r}\left(\widehat{T}_k\right) \widehat{\boldsymbol{G}}_r^T, \end{aligned}$$

which is the most economic way to compute this matrix. Note that $\boldsymbol{B}_{m-r}\left(\widehat{T}_k\right) \in \mathbb{R}^{n-r \times n-r}$ which nicely explains the loss of rank that occurs when passing to derivatives.

[15] It is no mistake that the word "unique" is missing here – we do not care for a unique interpolant, any interpolant will do!

5.5 An example of smoothing splines

Let us finally consider an example, the "standard" case where smoothness is considered in terms of second derivatives and we use the Greville abscissae with respect to T as approximation sites. So, we begin by

```
octave> T = [ 0 0 0 ( 0:10 ) 10 10 10 ];X = Greville( 3,T );
```

and consider smoothing of a randomly perturbed linear function:

```
octave> y = X .* ( 1 + .4*( rand( size(X) ) .- 1 ) );
```

The perturbation is a relative error of at most 20%. To solve, for given $\lambda > 0$, the minimization problem (with identical weights), we first compute the two matrices

```
octave> A = BSplVander( 3,T,X ); A = A*A';
octave> B = BSplGramDer( 3,T,2 );
```

as well as the right hand side

```
octave> yy = BSplVander( 3,T,X ) * y';
```

We plot `y`

```
octave> clearplot; plot( X,y,"*" )
```

define a point set for plotting the splines,

```
octave> Z = (min(T):.01:max(T));
```

and compute and plot the smoothing splines are computed by calling

```
octave> l = 0; d = ( ( A + l*B ) \ yy )';
octave> s = d*BSplVander( 3,T,Z ); plot( Z,s );
```

The effect of increasing the smoothing parameter can be nicely seen in Fig. 5.1. The deviation from the data increases, but the smoothness of the spline, in the sense of reduced oscillation, is increased in exchange. One remarkable property of this process is as follows

> The solution for $\lambda = 0$ of this optimization problem is the good old **natural cubic spline**. The solution for $\lambda = \infty$, on the other hand, is the **linear regression**, i.e., the unique line that has minimal least squares distance from the data.

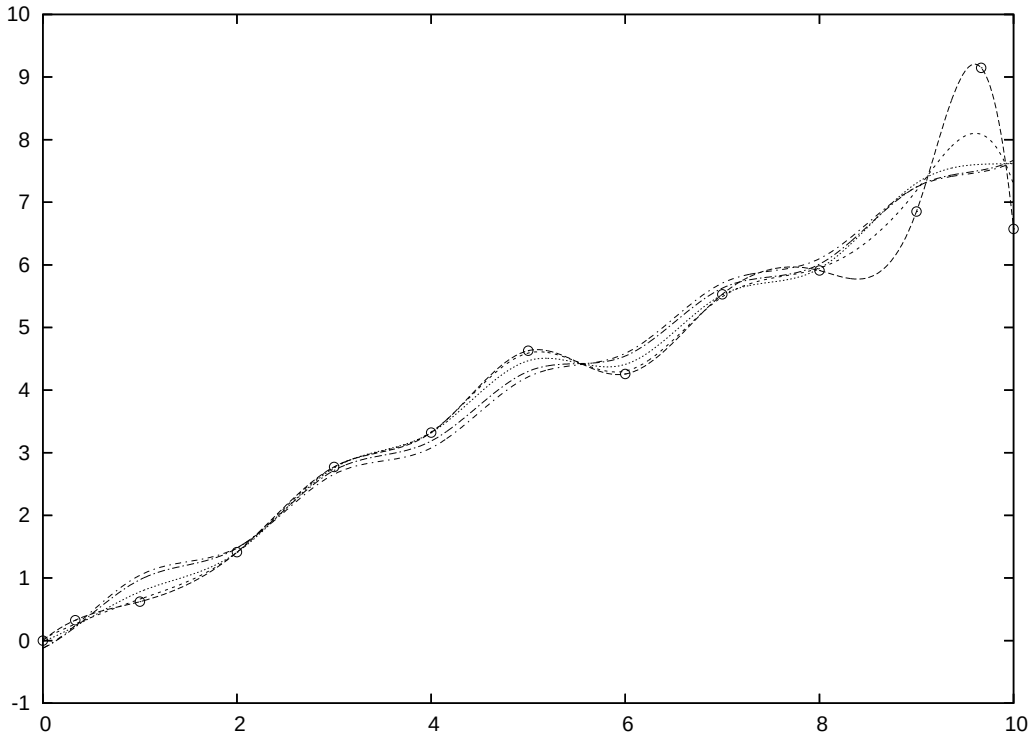

Figure 5.1: The smoothing splines for the parameter values $\lambda = 0, 0.01, 0.1, 1, 10$.

5.6 A sup–smoothing spline

We could also combine the idea of the smoothing spline

> Balance approximation at the sites with a smoothness term

with the minimization with respect to the ∞–norm which will[16] lead once more to a linear program. So, we now consider the problem

$$\min_{\boldsymbol{d}} \left(\max_{x \in X} \|S_m \boldsymbol{d}(x) - \boldsymbol{y}_x\| + \lambda \max_{j=1,\ldots,n-k} \left\| \left(\boldsymbol{d}\, \widehat{\boldsymbol{G}}_k \right)_j \right\| \right). \tag{5.12}$$

We already know how to change a minimax problem into a linear program, see (5.7), and the side conditions coming from the smoothing term are almost like there, except that we drop the interpolation term:

$$\begin{bmatrix} \widehat{\boldsymbol{G}}_k^T & -\mathbf{1}_{n-k\times d} & \boldsymbol{I} & \mathbf{0} \\ -\widehat{\boldsymbol{G}}_k^T & -\mathbf{1}_{n-k\times d} & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}^T \\ v \\ \boldsymbol{s}_+^T \\ \boldsymbol{s}_-^T \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{5.13}$$

[16] This is no surprise – at least it should not be!

To handle the first, the "approximation", term, we also proceed as above, set

$$u = \max_{x\in X} \|S_m \boldsymbol{d}(x) - \boldsymbol{y}_x\| = \max_{x\in X} \|\boldsymbol{d}\, \boldsymbol{N}_m(x) - \boldsymbol{y}_x\|\,,$$

hence, with additional slack variables $\boldsymbol{t}_\pm$,

$$\pm \boldsymbol{d}\, \boldsymbol{N}_m(X) - u\mathbf{1} + \boldsymbol{t}_{x,\pm} = \pm \boldsymbol{y}_x, \qquad x \in X,$$

giving

$$\begin{bmatrix} \boldsymbol{N}_m^T(X) & -\mathbf{1}_{\#X\times d} & \boldsymbol{I} & \mathbf{0} \\ -\boldsymbol{N}_m^T & -\mathbf{1}_{\#X\times d} & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}^T \\ u \\ \boldsymbol{t}_+^T \\ \boldsymbol{t}_-^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}^T \\ -\boldsymbol{y}^T \end{bmatrix}. \tag{5.14}$$

Dropping the subscript of the $\mathbf{1}$–matrices[17], we thus find that the sup–smoothing spline solving (5.12) is the solution of the linear program

$$\min u + \lambda v, \qquad \begin{bmatrix} \boldsymbol{N}_m^T(X) & -\mathbf{1} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{N}_m^T & -\mathbf{1} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \widehat{\boldsymbol{G}}_k^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{1} & \boldsymbol{I} & \mathbf{0} \\ -\widehat{\boldsymbol{G}}_k^T & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{1} & \mathbf{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}^T \\ u \\ \boldsymbol{t}_+^T \\ \boldsymbol{t}_-^T \\ v \\ \boldsymbol{s}_+^T \\ \boldsymbol{s}_-^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}^T \\ -\boldsymbol{y}^T \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{5.16}$$

In a linear programming environment with a normal form

$$\min c^T x, \qquad Ax \le b,$$

we can drop the slack variable $\boldsymbol{s}_\pm$ and $\boldsymbol{t}_\pm$ and just consider

$$\min u + \lambda v, \qquad \begin{bmatrix} \boldsymbol{N}_m^T(X) & -\mathbf{1} & \mathbf{0} \\ -\boldsymbol{N}_m^T(X) & -\mathbf{1} & \mathbf{0} \\ \widehat{\boldsymbol{G}}_k^T & \mathbf{0} & -\mathbf{1} \\ -\widehat{\boldsymbol{G}}_k^T & \mathbf{0} & -\mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{d} \\ u \\ v \end{bmatrix} \le \begin{bmatrix} \boldsymbol{y}^T \\ -\boldsymbol{y}^T \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

Hence, in principle it is possible to solve this problem numerically, however, the remarks made after (5.7) still haven't lost any of their validity and it will be necessary to really test the performance and quality of this method on real–world data. What remains unchanged, however, is the fact that the problem still contains $\boldsymbol{d}$ as a **free variable**. We compare the behavior of the smoothing splines with respect to the two different norms in Fig 5.2. Apparently, the sup–smoothing spline has a much stronger tendency to trade in deviation from the data points against

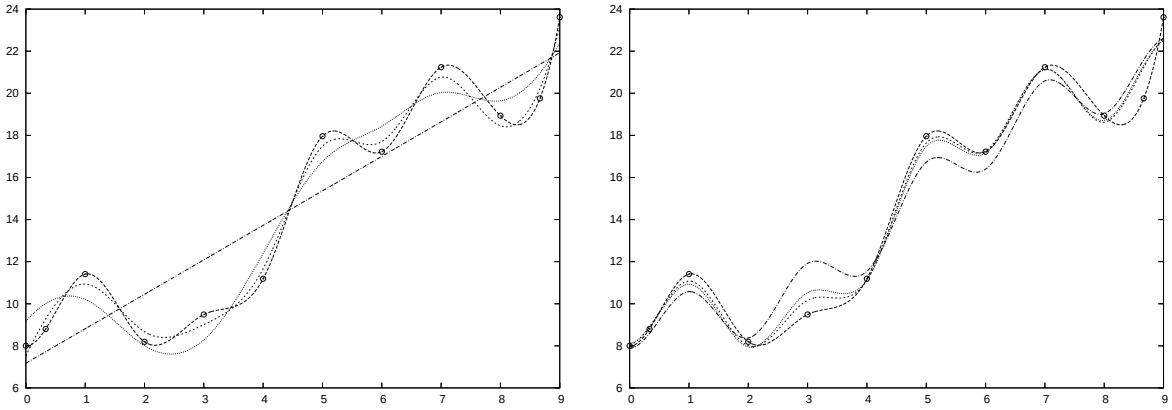

Figure 5.2: The smoothing splines for $p = \infty$ *(left)* and $p = 2$ *(right)* and the parameter values $\lambda = 0, 0.05, 0.1, 1$. The parameter values are not really comparable in these two situations but it seems that the sup–smoothing spline has a general tendency to emphasize the smoothness part more strongly.

smoothness of the curve – once more smoothness in the sense of reduced oscillation. Another example is shown in Fig. 5.3, this time for a sampled absolute value function. Again the effect of the smoothing parameter can be seen nicely and of course the limit is not the line which minimizes the maximal distance to the data.

[17] They should be clear from the context now!

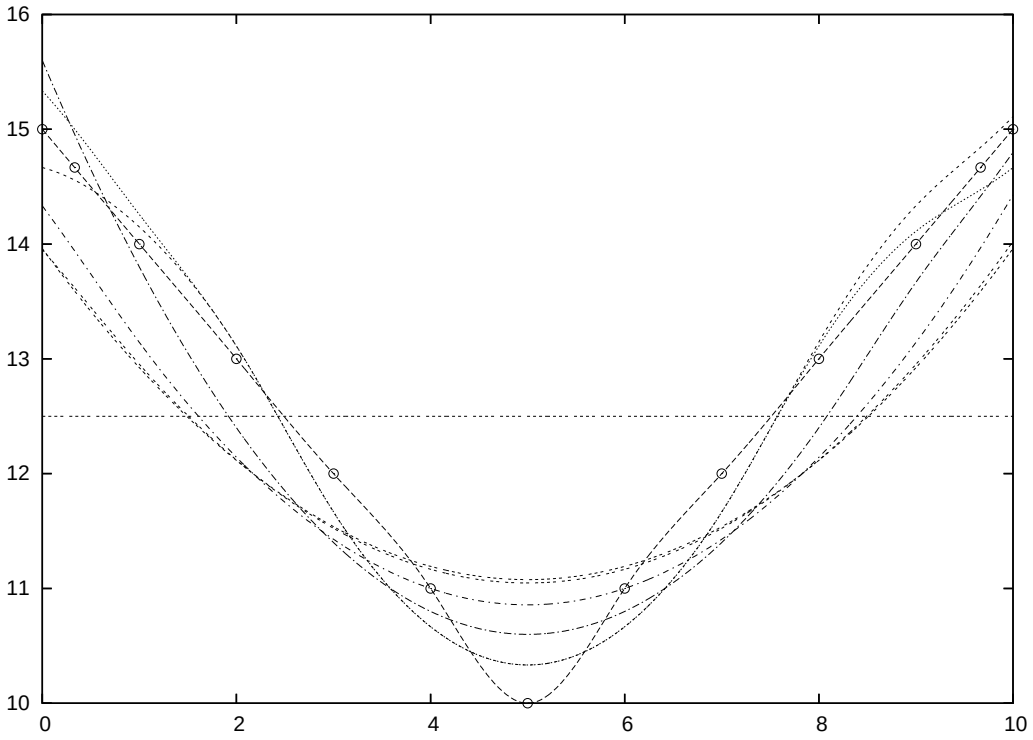

Figure 5.3: Approximating a sampled absolute value function with the sup–smoothing spline for the parameter values $\lambda = 0, .5, 1, 2, 3, 4, 5, 10$.

Although this may seen as a paradox, all exact science is dominated by the idea of approximation

Bertrand Russell

6 Knot insertion and removal

The Curry–Schoenberg theorem tells us that a spline is a piecewise polynomial function of a given order of smoothness at the knots, depending on the multiplicity of the knot. Trivially, a polynomial is a also a piecewise polynomial – with the little side effect that the connection between the polynomial pieces in this case is a C^m or a C^∞ one; there is no more distinction between the two of them for polynomials in Π_m. In other words:

If we insert knots into a given knot sequence, then any spline with respect to the coarse knot sequence is also a spline with respect to the fine knot sequence.

6.1 Refinement of knot sequences

So what does it mean to refine a knot sequence? Intuitively, there must be two properties:

1. The refined knot sequence must contain any knot of the "original" knot sequence.
2. The multiplicities of each knot in the refined knot sequence must be at least as high as the multiplicity of the knot in the original knot sequence.

There are two ways to refine a knot sequence:

Insertion of a new knot: replace T by T^* where

$$T^* = \{t_1, \ldots, t_j, t^*, t_{j+1}, \ldots, t_{m+n+1}\}, \qquad t_j < t^* < t_{j+1}.$$

Increase the multiplicity: replace T by T^* where

$$T^* = \{t_1, \ldots, t_j, t_j, t_{j+1}, \ldots, t_{m+n+1}\}, \qquad t_j < t_{j+1}.$$

We will call $T^* = T^*_{m,n^*}$ a **refinement** of T, written as $T \subseteq T^*$, if T^* is again a valid[18] knot sequence of order m and if there exist a strictly increasing function $\nu : \{1, \ldots, n+m+1\} \to \{1, \ldots, n^*+m+1\}$ such that $t^*_{\nu(j)} = t_j$, $j = 1, \ldots, n+m+1$. Note that this definition is just a formalization of the intuitive criterion mentioned before.

One might argue whether it makes sense to refine a knot sequence "outside" the boundary knots. The above definition does not exclude this, but it also does not encourage such a strategy!

Clearly, any piecewise polynomial on T is also a piecewise polynomial on T^* and since the multiplicities of knots can only be increased when passing to refined knot sequence, the respective smoothness conditions become *relaxed*. Therefore, we have the following observation:

If $T \subseteq T^*$ then also $\mathbb{S}_m(T) \subseteq \mathbb{S}_m(T^*)$.

The B–splines with respect to T^* are a basis of $\mathbb{S}_m(T^*)$, which contains $\mathbb{S}_m(T)$, and so there must be, for any coefficient vector $\boldsymbol{d}$, an associated coefficient vector $\boldsymbol{d}^*$ such that

$$\sum_{j=1}^{n} \boldsymbol{d}_j \, N_j^m(\cdot \,|\, T) = \sum_{j=1}^{n^*} \boldsymbol{d}_j^* \, N_j^m(\cdot \,|\, T^*)$$

and of course the question is how to compute $\boldsymbol{d}^*$.

6.2 Knot insertion

We will focus here on the method to insert a *single knot*, usually referred to as the **Boehm algorithm** – in contrast to this one, there also exists the **Oslo algorithm** which inserts an arbitrary number of knots *simultaneously*.
We consider the following situation:

Insert a knot t^* between two distinct knots $t_j < t_{j+1}$ such that $t_j \le t^* < t_{j+1}$. That is, we increase the multiplicity of a knot "from the right".

Our first observation is that knot insertion is and must be a *local* process, it only affects those B–splines whose support contains the interval $[t_j, t_{j+1}]$, and these are precisely the B–Splines $N^m_{j-m}, \ldots, N^m_j$. That also means that only the associated control points $\boldsymbol{d}_{j-m}, \ldots, \boldsymbol{d}_j$ will be relevant for the knot insertion algorithm. Moreover, insertion of a single knot will result in $n^* = n+1$, so that $\boldsymbol{d}^*$ contains exactly one coefficient more than $\boldsymbol{d}$.

Algorithm 6.1 (Knot insertion)
Input:

[18] This means that there still a no knots whose multiplicity exceeds $m+1$.

- *Knot sequence* $T = T_{m,n}$.
- *Knot to be inserted* $t_j \leq t^* < t_{j+1}$.

Procedure:

1. *For* $k = 1, \ldots, j - m$ *set*
$$\boldsymbol{d}_k^* = \boldsymbol{d}_k.$$
2. *For* $k = j - m + 1, \ldots, j$
   (a) *Compute*
   $$\alpha_k = \frac{t_{k+m} - t^*}{t_{k+m} - t_k}$$
   (b) *Set*
   $$\boldsymbol{d}_k^* = \alpha_k \, \boldsymbol{d}_{k-1} + (1 - \alpha_k) \, \boldsymbol{d}_k.$$
3. *For* $k = j + 1, \ldots, n + 1$ *set*
$$\boldsymbol{d}_k^* = \boldsymbol{d}_{k-1}.$$

Result: *Control points* $\boldsymbol{d}^*$ *such that*

$$S_m \boldsymbol{d}^* \left(\cdot \,|\, T^* \right) = S_m \boldsymbol{d} \left(\cdot \,|\, T \right).$$

Once more, we can express this process in terms of linear algebra, namely as a matrix multiplication

$$\boldsymbol{d}^* = \boldsymbol{d}\boldsymbol{A}\left(t^*\right), \qquad \boldsymbol{A}\left(t^*\right) = \begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & \alpha_{j-m+1} & & & & & \\ & & & 1 - \alpha_{j-m+1} & \ddots & & & & \\ & & & & \ddots & \alpha_j & & & \\ & & & & & 1 - \alpha_j & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{bmatrix} \in \mathbb{R}^{n \times n+1}.$$

Note that $\boldsymbol{A}\left(t^*\right)$ is a very sparse, banded matrix. The procedure to compute such a knot insertion matrix is named `KinsMat`, the pair $\boldsymbol{A}, T^*$ by the function `KInsert`. Let us reconsider our earlier example, namely

```
octave> T = [ 0,0,0,0,1,2,3,4,4,4,4 ];
octave> d = [ 0 1 1 .5 0 0 1 ; 0 .4 .8 1 .8 .4 0 ];
```

and insert the knot $t^* = 2.5$. The new control polygon can be determined as

```
octave> [A,Tt] = KInsert( 3,T,2.5 ); dd = d * A;
```

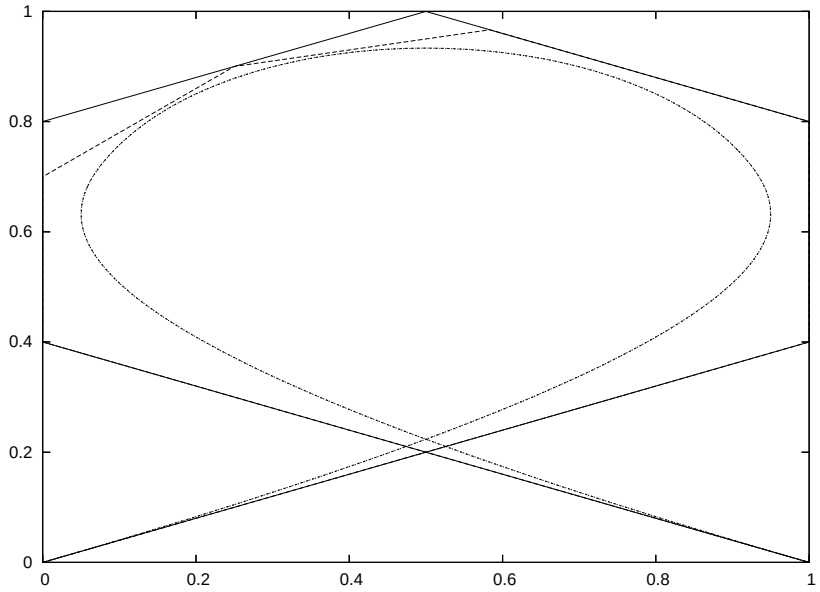

Figure 6.1: Knot insertion. The two control polygons and the two splines are drawn, but of course only one spline curve is visible as knot insertion does not change the curve.

The result of single knot insertion is shown in Fig. 6.1. Since knot insertion is again a **convex combination** process, the new control polygon always lies inside the convex hull of the original one.

Knot insertion can be repeated. For example, we can insert a knot m times and since spline functions always *interpolate*[19] at knots of multiplicity m, this gives a way to *evaluate* a spline function at the newly inserted knot – which happens to be the de Boor algorithm if looked at carefully.

```
octave> d = [ 0 1 1 .8 .2 0 0 1 ; 0 .2 .6 1 1 .6 .2 0 ];
octave> T = [ 0,0,0,0,1,2,3,4,5,5,5,5 ];
octave> for j=1:3 [ A,T ] = KInsert( 3,T,2.5 ); d = d*A; end
```

Then one of the control points is indeed located on the curve. This can be seen in Fig 6.2 where the outcome of triple knot insertion for a cubic spline is depicted.

There is a nice side effect of our "linear algebra" approach to do knot insertion, it allows us to do **virtual knot insertion**: instead of modifying the control polygon and the knot sequence, we just store the matrix

$$\boldsymbol{A}\left(t_1^*,\ldots,t_k^*\right) = \boldsymbol{A}\left(t_k^*\right)\cdots\boldsymbol{A}\left(t_1^*\right)$$

[19] We did not mention this property before which follows quite immediately from the de Boor algorithm, because at knots of multiplicity m one always "sits" on the left boundary of all the intervals to be considered, so that the coefficient $\boldsymbol{d}_{j-m}$ is reproduced in all stages of the algorithm.

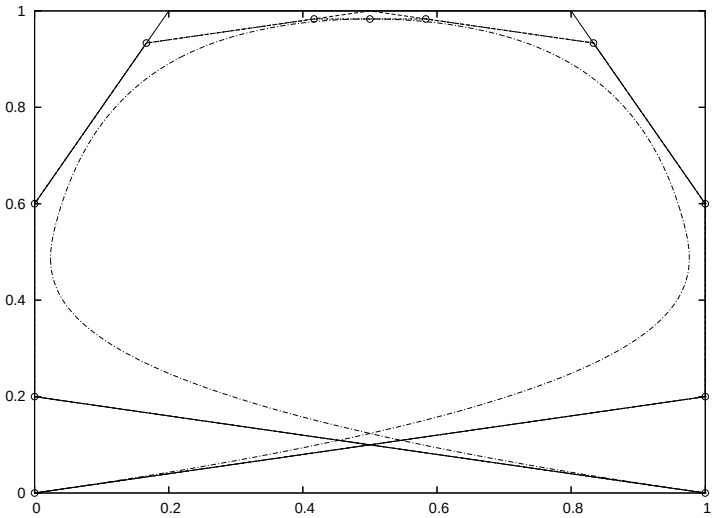

Figure 6.2: An example of triple knot insertion with the intermediate control polygons and the control points of final level. One of them now lies on the curve where the triple knot has been inserted.

that defines the action of knot insertion; note, however, that $\boldsymbol{A}\left(t_j^*\right)$ depends on the knot sequence $T \cup \left\{t_1^*, \ldots, t_{j-1}^*\right\}$ so that there is no commuting of the knot insertion matrices or something similar.

6.3 First applications of knot insertion

Knot insertion has more use than just improving the flexibility of a spline curve by adding further control points, though of course this has been the main intention in and for the development of knot insertion algorithms.

We begin by the conversion of a spline curve into **piecewise polynomial form**, often simply called the PP form of the spline curve. The polynomial pieces could be represented with respect to the monomial basis, but since that one is not of geometric relevance, we might as well reference to the **Bézier representation** of a polynomial curve on the nondegenerate interval $[t_j, t_{j+1}]$:

$$\boldsymbol{p}(x) = B_n \boldsymbol{d}(x) = \sum_{j=0}^{n} \boldsymbol{d} \binom{n}{j} \lambda^j(x) \left(1 - \lambda(x)\right)^{n-j}, \qquad \lambda(x) = \frac{x - t_j}{t_{j+1} - t_j}.$$

The Bézier representation of a polynomial on $[a, b]$ corresponds to a most simple spline function, namely the one with knots of multiplicity $m + 1$ at a and b and nowhere else. So, if we want to compute the restriction of a spline curve to an interval $[a, b]$, $t_j \leq a < b \leq t_{j+1}$, then this can be obtained quite easily:

The restriction of a spline curve to a given interval whose interior does not contain a knot can be obtained by inserting the endpoints of the interval into the knot sequence until they are knots of multiplicity $m+1$.

Clearly, the knot insertion only gives the coefficients of the Bézier representation of the polynomial, but these coefficients can be easily converted into coefficients with respect to a **monomial representation**. There are some advantages of a piecewise polynomial representation:

- NC–milling machines "understand" polynomials, so the coefficients (of degree up to 5) can be fed into an NC machine directly.
- The local polynomial evaluation can be done with a complexity of $O(m)$ while the de Boor algorithm or also the spline recurrences have the higher complexity order of $O\left(m^2\right)$. Hence, if a spline has to be evaluated at many points, such a conversion could be worthwhile.

Note that `Matlab`'s spline toolbox even has a built–in conversion routine for determining the piecewise polynomial form, see [2].

The next application is the **comparison of spline curves**. Here, we consider two spline curves,

$$S_m\boldsymbol{d}\left(\cdot \,|\, T\right) = \sum_{j=1}^{n} \boldsymbol{d}_j \, N_j^m\left(\cdot \,|\, T\right) \qquad \text{and} \qquad S_m\boldsymbol{d}'\left(\cdot \,|\, T'\right) = \sum_{j=1}^{n'} \boldsymbol{d}'_j \, N_j^m\left(\cdot \,|\, T'\right).$$

Let $T^* = T \cup T'$ be the union of the two knot sequences, i.e., the *smallest* knot sequence such that $T \subseteq T^*$ and $T' \subset T^*$, then we can insert the "missing" knots into both sequences and write

$$S_m\boldsymbol{d}\left(\cdot \,|\, T\right) = S_m\boldsymbol{d}\boldsymbol{A}\left(T^* \setminus T\right)\left(\cdot \,|\, T^*\right)$$

as well as

$$S_m\boldsymbol{d}'\left(\cdot \,|\, T'\right) = S_m\boldsymbol{d}'\boldsymbol{A}\left(T^* \setminus T'\right)\left(\cdot \,|\, T^*\right)$$

so that the difference between the two splines is estimated by

$$\left\|\boldsymbol{d}\boldsymbol{A}\left(T^* \setminus T\right) - \boldsymbol{d}'\boldsymbol{A}\left(T^* \setminus T'\right)\right\|$$

where as norms we could either take the **Frobenius norm**

$$\left\|\boldsymbol{d}\boldsymbol{A}\left(T^* \setminus T\right) - \boldsymbol{d}'\boldsymbol{A}\left(T^* \setminus T'\right)\right\|_F = \left(\sum_{j=1}^{d}\sum_{k=1}^{n^*}\left|\left(\boldsymbol{d}\boldsymbol{A}\left(T^* \setminus T\right)\right)_{j,k} - \left(\boldsymbol{d}'\boldsymbol{A}\left(T^* \setminus T'\right)\right)_{j,k}\right|^2\right)^{1/2} \tag{6.1}$$

or the **vector sup–norm**

$$\left\|\boldsymbol{d}\boldsymbol{A}\left(T^* \setminus T\right) - \boldsymbol{d}'\boldsymbol{A}\left(T^* \setminus T'\right)\right\|_\infty = \max_{j=1,\ldots,d}\max_{k=1,\ldots,n^*}\left|\left(\boldsymbol{d}\boldsymbol{A}\left(T^* \setminus T\right)\right)_{j,k} - \left(\boldsymbol{d}'\boldsymbol{A}\left(T^* \setminus T'\right)\right)_{j,k}\right|. \tag{6.2}$$

Therefore, we can approximate a given spline $S_m\boldsymbol{d}'(\cdot\,|\,T')$ by a spline of the form $S_m\boldsymbol{d}(\cdot\,|\,T)$ with a possibly different knot sequence[20] by once more solving one of our smoothing problems like either

$$\min_{\boldsymbol{d}} \|\boldsymbol{d}\boldsymbol{A}(T^*\setminus T)-\boldsymbol{d}'\boldsymbol{A}(T^*\setminus T')\|_F^2+\sum_{r=1}^{m-1}\lambda_r\int\left\|S_m^{(r)}\boldsymbol{d}\right\|^2 \tag{6.3}$$

or

$$\min_{\boldsymbol{d}} \|\boldsymbol{d}\boldsymbol{A}(T^*\setminus T)-\boldsymbol{d}'\boldsymbol{A}(T^*\setminus T')\|_\infty+\sum_{r=1}^{m-1}\lambda_r\int\left\|\boldsymbol{d}\widehat{\boldsymbol{G}}_r\right\|_\infty \tag{6.4}$$

by the smoothing spline methods described in the previous chapters. Again, for sufficiently small values of λ, the approximants will come to the original spline as close as possible while for larger values of λ they will put more emphasis on reducing oscillation.

It is important to emphasize that the minimization is still performed in terms of $\boldsymbol{d}$ only – the knot insertion process is represented entirely by the matrix $\boldsymbol{A}(T^*\setminus T)$. So this is another case of **virtual knot insertion**.

6.4 Zeros of spline functions

A somewhat unexpected application of knot insertion is the determination of zeros, or, more generally, in the curves, the determination of **intersection** of a spline curve with a given line. In fact, this is essentially the problem of finding a zero of a spline curve: Let $\boldsymbol{a},\boldsymbol{b}\in\mathbb{R}^d$ the beginning and end point of the line in question and set

$$\boldsymbol{l}_j=\frac{t_{n+1}-x_j}{t_{n+1}-t_{m+1}}\boldsymbol{a}+\frac{x_j-t_{m+1}}{t_{n+1}-t_{m+1}}\boldsymbol{b},\qquad j=1,\ldots,n,$$

where the x_j are once more the **Greville abscissae**, then $S_m\boldsymbol{l}$ is a linear function[21] And thus the spline $S_m\boldsymbol{d}-S_m\boldsymbol{l}=S_m(\boldsymbol{d}-\boldsymbol{l})$ has a zero precisely at the points where $S_m\boldsymbol{d}$ intersects the line.

The standard method to compute **zeros of functions** is **Newton's method**, which determines a zero of f by means of the iteration

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)},\qquad k=0,1,2,\ldots,$$

subject to an initial value x_0. However, Newton's method has some problems here:

1. The iteration depends on a good initial value x_0 as it is well known to converge only locally. How to determine this value is usually difficult.

[20] For example, the knots in T can be placed with a density according to the curvature of the spline $S_m\boldsymbol{d}'$.

[21] More precisely, it is the Schoenberg operator applied to the linear function connecting $\boldsymbol{a}$ and $\boldsymbol{b}$.

2. To perform the iteration, the derivative of the spline has to be evaluated. This is in fact the smallest of our problems since the de Boor algorithm can easily be extended to provide this value too.

3. If we work with spline *curves*, we deal with *vector valued* functions so that the iteration above is not even well–defined[22]! The "self–suggesting" cure is to consider the curve component by component, but then we have to look for **simultaneous zeros** of these scalar functions, so in most cases a zero in one component may be rendered useless.

These difficulties can be overcome by a knot–insertion based algorithm due to Mørken and Reimers [15]. We will just describe the basic idea here, details and, in particular, a proof of[23] convergence can be found in [15]. In fact, the basic idea behind the method is really very intuitive:

> Intersect the control point with the line and insert the abscissa of this intersection into the knot sequence.

In fact, if there is **no** intersection between the control polygon and the line, then there is **no** intersection between the curve and the line[24] since the curve lies in the convex hull of the control polygon. Generally, the **variation diminishing** property of the splines, [12, 13] says that any hyperplane in $\mathbb{R}^d$ has more intersections with the control polygon than with the curve. Hence, if there is no intersection with the polygon, there is no intersection with the curve and we can immediately stop the search for zeros.

If, on the other hand, the control polygon intersects or has a zero somewhere, then this happens between two control points, say $\boldsymbol{d}_j$ and $\boldsymbol{d}_{j+1}$, more precisely, at the point

$$\lambda\, \boldsymbol{d}_j + (1-\lambda)\; \boldsymbol{d}_{j+1}, \qquad \lambda \in [0,1].$$

Motivated by the Schoenberg operator, we interpret $S_m \boldsymbol{d} = \mathscr{S}_m \boldsymbol{f}$ for some continuous function $\boldsymbol{f}$ and obtain that $\boldsymbol{d}_k = \boldsymbol{f}\,(x_k)$, so that it is natural to connect the control points to the Greville abscissae. Therefore, a good[25] guess for the zero is the point

$$t^* = \lambda\, x_j + (1-\lambda)\; x_{j+1},$$

which is precisely the knot to be inserted into T. This leads to a refined control polygon and the process is repeated until there exists a control which is either sufficiently close to the line or sufficiently small, depending on the type of problem one considers. In fact, the idea of iterating zeros of linearized objects appears very similar to Newton's method where a zero of the tangent is computed.

[22] How to divide a vector by a vector?
[23] Even quadratic
[24] For $d > 2$ one has to be slightly more careful here!
[25] Or, more correctly, a *linearized.*

An important application of zero finding is the determination of closest points: Given a point $\boldsymbol{y} \in \mathbb{R}^d$ and a spline curve $S_m\boldsymbol{d}$, defined in terms of its knot sequence and control polygon, find the closest point to $\boldsymbol{y}$ on the curve with respect to the euclidian distance, i.e., solve

$$\min_x \|S_m\boldsymbol{d}(x) - \boldsymbol{y}\|_2 .$$

Squaring the objective function once more we thus get that we can as well minimize

$$\begin{aligned} \|S_m\boldsymbol{d}(x) - \boldsymbol{y}\|_2^2 &= \sum_{j=1}^{d} \boldsymbol{d}_j \boldsymbol{N}_m(x) \boldsymbol{N}_m^T(x) \boldsymbol{d}_j^T - 2\boldsymbol{d}_j \boldsymbol{N}_m(x) y_j + y_j^2 \\ &= \sum_{j=1}^{d} \sum_{k,\ell=1}^{n} \boldsymbol{d}_{jk}\boldsymbol{d}_{j\ell} N_j^m(x) N_k^m(x) - 2\sum_{j=1}^{d} \sum_{k,\ell=1}^{n} \boldsymbol{d}_{jk} y_j N_k^m(x) + \sum_{j=1}^{d} y_j^2, \end{aligned}$$

where this time $\boldsymbol{d}_j$ denotes the jth *row* of the matrix $\boldsymbol{d}$ and thus the jth component of the curve. The function above is a scalar valued **spline function** of degree at most $2m$ with breakpoints at the knots of T, so it belongs to $\mathbb{S}_{2m}(T^*)$, where

$$T^* = \{t_1, \ldots, T, t_{n+m+1}, \ldots, t_{n+m+1}\},$$

so that just the coefficients of the boundary points are adapted. Consequently,

$$\|S_m\boldsymbol{d}(x) - \boldsymbol{y}\|_2^2 = \sum_{j=1}^{n} c_j \, N_j^{2m}(x \,|\, T^*) = \boldsymbol{c}\, \boldsymbol{N}_{2m}(x \,|\, T^*),$$

and to compute $\boldsymbol{c}$ we would need two operations:

1. a multiplication formula for splines,
2. a degree raising formula for splines,

but here we restrict ourselves to mentioning that both operations can be performed. To find the abscissa x of the closest point on the curve, we now "only" have to apply our zero search to the function

$$\frac{d}{dx} \|S_m\boldsymbol{d}(x) - \boldsymbol{y}\|_2^2 = \boldsymbol{c}\widehat{\boldsymbol{G}}_1 \, \boldsymbol{N}_{2m-1}\left(x \,|\, \widehat{T^*}\right).$$

6.5 Knot removal

Knot removal is the inverse process of knot insertion – instead to passing from T to $T^* \supset T$, we want to pass from T^* to T. Since we still have $\mathbb{S}_m(T) \subset \mathbb{S}_m(T^*)$, we cannot expect **knot removal** to be done exactly, but have to do it in such a way that the "knot removed" spline curve approximates the original one "as good as possible". So now we start with $S_m\boldsymbol{d}^*$ and want to determine $\boldsymbol{d}$ *after* removing t^* from T^*. To determine $\boldsymbol{d}$, we simply re–insert t^* and compare the resulting control polygon $\boldsymbol{d}\boldsymbol{A}(t^*)$ with $\boldsymbol{d}^*$, minimizing, for example, the expression

$$\|\boldsymbol{d}\boldsymbol{A}(t^*) - \boldsymbol{d}^*\|_2^2 = \boldsymbol{d}\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{d}^T - 2\boldsymbol{d}^*\boldsymbol{A}^T\boldsymbol{d}^T + \boldsymbol{d}^*(\boldsymbol{d}^*)^T$$

leading once more to the **normal equations**

$$\boldsymbol{d}\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{d}^*\boldsymbol{A}^T.$$

This, however, is only one way to determine the control polygon of a knot removed spline. Based on our knowledge from the smoothing splines, we can immediately come up with generalizations and extensions:

1. use a different norm, in particular $\|\boldsymbol{d}\boldsymbol{A}(t^*) - \boldsymbol{d}^*\|_\infty$. This will lead to another linear programming problem with all the pleasant things like free variables and so on.

2. minimize also with respect to a weighted norm that contains a **smoothing term** for $S_m\boldsymbol{d}$. After all, knot removal can be seen as passing from a "complicated" to a "simpler" curve and it may be beneficial if this curve is also as smooth as possible.

3. remove several knots at once – the matrix $\boldsymbol{A}$ is then a little bit more intricate.

Uns ist in alten mæren
wunders viel geseit
von Helden lobebæren
von grôzer arebeit

Das Nibelungenlied

A References

[1] M. Abramowitz and I. A. Stegun (eds.), *Handbook of mathematical functions*, Dover, 1972, 10th printing.

[2] C. de Boor, *A practical guide to splines*, Springer–Verlag, New York, 1978.

[3] C. de Boor, *Splinefunktionen*, Lectures in Mathematics, ETH Zürich, Birkhäuser, 1990.

[4] R. Christensen, *Advanced linear modeling*, 2. ed., Springer Texts in Statistics, Springer, 2001.

[5] G. B. Dantzig, *Linear programming and extensions*, Pinceton University Press, 1963.

[6] G. Farin, *Curves and surfaces for computer aided geometric design*, Academic Press, 1988.

[7] W. Gautschi, *Numerical analysis. an introduction*, Birkhäuser, 1997.

[8] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, John Wiley & Sons, 1966.

[9] S. Karlin, *Mathemtical methods and theory in games, programming and economics*, Dover Phoenix Editions, Addison–Wesley, 1959, Dover Reprint 2003.

[10] S. Mallat, *A wavelet tour of signal processing*, 2. ed., Academic Press, 1999.

[11] Th. Maresch, *Mathematik–Verknüpfung von 2d- und 3d–Punktwolken*, Ph.D. thesis, Justus–Liebig–Universität Gießen, 2006, Supervisor: T. Sauer.

[12] M. Marsden and I. J. Schoenberg, *On variation diminishing spline approximation methods*, Mathmatica **8** (1966), 61–82.

[13] _____, *On variation diminishing spline approximation methods*, I. J. Schoenberg, Selected Papers (C. de Boor, ed.), Contemporary Mathetics, vol. 2, Birkhäuser, 1988, pp. 247–268.

[14] C. A. Micchelli, *Interpolation of scattered data: distance matrices and conditionally positive definite functions*, Constr. Approx. **2** (1986), 11–22.

[15] K. Mørken and M. Reimers, *An unconditionally convergent method for computing zeros of splines and polynomials*, Math. Comp.

[16] J. Nocedal and S. J Wright, *Numerical optimization*, Springer Series in Operations Research, Springer, 1999.

[17] G. Nürnberger, *Approximation by spline functions*, Springer–Verlag, 1989.

[18] I. J. Schoenberg, *Cardinal spline interpolation*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 12, SIAM, 1973.

[19] L. L. Schumaker, *Spline funtions: Basic theory*, Pure and Applied Mathematics: A Wiley–Interscience Series of Texts, Monographs and Tracts, John Wiley & Sons, 1981.

[20] P. Spellucci, *Numerische Verfahren der nichtlinearen Optimierung*, Internationale Schriftenreihe zu Numerischen Mathematik, Birkhäuser, 1993.

Index B

approximation order, 30

B–spline Gramian, 33
B–splines, 3
Bézier representation, 48
Boehm algorithm, 45
boundary knots, 6

comparison of spline curves, 49
control points, 4
control polygon, 6
convex combination, 47
convex combinations, 6
cross validation, 37

difference matrix, 11

fitting problems, 37
free variable, 35, 41
Frobenius norm, 49

Gauss quadrature formula, 13
global maximum, 34
Greville abscissae, 20, 29, 50

interpolation problem, 15
interpolation sites, 19
intersection, 50

knot removal, 52
knot sequence, 2, 4
knot spacing, 30
kriging, 32

Lagrange multipliers, 32
linear precision, 29
linear regression, 39

Marsden identity, 31
monomial representation, 49
multiplicity, 2

natural cubic spline, 39
natural spline, 16
natural spline interpolant, 16
Newton's method, 50
normal equations, 53

optimal interpolation, 32
Oslo algorithm, 45

piecewise linear curve, 34
piecewise polynomial form, 48

quadratic functional, 33
quasi–interpolant, 29

reconstruction, 29
refinement, 45
relevant knots, 15

Schoenberg operator, 20, 29
Schoenberg–Whitney condition, 20
shape information, 25
simple knots, 3
simultaneous zeros, 51
sites, 15
slack variables, 34
smoothing splines, 36
smoothing term, 53
spline curve, 4
spline function, 52
spline space, 7

truncated powers, 8

Vandermonde matrix, 15
variation diminishing, 31, 51

vector sup–norm, 49
virtual knot insertion, 47, 50

weights, 36

zeros of functions, 50