PFC/RR-83-27 DOE/ET-51013-98

UC-20

Spline Techniques for Magnetic Fields

John G. Aspinall

MIT Plasma Fusion Center, Cambridge Ma. USA

June 8, 1984

Spline Techniques for Magnetic Fields.

John G. Aspinall

Spline techniques are becoming an increasingly popular method to approximate magnetic fields for computational studies in controlled fusion. This report provides an introduction to the theory and practice of B-spline techniques for the computational physicist.

0. Forward — 2
1. What is a Spline? — 4
2. B-splines in One Dimension – on an Infinite Set of Knots — 8
3. B-splines in One Dimension – Boundary Conditions — 22
4. Other Methods and Bases for Interpolation — 29
5. B-splines in Multiple Dimensions — 35
6. Approximating a Magnetic Field with B-splines — 42
7. Algorithms and Data Structure — 49
8. Error and Sensitivity Analysis — 57

0. Forward

This report is an overview of B-spline techniques, oriented towards magnetic field computation. These techniques form a powerful mathematical approximating method for many physics and engineering calculations.

Sections 1, 2, and 3 form the core of the report. In section 1, the concept of a polynomial spline is introduced. Section 2 shows how a particular spline with well chosen properties, the B-spline, can be used to build any spline. In section 3, the description of how to solve a simple spline approximation problem is completed, and some practical examples of using splines are shown. All these sections deal exclusively in scalar functions of one variable for simplicity.

Section 4 is partly digression. Techniques that are not B-spline techniques, but are closely related, are covered. These methods are not needed for what follows, until the last section on errors.

Sections 5, 6, and 7 form a second group which work toward the final goal of using B-splines to approximate a magnetic field. Section 5 demonstrates how to approximate a scalar function of many variables. The necessary mathematics is completed in section 6, where the problems of approximating a vector function in general, and a magnetic field in particular, are examined. Finally some algorithms and data organization are shown in section 7.

Section 8 deals with error analysis. This is not intended as a substitute for a text on approximation theory, but simply as a brief overview of the issues involved, to show

the limits of the techniques in the previous sections.

In this report, only a subset of spline techniques, applied to a subset of all the myriad applications, are discussed. For "cultural interest" some other techniques and applications are mentioned, but the focus is generally narrow. On the other hand, certain topics that would not appear in a mathematical text are mentioned here – these are mainly suggestions about the practical implementation of these techniques in a high-level computer language. There are not many programming examples contained in the report, but where some concept can easily be expressed by showing how to implement it, that method of explanation has been employed.

The size of the domain where splines are being applied is growing larger daily, and there is no shortage of learning materials. The books by de Boor [10] and Schumaker [16] are good starting points for study of the fundamentals, while journals such as Computer Aided Design provide a view of some very thought-provoking applications.

1. What is a Spline?

When performing scientific or engineering analysis, the issue of approximation enters early. Any numerical calculation performed with finite precision involves approximation, and even analytic techniques lean heavily on methods such as expansion in terms of a power series, so that small terms can be dropped. Many interesting problems fall into the class where a certain amount of effort will obtain a good approximation, but increasing amounts of effort yield decreasing amounts of improvement to the approximation. Approximation is a fact of life.

Polynomials are used for approximations more than any other class of function. This is because their simple mathematical structure makes evaluation of the function, its derivatives and integrals, and other functionals simple.

Suppose we wish to interpolate between known values of a function. In providing an intuitive definition of a spline it is useful to first consider two extreme cases of the choice of approximating function. Consider first the polygonal approximation. (See figure 1.1.) In this case, the approximating function consists of straight line segments between neighboring points. This approximation can be considered highly local, in the sense that the approximating function between two points depends only on the known function value at those two nearby points.

Now consider fitting a single polynomial, of order equal to the number of points, as the approximating function. Lagrange interpolation, described in section 4, can be used to do this. This approximation is highly non-local, since the approximating function over its entire range depends on all the known function values. The well

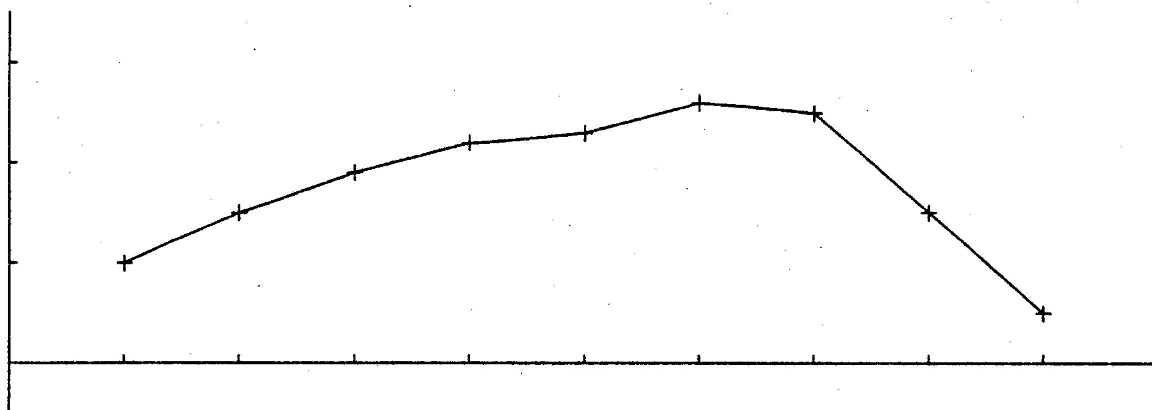

Figure 1.1: Polygonal interpolation to a set of points.

known oscillatory nature of higher order polynomials is a disadvantage; the non-local coupling (*i.e.* the effect of the function value at one point upon the polynomial far away) can also be seen by the global effect of changing one known function value. (See figure 1.2.) In fact it is quite easy to construct an example [10] where the maximum error between the function and its approximating polynomial increases as the number of interpolation points increases.

A Working Definition

The literature on splines contains a number of similar, but not identical, definitions of a spline. Let us use the following:

> *Let $x_i; i = 0, \ldots, n$ be a set of points, in non-decreasing order. Let $p_i(x); i = 1, \ldots, n$ be a set of polynomials. A* ***piecewise polynomial*** *is a function defined as being identically equal to each polynomial p_i in its respective interval. That is*
>
> $$f_{PP}(x) = p_i(x) \quad \text{for } x_{i-1} \le x < x_i.$$
>
> *A* ***spline*** *is a piecewise-polynomial with some degree of continuity from one interval to another. The breakpoints between intervals of different polynomials are called* ***knots.***

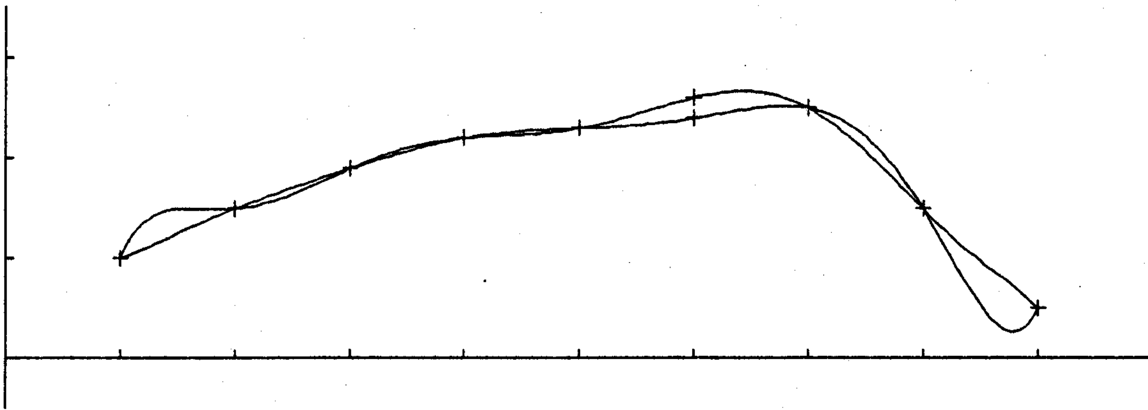

Figure 1.2: High order polynomial interpolation to two sets of points. The two sets are identical except for one member, however the interpolating polynomials differ markedly at many places in the range.

Let us return to fitting a function to the points of figures 1.1 and 1.2. A spline can be seen as a compromise between the two extremes above. A spline approximates a set of known values with a set of mth degree polynomials spanning the ranges between known points, but satisfying continuity only up to the $(m-1)$th derivative at the boundaries between spans. By discarding continuity of higher derivatives a certain amount of decoupling is provided and the function value at one point has a much more limited effect upon the approximation far away. By keeping some continuity in the lower derivatives, the approximation is smoother than the polygonal approximation.

The spline defined above is called a spline with simple knots; if only one degree of continuity is "lost" at a knot, it is called a simple knot. A multiple knot is the generalization of a simple knot; it is best thought of as the limit of several simple knots approaching each other. The multiplicity of a knot is an integer, denoting the number of degrees of continuity lost at a multiple knot.

Aside for the Mathematically Inclined. De Boor [10] fills one loophole in the above argument for piecewise polynomials. The oscillatory behavior of high order polynomials is often particularly bad for even spacing of the interpolation points. If a more appropriate set of interpolation points is chosen (Chebycheff points are the usual choice) then the error *can* be made to decrease as the number of points increases. Nevertheless, de Boor goes on to show that a choice of lower order

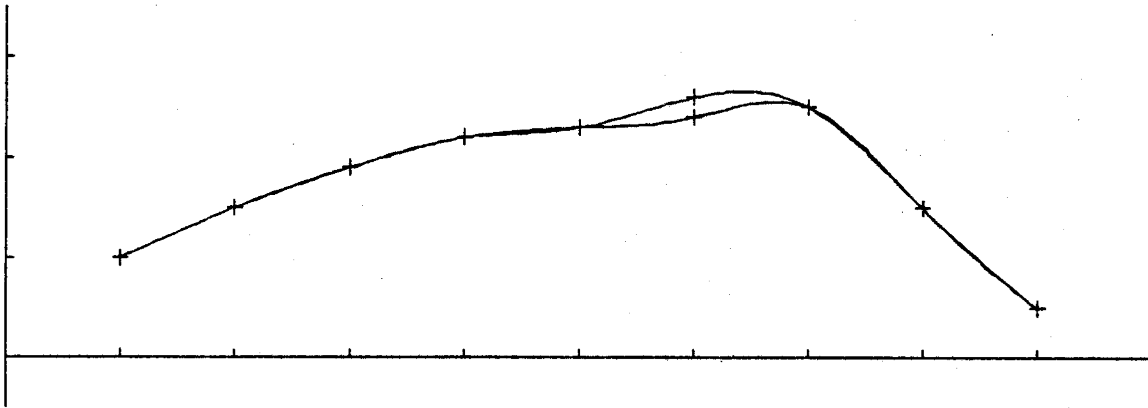

Figure 1.3: Cubic spline interpolation to the same two sets of points as figure 1.2.

piecewise polynomials can still be more efficient, and less sensitive to numerical conditioning problems.

Historical footnote. Traditionally, the spline was a draftsman's tool: a long flexible piece of wood used to draw a smooth curve through a series of points. The spline was held in place by a number of weights, called ducks, attached along the length of the spline. Under conditions of small bending this is the classical thin beam with point supports. The solution to this is cubic polynomial spans between supports and continuous curvature at the supports. The term "duck" did not make the leap into the mathematical terminology.

2. B-splines in One Dimension – on an Infinite Set of Knots

In this section the B-spline formulation of polynomial interpolation is introduced. For the sake of simplicity, all knots are treated equally; this is equivalent to interpolating on a set of knots that extend to infinity in both directions on the real line. (This does *not* mean that the knots are equally spaced – simply that there is no "first" or "last" knot in the sequence.) In reality, this is very close to the situation where the interpolating function is periodic in the independent variable.

Consider this as a typical problem to be solved:

> *Solve for the cubic spline that interpolates a function $f(x)$, where we know function values $f(x_i)$. The interpolation points x_i will be the knots of the spline.*
>
> **Aside for the Mathematically Inclined.** Remember that a knot is the point where the approximating spline loses its higher degrees of continuity. It does *not* have to be the same point as the interpolating point where the function and its approximation take on the same value. In most of what follows, the two will be the same, but it is worth remembering that this is not mandatory. In effect, by doing this, we are specifying what we mean by the "correct" interpolation – the one that solves the problem posed immediately above.

A Naive Formulation

Let's consider a very naive formulation of the spline fitting problem posed above. This is not the B-spline formulation, but it will serve to counterpoint the B-spline (and other) methods. The general cubic in the interval from x_i to x_{i+1} will be written in the form

$$A_i + B_i(x - x_i) + C_i(x - x_i)^2 + D_i(x - x_i)^3. \tag{2.1}$$

Then for each interval we have four equations: matching the function value at the lower end of the interval:

$$A_i = f(x_i); \tag{2.2}$$

matching the function value at the higher end of the interval:

$$A_i + B_i(x_{i+1} - x_i) + C_i(x_{i+1} - x_i)^2 + D_i(x_{i+1} - x_i)^3 = f(x_{i+1}); \tag{2.3}$$

ensuring continuity of the first derivative at the higher end of the interval:

$$B_i + 2C_i(x_{i+1} - x_i) + 3D_i(x_{i+1} - x_i)^2 = B_{i+1}; \tag{2.4}$$

and doing the same for the second derivative:

$$2C_i + 6D_i(x_{i+1} - x_i) = 2C_{i+1}. \tag{2.5}$$

Remembering that we are ignoring the problem of end points for the moment, and noting that the continuity equations at the lower end of the interval "belong to" the next lower interval, we see that for n piecewise continuous polynomials we have a system of $4n$ equations to solve.

There are many ways to reduce the size of the system of equations shown above, each involving formulating the problem differently. Most reduce the system to one of n (plus one or two, sometimes) equations. One of the more powerful ways is to use a basis function called a B-spline.

The B-Spline as a Basis

The concept of a basis appears in many parts of mathematical analysis, not just spline theory. The familiar ***i***, ***j*** and ***k*** unit vectors are a basis for vectors in three-dimensional space. They are a necessary and sufficient set such that any vector in the space can be represented as a unique linear combination of the members of the basis. In a function space, the harmonics of a Fourier series are the members of a basis for uniquely

representing any member of the set of periodic functions. Note that there are many possible bases for a given space. The dimension of a space is the greatest number of linearly independent members of that space it is possible to have.

Once we have settled on a basis for the space, any member of that space can be represented as a linear combination of the members of the basis, and *the set of coefficients multiplying each member of the basis can be considered as another representation of the original function.*

A set of B-splines is a basis for the space of splines. Each B-spline is itself a spline, that is, it is a piecewise polynomial of order k (degree $k-1$), with continuity up to the $(k-m-1)$th derivative at the knots. (m is the multiplicity of the knot.) A linear combination of B-splines will retain the same properties of continuity of the individual B-splines. This is one of the motivations for using such a basis. By putting continuity into the basis functions, we can ensure the same continuity in the spline without having to include equations for that continuity explicitly in the system to be solved.

Succinctly, we seek basis functions $N_j(x)$ so that we can represent a spline $S(x)$ as a sum

$$S(x) = \sum_j A_j N_j(x), \tag{2.6}$$

and the A_j will be a new representation of $S(x)$. We will see that the choice of a B-spline for $N_j(x)$ allows a well conditioned problem to be posed in many cases for finding the coefficients A_j, as well as an efficient evaluation of the sum when evaluating the function.

What does a B-spline look like? It is worth noting a few points before showing the general mathematical form of a B-spline.

- B-splines are zero outside a small range. This is called having a "compact basis".

- They are strictly positive within that range. A kth order B-spline will be non-zero over a range from knot i to knot $i+k$.

- A full basis for splines of a given order is made by placing a B-spline on each possible set of $k+1$ consecutive knots. Therefore there will be k non-zero basis functions overlapping in any given interval between two adjacent knots.

The One-Sided Polynomial

B-splines are constructed using the function:

$$(x)_+^n := \begin{cases} x^n, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} \tag{2.7}$$

often called the one-sided function or generalized step function. Since $(x-a)_+^{k-1}$ is a kth order spline with one knot at $x = a$, any spline can be made up from a linear combination of these one-sided functions on the knots of the desired spline. The one-sided function may be differentiated and integrated almost as if it were the polynomial of the same degree:

$$\frac{d}{dx}(x)_+^n = \begin{cases} n \cdot (x)_+^{n-1}, & \text{if } n > 0 \\ \delta(x), & \text{if } n = 0 \end{cases} \tag{2.8}$$

$$\int (x)_+^n \, dx = \frac{1}{n}(x)_+^{n+1}. \tag{2.9}$$

Finally, note that to negate the argument yields "the other side":

$$(x)_+^n + (-)^n(-x)_+^n = x^n. \tag{2.10}$$

Another look at Polygonal Approximation

Before proceeding to general spline interpolation with B-splines, let us discuss linear interpolation in terms of basis functions. Linear interpolation has few subtleties to it; this section gives a different perspective on the role of basis functions.

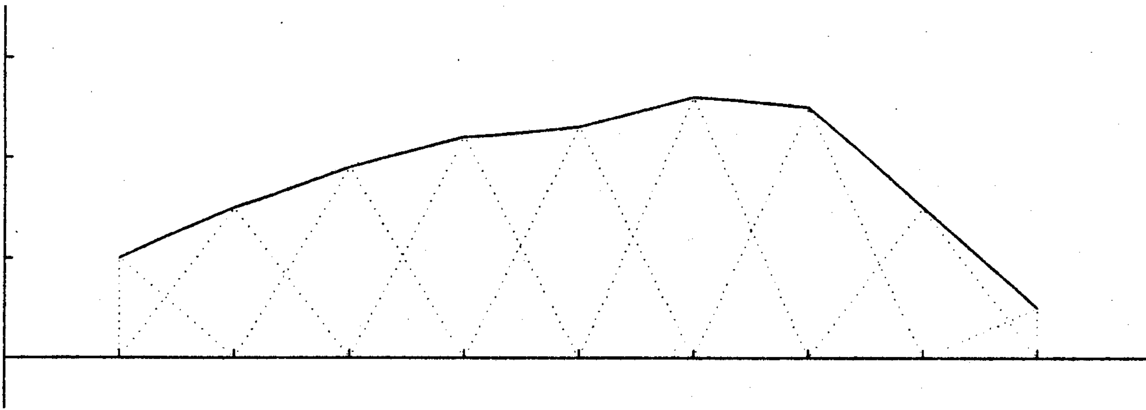

Figure 2.1: The polygonal interpolation of figure 1.1 shown as a linear combination of 2nd order B-spline basis functions ("hat" functions).

The linear B-spline basis function, or "hat" function, is the second order case of the general B-spline and is defined, using the second order step function, as follows:

$$N_{i,2}(x) := \frac{(x-x_i)_+ - (x-x_{i+1})_+}{x_{i+1}-x_i} - \frac{(x-x_{i+1})_+ - (x-x_{i+2})_+}{x_{i+2}-x_{i+1}}. \qquad (2.11)$$

In the case of equal knot spacing h, this becomes

$$N_{i,2}(x) := \frac{1}{h}((x-x_{i-1})_+ - 2(x-x_i)_+ + (x-x_{i+1})_+). \qquad (2.12)$$

Note that this function has all the properties of a B-spline listed above. It is non-zero over the range from x_i to x_{i+2}. It rises linearly to magnitude 1.0 at x_{i+1}. In any given interval between two knots there are two non-zero basis functions. The polygonal approximation of the first section can be posed as a linear combination of these basis functions. (See Figure 2.1.) Their respective magnitudes are simply the magnitudes of the function at the knots.

In general, we will see, the vector of B-spline basis function magnitudes can be found from the vector of original function magnitudes with a linear transformation; *i.e.* a matrix inversion. It won't always be as simple as the linear case above where, in effect, we had an identity matrix to invert, but the important point is that there are

only a handful of basis functions (actually k, the order, of them) that are non-zero at any point. It is this property, as advertised earlier, that makes the linear combination of basis functions easy to evaluate.

The Normalized B-spline of Arbitrary Order

As promised earlier, the general form of the B-spline is

$$N_i(x) := (x_{i+k} - x_i) g_k([x_i, x_{i+1}, \ldots, x_{i+k}]; x) \tag{2.13}$$

where $g_k([x_i, x_{i+1}, \ldots, x_{i+k}]; x)$ denotes the kth divided difference in the first argument of the function

$$g_k(\cdot\,; x) = (\cdot - x)_+^{k-1} \tag{2.14}$$

in so-called "place-holder" notation. To be precise, this is *the normalized B-spline of order k on the knot sequence $x_i, \ldots, x_{i+k}$*.

This general form will not be derived here. For the interested reader, derivations appear in many detailed books [10,11]. One important result will be presented though, as it is the starting point for efficiently evaluating B-splines on arbitrary knots.

Divided differences are functional operators that bear strong resemblance to derivatives. In particular, one can write a product rule for the divided difference of $h(x) = f(x)g(x)$ such as

$$h[x_0, x_1] = f[x_0, x_1]g[x_1] + f[x_0]g[x_0, x_1] \tag{2.15}$$

where $f[x] \equiv f(x)$, or for higher order operators

$$h[x_0, \ldots, x_k] = \sum_{r=0}^{k} f[x_0, \ldots, x_r]g[x_r, \ldots, x_k]. \tag{2.16}$$

This is attributed to Leibniz.

If (2.16) is applied to the product

$$g_k(\cdot\,; x) = (\cdot - x)\, g_{k-1}(\cdot\,; x), \tag{2.17}$$

only two terms in the sum appear, to give

$$g_k([x_i, x_{i+1}, \ldots, x_{i+k}]; x) = \\ g_{k-1}([x_i, x_{i+1}, \ldots, x_{i+k-1}]; x) + g_{k-1}([x_i, x_{i+1}, \ldots, x_{i+k}]; x)(x_{i+k} - x). \tag{2.18}$$

This is easily rewritten in terms of the B-splines, N_i,

$$N_{i,k}(x) = \frac{x - x_i}{x_{i+k-1} - x_i} N_{i,k-1}(x) + \frac{x_{i+k} - x}{x_{i+k} - x_{i+1}} N_{i+1,k-1}(x). \tag{2.19}$$

This relationship will appear again later.

It is also worth considering the form of the function in terms of the three properties listed earlier.

- The B-spline is zero outside the range (x_i, x_{i+k}). For $x < x_i$ this is obvious. All the individual truncated power functions are zero, therefore the sum is zero. For $x > x_{i+k}$ the one-sided nature of the power functions is not relevant here as all of the terms are non-zero. Therefore the function is the kth divided difference of a kth order ordinary polynomial, which is zero.

- The B-spline is strictly positive within the range (x_i, x_{i+k}). By inspection this is true for the first and second order B-splines. The coefficients in front of $N_{i,k-1}$ and $N_{i+1,k-1}$ in the recursion relation (2.19) are always positive. Therefore by induction this is true for all orders.

- The B-splines form a full basis. It will not be proved rigorously here that the set of B-splines on a series of knots is a basis for the set of all splines on those knots; the proof is done by Holland and Sahney [11] among others. Intuitively, though, it should be clear that the different B-spline basis functions are linearly independent, as given any pair of non-identical basis functions on simple knots we can always find some range between two knots where one is zero and the other is not. Also it should be easy to see that the k non-zero basis functions in a given range (between two knots) provide the right number of degrees of freedom for the polynomial in that range.

Aside for the Mathematically Inclined. Multiple knots render the letter of the above proof false, even though the spirit is still valid. The problem happens when there are two different basis functions starting and ending on the same value; *i.e.* at multiple knots. We then must compare the appropriate order derivative value at that point for the two splines. Because the knot will appear with different multiplicities in the different splines, there will always be some order of derivative (less than the order of the spline) where we can make the zero – non-zero distinction at that point.

Specialization to a Uniform Set of Knots

It is no surprise that the very important special case of uniform knot spacing allows some simplification. If $x_{j+1} - x_j = h$ for all j then the basis function becomes

$$N_{i,k}(x) = \frac{1}{(k-1)!\, h^{k-1}} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (x - x_{i+j})_+^{k-1}. \tag{2.20}$$

Every B-spline has the same shape here - they differ only by a translation.

The special case of uniform knot spacing will reappear several more times as various topics are covered.

Alternate Notation and Definition

At the risk of confusing the reader, there are two places where notation and usage in some texts differ from what has used so far.

First: individual B-splines have been indexed according to the lowest knot in their range. This keeps definitions cleaner when talking about splines of arbitrary order. When dealing with splines of one order only, and especially when that order is even, it is tempting to label the B-spline with the index of the knot where the B-spline reaches its maximum value. This is particularly common for cubic splines on uniformly spaced knots as the B-spline is then symmetric about its "home knot". There is little harm in this notation as long as one is aware of its use.

Second: the B-spline retains all of its continuity properties and so forth when multiplied by a constant. Therefore the basis function is sometimes defined with a different

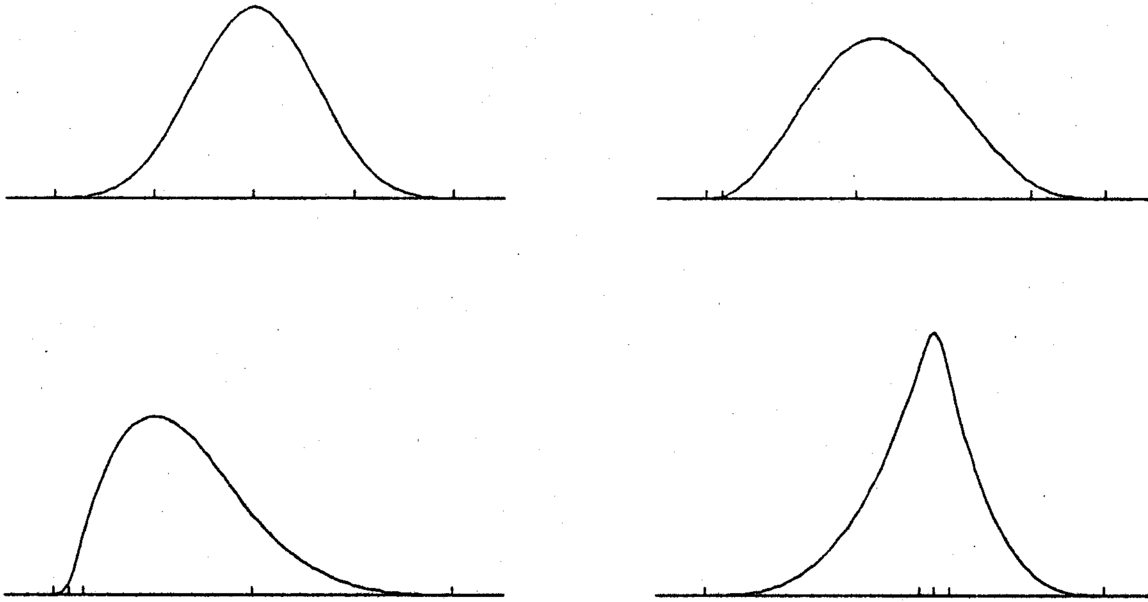

Figure 2.2: B-splines of order 4, showing the effect of knot spacing. The knots are denoted by the marks on the axis.

multiplier in front. This is a hard discrepancy to ignore in practice because it changes the relationships between splines of different order. The reason that the normalized B-spline is called "normalized", is that B-splines defined in this way form a "partition of unity". This means that at any point x, the sum of all $N_i(x)$ will be 1, or in other words if the spline function is a constant, its B-spline coefficients will all be that constant.

Evaluation of general B-splines

It may seem strange to start discussing how to evaluate a function represented as a sum of B-splines before discussing how to find the coefficients in that sum. The boundary condition techniques discussed in the next section are designed to make the boundary look like any other point in the range, and that the problem of converting

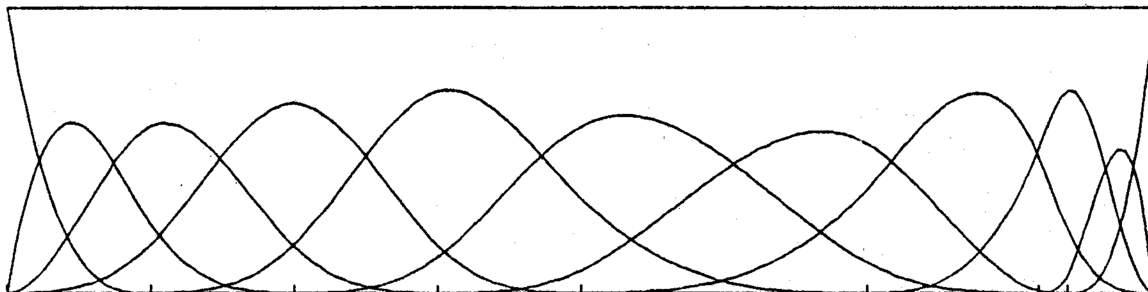

Figure 2.3: B-splines form a partition of unity. With unit coefficients, B-splines sum to 1, regardless of the knot spacing.

to a spline representation will be easier with full knowledge of how that representation will be used.

One of the features that makes B-spline evaluation fast, is that only k B-splines (where k is the order) will be non-zero at any point. The evaluator, summing terms in equation 2.6, need only consider the corresponding k coefficients times the value of their respective B-splines.

To find which B-splines contribute to the spline, a look-up function is used. A look-up function is defined as follows.

Given a non-decreasing set of values x_j, and any x, find i such that $x_i \leq x < x_{i+1}$.

A bisection method is often used in a standard look-up algorithm. A bisection method works by splitting in the search space into successive halves, until the possibilities are narowed down to one knot. Where successive function evaluations are likely to be at points close to each other (when plotting the function, for example) a look-up with guess may be used. A look-up with guess starts by searching the space around an initial guess before expanding to include the whole range. Usually the guess is the value from the previous look-up.

Having found i as above, the compact support property of the B-splines means the sum in (2.6) need only be evaluated from $i-k+1$ to i. The recursion relation (2.19) provides an efficient way to do this. Following de Boor's excellent derivation [8], let us express each kth order B-spline in terms of B-splines of the next lower order. Then

$$S(x) = \sum_j A_j N_{j,k}(x)$$

substituting from the recursion relation

$$= \sum_j A_j \left(\frac{x - x_i}{x_{i+k-1} - x_i} N_{j,k-1}(x) + \frac{x_{i+k} - x}{x_{i+k} - x_{i+1}} N_{j+1,k-1}(x) \right)$$

and regrouping terms

$$= \sum_j \left(\frac{x - x_i}{x_{i+k-1} - x_i} A_j + \frac{x_{i+k} - x}{x_{i+k} - x_{i+1}} A_{j-1} \right) N_{j,k-1}(x)$$

$$= \sum_j A_j^{[1]}(x)\, N_{j,k-1}(x) \tag{2.21}$$

where

$$A_j^{[1]}(x) = \frac{x - x_j}{x_{j+k-1} - x_j} A_j + \frac{x_{j+k-1} - x}{x_{j+k-1} - x_j} A_{j-1}. \tag{2.22}$$

Continuing this process

$$S(x) = \sum_j A_j^{[s]}(x)\, N_{j,k-s}(x) \tag{2.23}$$

where

$$A_j^{[s]}(x) = \begin{cases} A_j, & \text{if } s = 0 \\ \dfrac{x - x_j}{x_{j+k-s} - x_j} A_j^{[s-1]}(x) + \dfrac{x_{j+k-s} - x}{x_{j+k-s} - x_j} A_{j-1}^{[s-1]}(x), & \text{if } s > 0. \end{cases} \tag{2.24}$$

The key to using this new recursion relation (2.24) is that $N_{i,1}(x) = 1$ for $x_i \le x < x_{i+1}$ and zero otherwise. It immediately follows that

$$S(x) = A_i^{[k-1]}(x). \tag{2.25}$$

It is also worth pointing out at this time that this process, equation 2.24, is numerically well conditioned, as convex combinations are repeatedly formed.

Evaluating any B-spline approximation (*i.e.* equation 2.6) can be done by generating the entries in the following table, often referred to as "de Boor's triangle".

$$
\begin{array}{cccccc}
A_{i-k+1}^{[0]} & & & & & \\
A_{i-k+2}^{[0]} & A_{i-k+2}^{[1]} & & & & \\
\cdot & \cdot & \cdot & & & \\
\cdot & \cdot & & \cdot & & \\
\cdot & \cdot & & & \cdot & \\
A_{i-1}^{[0]} & A_{i-1}^{[1]} & \cdot\ \cdot\ \cdot & & A_{i-1}^{[k-2]} & \\
A_{i}^{[0]} & A_{i}^{[1]} & \cdot\ \cdot\ \cdot & & A_{i}^{[k-2]} & A_{i}^{[k-1]}
\end{array}
\tag{2.26}
$$

For efficient evaluation of the spline $S(x)$ *and* its derivatives, see Boehm [3] and Lee [12] for an elegant generalization of this method.

Evaluation of B-splines on a Uniform Set of Knots

The specialized case of uniform knot spacing can, if desired, provide some savings in the cost of evaluating the spline $S(x)$. Uniform knot spacing means that there is no need to store coordinates of all the knots – it is sufficient to store the inter-knot spacing and some sort of origin, usually the coordinate of the first knot. No look-up is needed. Translational symmetry of the B-spline means that the evaluation can be done by expressing each piece of the B-spline as a piece-wise polynomial in some coordinate local to the basis function.

Let's consider an example for cubic B-splines. The form of the 4th order B-spline is

$$
\begin{aligned}
N_i(x) = \frac{1}{6\,h^3}((x-x_i)_+^3 - 4(x-x_{i+1})_+^3 + 6(x-x_{i+2})_+^3 \\
- 4(x-x_{i+3})_+^3 + (x-x_{i+4})_+^3)
\end{aligned}
\tag{2.27}
$$

(The order of the spline has not been mentioned in the notation.) So for $x_i \leq x < x_{i+1}$, the contributing terms can be simplified

$$
\begin{aligned}
\sum_j A_j N_j(x) = \\
\frac{1}{6\,h^3}\Big(\quad & A_{i-3}\left((x-x_{i-3})^3 - 4(x-x_{i-2})^3 + 6(x-x_{i-1})^3 - 4(x-x_i)^3\right) \\
& + A_{i-2}\left((x-x_{i-2})^3 - 4(x-x_{i-1})^3 + 6(x-x_i)^3\right) \\
& + A_{i-1}\left((x-x_{i-1})^3 - 4(x-x_i)^3\right) \\
& + A_i\,(x-x_i)^3\Big)
\end{aligned}
\tag{2.28}
$$

and if we define the local coordinate

$$
\tilde{x} = \frac{x - x_i}{h} \tag{2.29}
$$

then

$$
\begin{aligned}
\sum_j A_j N_j(x) = \frac{1}{6}\Big(\quad & A_{i-3}\left(1 - 3\tilde{x} + 3\tilde{x}^2 - \tilde{x}^3\right) \\
& + A_{i-2}\left(4 - 6\tilde{x}^2 + 3\tilde{x}^3\right) \\
& + A_{i-1}\left(1 + 3\tilde{x} + 3\tilde{x}^2 - 3\tilde{x}^3\right) \\
& + A_i\,\tilde{x}^3\Big)
\end{aligned}
\tag{2.30}
$$

It is certainly possible to mechanize the procedure for converting any B-spline to its piece-wise polynomial representation. Also, it is possible to use the arbitrary knot recursion relation on evenly spaced knots. The recursion relation will be more efficient, except in special cases, than a conversion to piecewise polynomials for evaluation of a spline on arbitrary knots. And uniformly spaced knots are the only commonly occurring special case where this doesn't apply.

A Preview of Solving the Interpolation Problem

There are many ways to "best" fit one function (*i.e.* the spline) to another. When expressing a vector f in a particular basis, one forms inner products with individual members of the basis to get a complete set of equations to solve. (In general, the space may be a function space, so by "vector" I mean function as well.) We get a

system of equations of the form

$$\sum_i \left\langle b_i\, b_j \right\rangle A_i = \left\langle f\, b_j \right\rangle. \tag{2.31}$$

When curve-fitting, that is fitting a function with only a few degrees of freedom to a large number of points, the criterion used for a good fit is some sort of a least-squares minimum.

In nearly all of what follows, the spline will be fitted using collocation; that is by creating a linear operator that matches the function and spline values at discreet points, and then inverting that operator. This will be fully covered in the next section, but by way of advance billing it is worth pointing out that there will be a narrow banded matrix to invert (because of the B-spline's compact basis), and that the matrix will be well conditioned, in general (because of the strictly positive nature of the B-spline).

3. B-splines in One Dimension - Boundary Conditions

Here the description of how to use B-splines in one dimension is completed, by dealing with a finite set of knots that begins and ends at some definite values. Several different techniques for this situation are presented: the final implementation decision among these choices will be seen to depend on the nature of the particular problem being solved as well as on purely mathematical criteria.

Consider a complete system of equations generated by the naive spline formulation as in (2.1). Over $n+1$ points there are n intervals, each specified by four scalars determining the particular cubic in that interval. The known function values account for $2n$ constraining equations of the form (2.2) and (2.3) (2 at each knot, except for only 1 at end points), and the conditions of first and second derivative continuity at internal knots account for $2(n-1)$ more equations of the form (2.4) and (2.5). This leaves the system underdetermined by 2 equations.

It may be tempting to assume that the B-spline formulation avoids these difficulties immediately: $n+1$ knots, $n+1$ basis functions; but this is not the case. If you consider a B-spline formulation with a basis function starting on each knot in the interpolation range, you will see that the interval between the first and second knots only has one basis function contributing to it.

There are a number of ways to sort this out, but most of them amount to placing extra knots on or outside each end of the range. The right number of basis functions then contribute to the spline in every interval in the range, and all basis functions have the same form. Now we need more equations to give us a completely determined system;

since those equations usually come from some sort of criterion applying to the end points of the range, they are lumped under the term "boundary conditions".

The choice between placing extra knots *on* the end of the interpolation range, and placing them *outside* the interpolation range is usually made on the basis of whether there is any reason to preserve uniform knot spacing. If uniform knot spacing has been used throughout the interpolation range so far, there may be reason to keep things simple and continue this spacing outside the range. However most formulations that deal with non-uniform knot spacing, such as de Boor's triangle in the previous section, can handle multiple knots without any extra effort.

A Simple Case on Uniform Knots

We want to approximate a function with a cubic spline in some interval. Knots have been placed at equal intervals h throughout the range, and so it makes sense to place additional knots with equal spacing outside the range so that spline evaluation at any point can follow the same procedure. Enough knots are placed so that every interval *within the approximation range* has the full number of basis functions contributing to it (in this case, 4). The spline is determined by the criteria that it match the function at all knots. This yields a set of equations of the form

$$\sum_j A_j N_j(ih) = f(ih) \quad \text{for } i = 0, 1, \ldots, n. \tag{3.1}$$

A quick accounting of equations and unknowns reveals the same figures as above: n intervals between knots in the range, $n+1$ function values to match, and $n+3$ basis functions, each with an unknown magnitude. For the sake of argument, it is asssumed that the first derivative of the spline is specified at the left end, and the second derivative is specified at the right end. These conditions take the form of the derivatives of equation 3.1:

$$\sum_j A_j N_j'(0) = f'(0) \tag{3.2}$$

and

$$\sum_j A_j N''_j(nh) = f''(nh). \tag{3.3}$$

This completes the set of equations. Values of the basis function and its derivatives are easily found from equations 2.27 or 2.29. A factor of h is multiplied into the first derivative equation to make it dimensionless; similarly a factor of h^2 is used in the second derivative equation. The complete system to be solved looks like this. Note the tridiagonal nature of the system (omitted elements are 0).

$$\begin{pmatrix} -\frac{1}{2} & 0 & \frac{1}{2} & & & & \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & & & \\ & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & & \\ & & & \cdots & & & \\ & & & & \cdots & & \\ & & & & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ & & & & & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ & & & & & -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} A_{-1} \\ A_0 \\ A_1 \\ \vdots \\ A_{n-1} \\ A_n \\ A_{n+1} \end{pmatrix} = \begin{pmatrix} h\,f'(0) \\ f(0) \\ f(h) \\ \vdots \\ f((n-1)h) \\ f(nh) \\ h^2\,f''(nh) \end{pmatrix} \tag{3.4}$$

The nearly tridiagonal and nearly symmetric nature of the matrix above suggests that there may be some efficient ways to solve this set of equations. The question of preferred matrix solution methods will addressed shortly.

Natural Splines

Where no strong reason for a particular boundary condition presents itself, the cubic spline is often calculated with second derivative equal to zero at the boundaries. In the physical spline mentioned in the first section, this corresponds to having no external bending moment applied at the ends. This is a particular case of a natural spline.

A spline of odd degree $(2k-1)$ is called a natural spline if its jth derivatives are equal to zero at the end points for $k+1 \le j \le 2k$.

Natural splines often occur as solutions of variational problems of the form:

$$\text{minimize} \quad \int [s''(x)]^2 \, dx. \tag{3.5}$$

Aside for the Mathematically Inclined. The natural spline minimizes the 2-norm of the spline. Another similar creature is the perfect spline – which minimizes the ∞-norm.

The first two equations in the collocation matrix will have the form

$$\begin{pmatrix} -1 & 2 & -1 & \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & \\ & & \cdots & \\ & & & \cdots \end{pmatrix} \begin{pmatrix} A_{-1} \\ A_0 \\ A_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} 0 \\ f(0) \\ \vdots \end{pmatrix} \tag{3.6}$$

which can be rearranged as

$$\begin{pmatrix} 0 & 1 & 0 & \\ 1 & 0 & 1 & \\ & & \cdots & \\ & & & \cdots \end{pmatrix} \begin{pmatrix} A_{-1} \\ A_0 \\ A_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} f(0) \\ 2f(0) \\ \vdots \end{pmatrix}. \tag{3.7}$$

The zeros on the diagonal will cause problems with the standard LU decomposition methods (including the tridiagonal solver), so this is a candidate to be solved in a reduced form.

The "Not-a-Knot" Condition

The natural spline suffers [10] from a lower order of convergence near the end points than in the rest of the range (unless f'' really is zero there).

According to de Boor, a better choice of boundary condition when one knows nothing about derivatives at the end points, is to ensure that the spline yields a single polynomial over the first two intervals. In other words – there is no loss of continuity, in any order derivative, at the first interpolation point inside the range. The term "not-a-knot" comes from the fact that this point *would* be a knot; it looks like a knot when the problem is formulated, but it is not a knot according to the definition of one.

For cubic B-splines on uniform knots, this means the additional equation in the system sets the sum jump-discontinuity in the third derivative at this point to zero, and the system will look like

$$\begin{pmatrix} 1 & -4 & 6 & -4 & 1 & & \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & & & \\ & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & & \\ & & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & \\ & & & & \cdots & & \\ & & & & & \cdots & \end{pmatrix} \begin{pmatrix} A_{-1} \\ A_0 \\ A_1 \\ A_2 \\ A_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} 0 \\ f(0) \\ f(h) \\ f(2h) \\ \vdots \end{pmatrix}. \tag{3.8}$$

With a few row operations, the first row can be reduced to

$$A_1 = \frac{-f(0) + 8f(h) - f(2h)}{6} \tag{3.9}$$

which is much more tractable.

Periodic Boundary Conditions

When interpolating a periodic function, knowledge of one full period specifies the entire function. In some sense, a periodic function has no boundary at all. (Or, any point may be the boundary.) So the purpose of applying boundary conditions to the solution procedure in this case is to make the boundary point look like any other.

Consider the concept of "neighboring basis functions" that we have used so far. The only basis functions that contribute to the spline at some particular point are those whose support includes the knots neighboring the point of interest. The effect of approximating a periodic function is best understood as making some knots at the right hand end of the range neighbors of those at the left hand end, and vice versa.

The smallest possible system to be solved, with the least redundancy, would then look like:

$$\begin{pmatrix} \frac{2}{3} & \frac{1}{6} & & & & & & & \frac{1}{6} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & & & & & \\ & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & & & & \\ & & & \cdots & & & & & \\ & & & & \cdots & & & & \\ & & & & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} & & \\ & & & & & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \frac{1}{6} & & & & & & \frac{1}{6} & \frac{2}{3} \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-3} \\ A_{n-2} \\ A_{n-1} \end{pmatrix} = \begin{pmatrix} f(0) \\ f(h) \\ f(2h) \\ \vdots \\ f((n-3)h) \\ f((n-2)h) \\ f((n-1)h) \end{pmatrix}. \quad (3.10)$$

Note that this assumes the knots in the range are numbered 0 through $n-1$, knot n *is* knot 0. Likewise, knots to the left of knot 0 are knots $n-1$, $n-2$, etc.

If you are writing a subroutine to handle a choice of boundary conditions, compatibility with other boundary formulations may be more important than solving the minimum system. In this case, other "duplicate" coefficient values may be included explicitly in the system to be solved.

Interpolation Points between Knots

It's worth having one example with a different order of spline, and this is also a good opportunity to demonstrate that interpolation points need not fall on knots of the spline.

Here is the system to be solved for fitting a quadratic (3rd order) spline to a function where the spline matches the function value at points half-way in between the knots. The interpolation range runs from 0 to nh, and knots sit at the points $-5h/2, -3h/2, -h/2, \ldots, (2n+5)h/2$. The coefficient of the B-spline that starts on the first knot will be denoted A_{-5}; the next, A_{-3}; and so on.

Half-knot interval values are calculated for the quadratic B-spline:

$$N_i(x) = \frac{1}{2h^2}\Big((x - x_i)_+^2 - 3(x - x_{i+1})_+^2 + 3(x - x_{i+2})_+^2 - (x - x_{i+3})_+^2\Big), \tag{3.11}$$

which supplies most of the rows of the matrix, and the "not-a-knot" boundary condition is applied to the knots nearest each boundary. The full system looks like:

$$\begin{pmatrix} -1 & 3 & -3 & 1 & & & & \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & & & & & \\ & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & & & & \\ & & & \cdots & & & & \\ & & & & \cdots & & & \\ & & & & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & \\ & & & & & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ & & & & -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} A_{-5} \\ A_{-3} \\ A_{-1} \\ \vdots \\ A_{2n+1} \\ A_{2n+3} \\ A_{2n+5} \end{pmatrix} = \begin{pmatrix} 0 \\ f(0) \\ f(h) \\ \vdots \\ f((n-1)h) \\ f(nh) \\ 0 \end{pmatrix}. \tag{3.12}$$

Section 8, in the discussion of cardinal splines, mentions one reason why interpolating between knots may produce a better approximation than interpolating on knots, for this quadratic spline.

4. Other Methods and Bases for Interpolation

This section covers three other interpolation methods. They all share something in common with the B-spline formulation described in the previous sections. In addition, Hermite interpolation will be used as an analytic tool to deal with error analysis in section 8.

Hermite Interpolation

Any text on polynomial interpolation should cover Lagrange interpolation. To review extremely briefly — given a set of known points $(x_0, y_0), (x_1, y_1), \ldots$ the Lagrange interpolation formula gives the unique lowest order polynomial that passes through the points.

$$p_L(x) = \sum_{i=0}^{n} L_i(x)\, y_i \tag{4.1}$$

where

$$\begin{aligned} L_i(x) &= \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} \\ &= \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)} \end{aligned} \tag{4.2}$$

The L_i are called Lagrange interpolation polynomials.

Now suppose we are given not only the function values $y_i = f(x_i)$, but also the slopes $y'_i = df(x_i)/dx$ and we wish to find the lowest order polynomial that passes through the points with the correct slope at each point. This is given by the Hermite

interpolation formula:

$$p_H(x) = \sum_{i=0}^{n} H_i^{(0)}(x)\, y_i + \sum_{i=0}^{n} H_i^{(1)}(x)\, y_i' \tag{4.3}$$

where

$$H_i^{(0)}(x) = \left(1 - 2L_i'(x_i)\,(x - x_i)\right) L_i^2(x) \tag{4.4}$$

and

$$H_i^{(1)}(x) = (x - x_i)\, L_i^2(x). \tag{4.5}$$

The functions L_i are the Lagrange polynomials defined above in equation 5.2.

The simplest case of Hermite interpolation — fitting a polynomial to two points with slopes (x_0, y_0, y_0') and (x_1, y_1, y_1') — is worth considering further. This yields a cubic of the form:

$$\begin{aligned} p_H(x) = \quad & \left(\frac{(x-x_1)^2}{(x_0-x_1)^2} - 2\frac{(x-x_0)(x-x_1)^2}{(x_0-x_1)^3}\right) y_0 \\ & + \left(\frac{(x-x_0)^2}{(x_1-x_0)^2} - 2\frac{(x-x_1)(x-x_0)^2}{(x_1-x_0)^3}\right) y_1 \\ & + \frac{(x-x_0)(x-x_1)^2}{(x_0-x_1)^2}\, y_0' \\ & + \frac{(x-x_1)(x-x_0)^2}{(x_1-x_0)^2}\, y_1' \end{aligned} \tag{4.6}$$

If we set $x_0 = 0$ and $x_1 = 1$ we get a simplified and more common form in terms of a normalized independent variable:

$$\begin{aligned} p_H(x) = \quad & (2x^3 - 3x^2 + 1)\, y_0 \\ & + (-2x^3 + 3x^2)\, y_1 \\ & + (x^3 - 2x^2 + x)\, y_0' \\ & + (x^3 - x^2)\, y_1' \end{aligned} \tag{4.7}$$

which is the form in which this formula is most often seen. In the CAD (computer aided design) literature these Hermite functions are often called "blending functions".

Note the difference between fitting a cubic spline to a series of points and doing Hermite interpolation between neighboring pairs of the same points. In the spline interpolation we maintain continuity of first and second derivatives across the knots, but we do not get to specify their values. In Hermite interpolation we specify the first derivative value (and therefore also guarantee its continuity) but lose any control over the second derivative.

The cubic Hermite may however be used as a different route to the standard cubic spline. Representing each interval of the spline as a cubic Hermite interpolation polynomial we can solve for the set of (as yet, unknown) first derivative values by applying continuity conditions for the second derivative. Equating second derivative values at internal knots, the general member of the system to be solved looks like:

$$\begin{aligned}&\frac{2}{x_i - x_{i-1}} y'_{i-1} + \left(\frac{4}{x_i - x_{i-1}} + \frac{4}{x_{i+1} - x_i}\right) y'_i + \frac{2}{x_{i+1} - x_i} y'_{i+1} \\ =& \frac{-6}{(x_i - x_{i-1})^2} y_{i-1} + \left(\frac{6}{(x_i - x_{i-1})^2} - \frac{6}{(x_{i+1} - x_i)^2}\right) y_i + \frac{6}{(x_{i+1} - x_i)^2} y_{i+1}.\end{aligned} \tag{4.8}$$

Two additional equations are needed to make a fully determined system. As with the B-splines in section 3, these are usually provided by the application of some sort of boundary condition at the first and last knots.

Splines under Tension

The standard cubic spline often produces extra unwanted points of inflection, or "overshoots" when fitting abruptly changing data. (See figure 4.1.) Using analogy with the physical spline again, some tension in the flexible strip should decrease this tendency. Splines under tension and ν-splines are two approaches to mathematically modelling this process.

A spline under tension is derived [5] by solving for a function that fits a set of points, and has two continuous derivatives (like a cubic spline), but satisfies one additional condition: the quantity $f''(x) - \sigma^2 f(x)$ should vary in a linear fashion from knot to

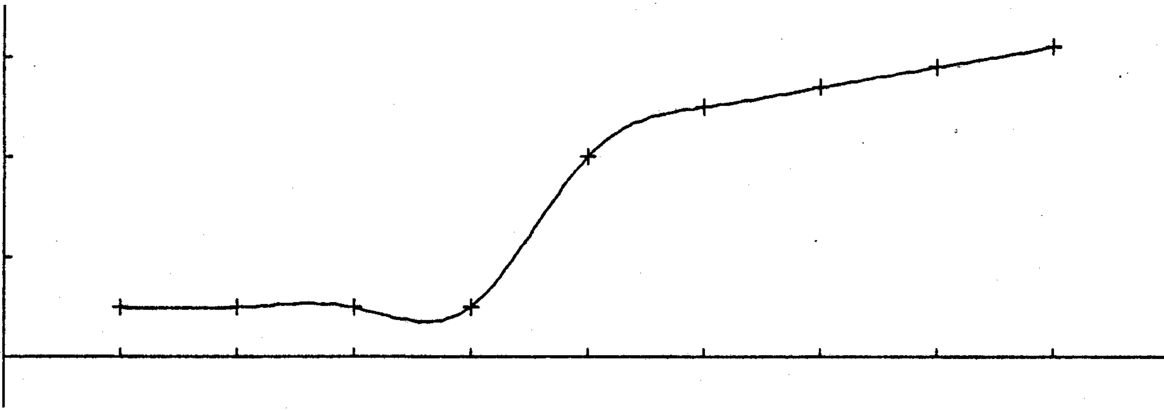

Figure 4.1: An example of "overshoot". (Actually, this function is "undershooting", but "overshoot" is the standard term.) For instance, we may not know the function exactly, but we may know that it is supposed to be monotonic. The known data is monotonic, but the spline shown here is not.

knot. The quantity σ is called the tension. As $\sigma \Rightarrow 0$ the curve approaches a cubic spline; as $\sigma \Rightarrow \infty$ the curve approaches a straight line segment.

A separate tension may be associated with each segment of the curve, although the appearance is more uniform if the same tension is applied throughout. Note that the tension, as defined here, has dimensions of the inverse of x. It is often redefined in a dimensionless way by multiplying by the average distance between knots, but this will not be done here.

Solving

$$f''(x) - \sigma^2 f(x) = \left[y_i'' - \sigma^2 y_i\right] \frac{x_{i+1} - x}{x_{i+1} - x_i} + \left[y_{i+1}'' - \sigma^2 y_{i+1}\right] \frac{x - x_i}{x_{i+1} - x_i} \tag{4.9}$$

with the function values at the knots $f(x_i) = y_i$ and $f(x_{i+1}) = y_{i+1}$ as boundary conditions we get

$$\begin{aligned} f(x) = & \frac{y_i''}{\sigma_i^2} \frac{\sinh(\sigma_i(x_{i+1} - x))}{\sinh(\sigma_i(x_{i+1} - x_i))} + \frac{y_{i+1}''}{\sigma_i^2} \frac{\sinh(\sigma_i(x - x_i))}{\sinh(\sigma_i(x_{i+1} - x_i))} \\ & + \left[y_i - \frac{y_i''}{\sigma_i^2}\right] \frac{x_{i+1} - x}{x_{i+1} - x_i} + \left[y_{i+1} - \frac{y_{i+1}''}{\sigma_i^2}\right] \frac{x - x_i}{x_{i+1} - x_i}. \end{aligned} \tag{4.10}$$

The shape of the curve in the interval from x_i to x_{i+1} is then determined by the (as yet unknown) values of y''_i and y''_{i+1}.

The solution procedure for splines under tension therefore consists of solving for the set of y'' values by applying the remaining continuity condition – that of the first derivative – at the interior knots. Differentiating equation 5.10 and equating left and right derivatives at the i-th knot, the general member of the system to be solved is

$$\begin{aligned}
&y''_{i-1}\left(\frac{1}{\sigma_{i-1}^2(x_i - x_{i-1})} - \frac{1}{\sigma_{i-1}}\operatorname{csch}(\sigma_{i-1}(x_i - x_{i-1}))\right) \\
&+y''_i\left(\frac{-1}{\sigma_{i-1}^2(x_i - x_{i-1})} + \frac{-1}{\sigma_i^2(x_{i+1} - x_i)}\right. \\
&\qquad\left. + \frac{1}{\sigma_{i-1}}\tanh(\sigma_{i-1}(x_i - x_{i-1})) + \frac{1}{\sigma_i}\tanh(\sigma_i(x_{i+1} - x_i))\right) \\
&+y''_{i+1}\left(\frac{1}{\sigma_i^2(x_{i+1} - x_i)} - \frac{1}{\sigma_i}\operatorname{csch}(\sigma_i(x_{i+1} - x_i))\right) \\
&\quad = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}}
\end{aligned} \tag{4.11}$$

which produces a diagonally dominant, tridiagonal system.

The usual variety of boundary conditions may be applied to complete the system of equations.

ν-Splines

While splines under tension achieve their desired result, they do so at the expense of introducing exponential functions in place of polynomials. In some applications this is not objectionable, but in others the increased cost of evaluating the function may be prohibitive. ν-splines were introduced [13] as a polynomial alternative to splines in tension in order to rectify this. They do so, however, at the cost of losing continuity in the second derivative.

Like splines in tension, ν-splines have a parameter called the tension. The tension in a ν-spline is associated with a point, rather than with a segment of the curve as in a

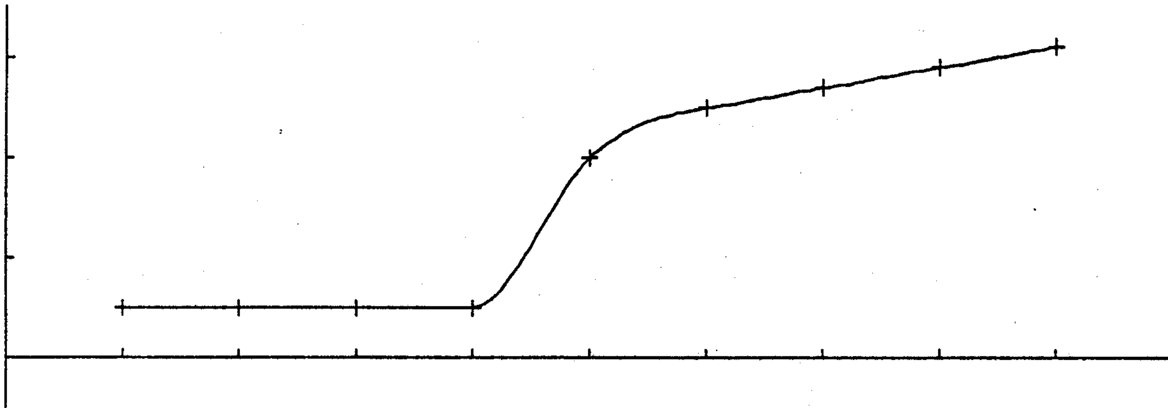

Figure 4.2: A ν-spline through the points of figure 4.1.

spline under tension; in fact the tension gives the magnitude of the discontinuity in the second derivative at that point. Zero tension at all points will again produce the standard cubic spline, and if the tensions at both ends of a given segment approach ∞, the curve will indeed approach a straight line.

The tension at a knot of a ν-spline is defined to be the ratio of the jump discontinuity in the second derivative to the (continuous) first derivative value at that point.

$$t_i \frac{df(x_i)}{dx} = \lim_{x \to x_i +} \frac{d^2 f(x)}{dx^2} - \lim_{x \to x_i -} \frac{d^2 f(x)}{dx^2} \tag{4.12}$$

The solution procedure is to represent each segment of the curve as a cubic Hermite interpolation polynomial and solve for the unknown values of the first derivatives using the known discontinuities in the second derivative. This is very similar to solving for the standard cubic spline represented as Hermite interpolation polynomials.

5. B-splines in Multiple Dimensions

In this section the B-spline formulation is generalized to interpolate a scalar function of more than one variable. The key point is the introduction of a multi-dimensional basis function which leads to the formulation of the collocation problem so that the solution procedure involves matrices no bigger than a one-dimensional case.

Suppose we have somehow found a two dimensional basis function $N(x, y)$. Presumably, by analogy with the one-dimensional case, it is non-zero over some small range about its center in all directions in the xy plane. If there are L by M knots upon which we want to approximate the function then our naive formulation suggests we will have an LM by LM matrix to invert. The matrix would be banded, but nevertheless the inversion might still be a formidable problem.

The proper construction of a multi-dimensional basis function avoids the complexity threatened above. The two dimensional basis function is defined as

$$N_{i,j}(x, y) := N_i(x)\, N_j(y), \tag{5.1}$$

where i counts knots in the x direction, and j counts knots in the y direction. The order of the spline does not appear explicitly in the notation here. The further generalization to three and more dimensions is obvious. A plot of $z = N(x, y)$, where N is cubic in both variables, is in figure 5.1.

Before coming to grips with formulating the collocation problem, a word about tensors is in order. This multi-dimensional B-spline formulation is often called a tensor formulation, because the function space of these splines is an outer product

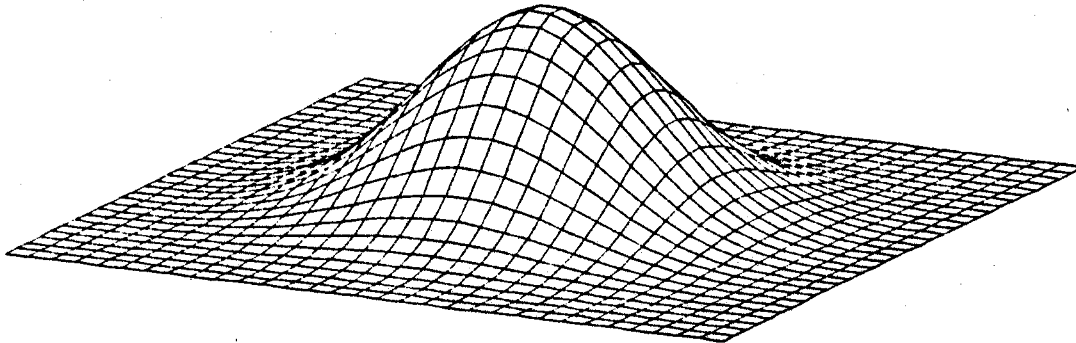

Figure 5.1: The two-dimensional cubic basis function, $N_4(x, y)$, on uniform knots. The function is plotted over the four knot – square area over which it is non-zero.

---

of two vector spaces. The only time this impacts the algebraic manipulations one does is that tensor notation is used to write the equations concisely, and so one is, in some sense, doing "tensor algebra". *Don't panic.* You don't need to know anything about tensors to understand this. If you do, fine, but all you really need to know is that quantities with more indices than you are used to (in the usual vanilla matrix algebra) will appear. Multiplication of these quantities looks more or less like matrix multiplication, except it will be necessary to note along which index of each quantity the sum of products proceeds. This is the "repeated index" rule of tensor notation.

> **Aside for the Mathematically Inclined.** Forget about contra- and co-variant indices for the moment. The notation here uses super- and sub-scripts to keep some things clear, but no implication about the way these things transform from one coordinate system to another is meant.

Because of this outer product nature it is significantly easier to deal with a mesh of knots that is (topologically) rectangular.

The Collocation Operator in Multi-Dimensions

Without worrying about boundary conditions yet, let's show the general form of a collocation operator in two dimensions. The example will be to fit a spline on knots evenly spaced in the x direction (spacing h_x) and in the y direction (spacing h_y).

For the moment, let the knots in the x direction run from $i = 0$ to L, and in the y direction from $j = 0$ to M. Extra knots will be added to cope with the boundary conditions later. The spline is a double sum:

$$S(x, y) = \sum_{i,j} A_{i,j} N_{i,j}(x, y) \tag{5.2}$$

over all the basis functions. (Compare this to equation 2.6.) Separating the two dimensional basis functions:

$$S(x, y) = \sum_j \sum_i A_{i,j} N_j(y) N_i(x) \tag{5.3}$$

we form a system of $(L + 1) \cdot (M + 1)$ equations, each corresponding to one point at which we match the spline and the function value:

$$\sum_j \sum_i A_{i,j} N_i(lh_x) N_j(mh_y) = f(lh_x, mh_y) \quad \text{for} \begin{cases} l = 0, 1, \ldots, L, \\ m = 0, 1, \ldots, M. \end{cases} \tag{5.4}$$

(Compare this to equation 3.1.) In the compact tensor notation, equation 5.4 is written

$$A^{ij} N_i^l N_j^m = f^{lm}. \tag{5.5}$$

(Note that certain indices have been changed to superscript status.)

The quantities N_i^l and N_j^m are not complete yet; they lack pieces which denote the boundary conditions. Once this has been completed they look exactly like their equivalents in the one dimensional problems. They may be inverted in a standard manner, just as you would solve the one dimensional problems such as 3.4. It is this ability to separate the multi-dimensional operator into a product of single-dimensional

operators that makes the collocation problem in many dimensions no harder than the single dimensional one.

Aside for the Mathematically Inclined. The outer product formulation above is, of course, a special case of a block-structured matrix. While block-structure techniques may be applied to solve these systems, there is no need to even *form* the complete block matrix when we have separability as above.

Extra Knots in Multiple Dimensions

Extra knots and basis functions may be added in multiple dimensions, in the same manner as in one dimension. If $N_{i,j}(x,y)$ is kth order in x, and k'th order in y, then its non-zero extent covers a rectangular region from $x = x_i$ to $x = x_{i+k}$ and from $y = y_j$ to $y = y_{j+k'}$. The same rule that we applied before in one dimension can be applied here: all (two dimensional) segments of the xy plane should have the same number of basis functions contributing to the spline in those segments.

For example, suppose we are approximating with splines of order $k = 3$ in x and $k' = 4$ in y, and the x and y axes are the lower and left boundaries respectively of the approximation region. For the sake of simplicity, let the knot spacing be 1. There are $k \cdot k' = 12$ basis functions contributing to every segment. For the segment $0 \le x \le 1$, $0 \le y \le 1$ they are $N_{i,j}; i = -2, -1, 0; j = -3, -2, -1, 0$.

Boundary Conditions in Multiple Dimensions

The purpose of introducing boundary condition equations into the system to be solved is to make a complete, well posed (fully determined) set of equations. In multiple dimensions this is no different. There is the additional requirement, though, that the separability introduced above be kept in the full system.

Consider one row of N_i^l (summing along i) in 5.5 above. That row refers to the contributions of various basis functions in x (the various basis functions are indexed by i), to the point x_l. But that row appears in $M+1$ different equations, depending on the y coordinate of the point where we are matching the function.

In order to maintain the separability of the N matrices, this principle must also apply to any boundary condition equations we add to the system. For example, let us add equations which specify the partial derivative of f with respect to x to the system described in 5.4. Taking the partial derivative of 5.3 and equating the spline and the original function at $x = 0$, say:

$$\sum_j \sum_i A_{i,j} \frac{\partial}{\partial x} N_i(x=0) N_j(mh_y) = \frac{\partial}{\partial x} f(x=0, mh_y) \quad \text{for } m = 0, 1, \ldots, M\,. \tag{5.6}$$

This set of $M+1$ equations appears as another row of N_i^l.

Now do the same thing across a y boundary. This yields an extra $L+1$ equations:

$$\sum_j \sum_i A_{i,j} N_i(lh_x) \frac{\partial}{\partial y} N_j(y=0) = \frac{\partial}{\partial y} f(lh_x, y=0) \quad \text{for } l = 0, 1, \ldots, L\,. \tag{5.7}$$

This introduces another row of N_j^m.

We have regarded equation 5.5 as representing a set of linear equations. There is one scalar equation for every pair of values of the indices l and m. By including the set 5.6 as an extra row in N_i^l and the set 5.7 as an extra row in N_j^m, and insisting on the separability of 5.5, we have included one more equation in the system to be solved. This equation corresponds to the case in 5.5 when both l and m take on values which yield the boundary condition rows. It is:

$$\sum_j \sum_i A_{i,j} \frac{\partial}{\partial x} N_i(x=0) \frac{\partial}{\partial y} N_j(y=0) = \frac{\partial^2}{\partial x\, \partial y} f(x=0, y=0). \tag{5.8}$$

In general, if there is a boundary condition which is reflected in the matrix of $N(x)$ values, and another which appears in the matrix of $N(y)$ values, then an equation will appear in the system which represents the concatenation of the two boundary condition operators.

This point is often misunderstood. As long as there are sufficient independent linear equations, the system 5.4 may be solved. It is the requirement that the system be separable that coerces the form of certain equations in the system.

Equations such as 5.8 are often called "corner" conditions, because they apply to a corner point of the interpolation range. This is not the only possibility though. It is more general to think of them as the intersection of two sets of extra equations, whether those sets describe derivatives on a boundary, periodicity, "not-a-knot" conditions, or anything else.

In more dimensions, the number of possibilities grow. In three dimensions there will be two dimensional subsets (faces) where one boundary condition is applied, one dimensional subsets (edges) where two boundary conditions intersect, and individual equations (corners) where three boundary conditions intersect.

6. Approximating a Magnetic Field with B-splines

This section considers the approximation or interpolation of a multi-dimensional function (*i.e.* a vector function) of a multi-dimensional space. The main result is a negative one: that there is no ideal vector basis function and we will be left to combine the scalar basis functions of section 4 as best we can. Nevertheless, several examples (both good and bad) can be presented as guidelines for good practice.

Yes, we have no ideal vector basis functions.

In section 2, when the B-spline basis functions were first introduced, part of the rationale for their form was that desirable properties of the approximation should put into the basis functions, so that constraints ensuring those properties could be left out of the solution procedure. It would be desirable to put the divergence-free nature of the magnetic field into the basis functions used to model the field.

It is possible to build a divergence-free basis for a vector field if we abandon the separability of dimensions. This is commonly done for finite element models. Unfortunately, the requirements of locality, smoothness and separability that have been with us from the beginning work against achieving zero divergence or curl of a vector basis function made from splines. We are left then to examine the curl and divergence of our approximation after the fact; a topic that will surface again in the last section.

Choice of Quantity to Approximate

An arbitrary vector in three-dimensional space can be specified by three scalars; the next decision is how to do this with the scalar basis functions. Any quantity that is

not approximated directly, will have to be derived from the approximation.

This choice is application dependent. Even so, certain guiding principles are apparent. First of all, if we are going to approximate a vector, one of the most obvious choices, especially in an orthogonal coordinate system, is to approximate each component of the field separately. (For instance, another possibility would be to approximate the magnitude and direction cosines.) Approximating the components is a good choice numerically; it will be superior to approximating the magnitude and two direction cosines, because subtraction of quantities with similar magnitudes will be needed to find the remaining component in the other method.

In a given application, we must also chose the physical quantity to approximate. Because a set of B-spline coefficients can yield the derivative of the spline as easily as the spline (see section 7 for a Fortran function example), it is possible to approximate the vector potential A with the spline, and evaluate its curl directly, producing a divergence-free field.

Example: A Field-Line Follower using Spline Fields

As an example of a simple spline application, consider the general organization of a field line follower before and after conversion to use a spline representation of fields from filamentary conductors.

The purpose of this software was to model the static fields of a toroidal magnetic configuration; tracing out the shapes of flux surfaces and finding integral surface properties such as rotational transform.

In the "before" version, the set of software contains three programs.

- *A Model Generator*, which takes as input the parametric form of the stellarator helical windings, and other field coils. This program writes a file, usually called the model data file, which contains a model of the stellarator windings in terms of straight and curved filamentary conductors.

- *A Field Line Integrator*, which reads the model data file, and a set of commands, specifying the initial and final conditions for the integration. These consist of starting points and directions in space and maximum lengths. This program writes the field line, as it is integrated, to a file called the line data file. In this program, the field at a point is found by summing the analytic expressions for the field (derived from the Biot-Savard law) over the filaments of the model.
- *A Graphics Post-Processor*, which produces "puncture plots", rotational transform values, and other average quantities from the field line.

In the "after" version, the same function is performed by five programs.

- *A Model Generator*, the same as before.
- *A Program to Evaluate the Field* at intervals on a three-dimensional grid. This reads the model data file, evaluates the field using the analytic Biot-Savart expressions, and writes a field grid data file.
- *A Program to Fit Cubic B-splines* to the field values. This reads the field grid values and writes spline coefficient values for the field data.
- *A Field Line Integrator*, which performs the same integration as above, except that the field evaluation is done using the spline coefficient values which are read instead of the model.
- *A Graphics Post-Processor*, the same as before.

Typically, the programs would be used as follows. Having chosen to analyze a particular configuration, three or four model data files would be created for it. These models would differ in the number of consecutively linked filaments used to model the curve of a helix, and the number of parallel filaments used to model a conductor with finite cross-section. Using the old set of programs one would follow most field lines using the cruder model, using the more detailed model for final calculations.

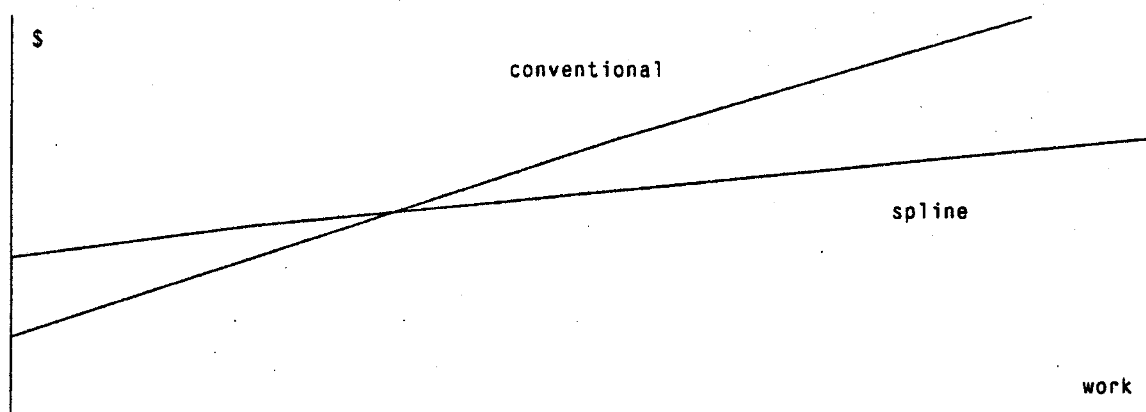

Figure 6.1: The economics of following field lines using a spline model.

When using the new (spline) set of programs one would create fewer filament models, usually one or two. Running the field grid and spline fitting programs once or twice for each model, with different grid sizes, spline coefficient data could be created for those grids. These sets of spline coefficients become the new models of the machine for input to the new integrator. They would be saved in long term storage, just as filament model files were saved previously.

The Economics of Spline Modelling

It is worthwhile to examine the comparative costs of using the two sets of programs above. In both cases there is a fixed "setup" cost, incurred before any field lines have been calculated, followed by an incremental cost, proportional to the length of lines followed. This model of the cost is fairly accurate - it ignores only small things such as the difference in the amount of file storage needed and the in-core size of the programs.

The start-up cost of the spline model is higher than the conventional one. Biot-Savart evaluations of the field have to be included in the spline start-up, and so does the spline fitting procedure. On the other hand, the cost of evaluating the field per integration step is lower. As illustrated in figure 6.1, there will be a break-even point at some finite amount of line following.

Specific values of the four constants (two fixed costs, two slopes) in the cost model above are application dependent. It is worthwhile estimating them for any large application. As an example, here are the figures for a typical problem.

Filamentary Model Complexity	:	576 straight filaments
	:	16 circular arcs
Spline Grid	:	16^3 cells (17^3 points)
Biot-Savart model setup	:	1.21 secs
Spline model setup	:	6.3 secs
Biot-Savart integration	:	3.35 secs / 1000 steps
Spline integration	:	0.78 secs / 1000 steps

For this problem, both the filamentary model and the spline grid were quite coarse. Increasing the complexity of the spline grid would increase the start-up time for the splines, but not the integration cost. Increasing the number of filaments in the initial model would increase all costs except the spline integration cost.

One additional economy in using splines has not been mentioned in the scenario above. Suppose we want to compare the effect of different currents in a vertical field coil, while leaving the rest of the conductors unchanged. The linearity of splines is useful here. Since splines are linear: if $S_f(x)$ approximates $f(x)$, and $S_g(x)$ approximates $g(x)$, then $S_f(x) + S_g(x)$ approximates $f(x) + g(x)$.

Spline coefficients can be created for that particular conductor in which one wishes to change the current. Then this set of coefficients can be added to or subtracted from the set describing the entire configuration. There is no need to perform the grid evaluations or spline inversion again.

This approach serves to reduce the start-up cost of the spline model; essentially we are spreading the cost of fewer models over the same number of lines to follow.

Splines in Other Magnetic Field Applications.

The example above has used field line following to illustrate the implementation

decisions and costs of modelling a magnetic field with splines. Field line following is one of the simpler applications of magnetic field modelling; even so, it illustrates most of the tradeoffs involved. Here I will briefly consider some other common applications and point out similarities and differences.

Collisionless particle following, using the guiding center approximation, is a similar process of integrating a set of coupled ordinary differential equations. We have seen above how the savings in computer time increase as we follow more field lines through a given configuration. This effect will be greater for collisionless guiding center integration as an interesting set of initial conditions must now span a four dimensional space (pitch angle as well as the three spatial coordinates). The details of the implementation would be quite similar.

When scattering is added to a particle following model, the dimensionality of the phase space increases again to five (energy is not conserved). In a typical test-particle – field-particle problem, with a Fokker-Planck scattering operator, the electric field is usually added to the model of the test configuration. It is quite easy to add the electric potential as a fourth scalar in the model on the same spline grid; the electric field is found directly from the spline coefficients by evaluating the derivatives of the basis functions.

Another common magnetic field calculation is the evaluation of forces on a conductor. Here it is less likely that you will model the self-fields of the test conductor with splines. The background fields are a suitable candidate for spline modelling though.

Effects of Grid Size

The effect of the choice of knot spacing, or grid size, on the costs mentioned above is clearly a strong factor. This decision should be made with some knowledge of the errors involved. While errors are covered in section 8, I will mention the basic concepts here.

If the field is sampled on a grid that is extremely coarse, important information about the shape of the field is lost. This happens when the field values change on a distance scale shorter than the spacing between grid sampling points. The approximation may bear no resemblance to the original field.

As the sampling grid gets finer, the approximation assumes the form of the original field, and the difference between the two scales as some integer power of the knot spacing. This is the region where the return on computational investment is most clear.

When the grid gets extremely fine, other factors: round-off error, error from the numerical integrator, become significant. The increasing cost of a finer grid ceases to pay off and diminishing returns set in.

7. Algorithms and Data Structure

Notes on Matrix Inversion

Examples of the matrices that we need to invert, to perform collocation, appear in section 3. As was pointed out, these matrices are often banded and close to symmetric. When fitting cubic B-splines, with function values matching on the knots, for instance, there was a tridiagonal matrix with a few extra elements in the first and last rows, depending on the boundary conditions. The choice of solution method comes down to three choices.

One possibility is brute force, using a completely general solution routine. The advantages are: no conditioning of the matrix is needed beforehand, plenty of software library routines are available, and if a new boundary condition is added (producing an unforeseen first or last row) no changes to the solution routine would be needed.

Modified LU decomposition methods, such as a tailored tridiagonal solver, are another set of candidates. Their main advantage is their efficiency. Also, only a small number, if any, of preconditioning row operations have to be performed.

For B-splines on uniform knots, the collocation matrices are almost symmetric. A few row operations can make them symmetric and positive definite in most cases. Iterative solution methods, such as conjugate gradient, are therefore a possibility.

As often happens, the choice comes down to a tradeoff between efficiency and generality. When the software described here was written, generality was chosen for two reasons. Firstly, the matrices are fairly small in most applications, and the spline

data is stored (as described in section 6) as a new model afterward. Once solved for, the splines are evaluated millions of times. So the cost of fitting the splines is a small part of the total computational cost. Programmer's cost also contributed in the same manner. It was easier to get software running, using the general tools, than it would have been with specially tailored routines.

These choices, listed above, are not the same for all applications. In some cases it may be worth writing the specially tailored tridiagonal solver. But the same sort of analysis should go into the choice as above.

Data Organization During Use of the Splines

Scientific programmers often give little thought to data organization within their programs. Sometimes little thought is needed. However it is worth considering how the spline coefficients will be stored in an applications program because the coefficients will often make up a significant proportion of the entire memory space.

Two main points to consider in the design of the data structure are: the advantages of linearity — spline representations of two fields can be added; and the question of storage order — storing one scalar at a time or one vector quantity at a time.

Combining sets of spline coefficients is quite easy to prepare for. The programmer should be aware that it is desirable to be able to read from more than one model file, storing a linear combination of the sets of coefficients to be used.

The choice between storing one scalar at a time, *i.e.* all the B_x values, then all the B_y values..., or storing all quantities one point at a time, *i.e.* (B_x, B_y, B_z) at the first point, (B_x, B_y, B_z) at the second point..., is harder to make. The spline inverter definitely wants to see one scalar at a time. However the spline evaluator deals with the evaluation of all quantities at one point at a time.

In this software, one entire scalar field is stored consecutively. The main reason for this was to be able to add another scalar (the electric potential) afterwards without

having to re-order all the other data.

Common Blocks and Argument Lists

In "old-style" Fortran programming, it was common to store all arguments to a function in a common block. This is because computers with "old-style" architectures extracted a penalty for the indirect reference through the address passed in an argument list. This made for less clarity in the program, and less flexibility in compiling pieces of the program separately.

On newer architectures, the penalty for this indirect reference is insignificant. The three-dimensional cubic spline evaluator, described later in this section, was tested on a Cray-1 in two versions: one with the coefficients in a common block, the other with them passed through the argument list. With the size of the array in common declared at compile time, there was less than 5% difference between the two methods. With the address calculations done at run-time, to provide equal flexibility, the common-block version was actually 25% slower. Common blocks have their uses. But they do not replace argument lists.

An Implementation of de Boor's Triangle

Here is a MacLisp implementation of de Boor's triangle (see equations 2.24 and 2.26). This evaluates splines of arbitrary order. The order of the spline is given by the argument `k`. Three arrays, `a`, `dm`, and `dp` are supplied for temporary storage. Array `a` stores successive columns of de Boor's triangle, the other two are used for the coefficients in equation 2.24.

The only non-standard features are the array-reference macro `aref$`, and the array-storage macro `aset$`. The function `luf` is the look-up function.

This function is based directly on de Boor's description of how to implement it; for a full description of the algorithm involved, the original reference [8] is well worth reading.

```
(defun sval (x knots coefs k a dm dp)
  (let ((i-k+1 (- (1+ (luf x knots k)) k)))
    (do ((r 0 (1+ r))
         (j i-k+1 (1+ j)))
        ((= r k))
      (aset$ (aref$ coefs j) a r)
      (aset$ (-$ x (aref$ knots j)) dm r)
      (aset$ (-$ (aref$ knots (+ j k)) x) dp r))
    (do ((kk (1- k) (1- kk))
         (s 1 (1+ s)))
        ((= kk 0) (aref$ a 0))
      (do ((r 0 (1+ r)))
          ((= r kk))
        (aset$ (//$ (+$ (*$ (aref$ dp r) (aref$ a r))
                        (*$ (aref$ dm (+ r s)) (aref$ a (1+ r))))
                    (+$ (aref$ dp r) (aref$ dm (+ r s))))
               a r)))))
```

Incidentally, this function evaluated the splines for a Lisp package that produced all the illustrations in this paper.

Cubic Spline Evaluation on Uniform Knots

Here is a Fortran implementation of one-dimensional cubic spline evaluation on uniform knots, using the piece-wise polynomial formulation of the basis functions. As mentioned earlier, the spline coefficients are passed in the argument list for greater flexibility without loss of efficiency.

The inline functions `polf1`, `polf2`, `polf3`, and `polf4` contain the four pieces of the cubic basis function as described in equation 2.30. The local variable `x` corresponds to $\tilde{x}$ of equation 2.29. The knot spacing `hx` appears in the argument list, but is not used. That is a stylistic choice to make the call to this function look the same as a call to the derivative function `s1div`, shown next.

Note that xmin is the start of the function interpolation range. It is the coordinate of the second knot, but the naive user need not know this. Also note the use of max and min functions so that if you try to evaluate outside the interpolation range, the effect is to extrapolate the polynomial in the nearest interval.

```
      real function slval (xx, nx, hx, rhx, xmin, alpha)
      integer nx
      real xx, hx, rhx, xmin, alpha(nx)

c        xx : independent variable
c        nx : number of knots
c        hx : knot spacing
c       rhx : reciprocal knot spacing
c      xmin : start of (independent variable) interpolation range
c             (actually coordinate of 2nd knot)
c     alpha : vector of basis function coefficients

      parameter (c116 = 1.0/6.0)
      polf4(x) =  1.0 + (-3.0 + ( 3.0 - 1.0 *x)* x)* x
      polf3(x) =  4.0 +         (-6.0 + 3.0 *x)* x * x
      polf2(x) =  1.0 + ( 3.0 + ( 3.0 - 3.0 *x)* x)* x
      polf1(x) =                               x * x * x

      x = (xx - xmin) * rhx + 2.0
      ix = min0 (max0 (ifix(x), 2), nx-2)
      cdx = x - float(ix)

      slval = c116*( alpha(ix-1) * polf4(cdx)
     %             + alpha(ix  ) * polf3(cdx)
     %             + alpha(ix+1) * polf2(cdx)
     %             + alpha(ix+2) * polf1(cdx) )

      return
      end
```

Here is the similar evaluation of the derivative of the spline. Note that the argument list looks the same as slval. The inline functions pold1, pold2, pold3, and pold4 implement the derivative of equation 2.30.

```
      real function s1div (xx, nx, hx, rhx, xmin, alpha)
      integer nx
      real xx, hx, rhx, xmin, alpha(nx)

c        xx : independent variable
c        nx : number of knots
c        hx : knot spacing
c       rhx : reciprocal knot spacing
c      xmin : start of (independent variable) interpolation range
c             (actually coordinate of 2nd knot)
c     alpha : vector of basis function coefficients

      pold4(x) = -1.0 + ( 2.0 - 1.0*x)* x
      pold3(x) =        (-4.0 + 3.0*x)* x
      pold2(x) =  1.0 + ( 2.0 - 3.0*x)* x
      pold1(x) =                    x * x

      x = (xx - xmin) * rhx + 2.0
      ix = min0 (max0 (ifix(x), 2), nx-2)
      cdx = x - float(ix)

      s1div = 0.5*( alpha(ix-1) * pold4(cdx)
     %            + alpha(ix  ) * pold3(cdx)
     %            + alpha(ix+1) * pold2(cdx)
     %            + alpha(ix+2) * pold1(cdx) ) * rhx

      return
      end
```

Finally, here is the corresponding three-dimensional evaluator. For the version that is used in practice, the inner two do loops at the end are unrolled, but they have been written explicitly here for clarity. Also, anyone planning to implement this on a vector machine should be aware that on the Cray-1 at least, those loops are faster *un*-vectorized. This is probably because of their short length; the overhead in setting up a vector loop outweighs the savings.

```
      real function s3val (xx, hx, rhx, xmin, narow, nacol, alpha)

      integer nx(3), narow, nacol
      real xx(3), hx(3), rhx(3), xmin(3), alpha(narow,nacol,1)

c         xx : independent variable
c         hx : mesh spacing
c        rhx : reciprocal mesh spacing
c       xmin : start of independent variable interpolation range
c              (actually coordinate of 2nd knot)
c      narow : row dimension of alpha (1st dimension varying fastest)
c      nacol : column dimension of alpha
c      alpha : three dimensional vector of basis function coefficients
c              (x coordinate varying fastest, y next fastest)

      real xt(4), yt(4), zt(4)

      parameter (c116 = (1.0/6.0)**3 )
      polf4(x) =  1.0 + (-3.0 + ( 3.0 - 1.0 *x)* x)* x
      polf3(x) =  4.0 +         (-6.0 + 3.0 *x)* x * x
      polf2(x) =  1.0 + ( 3.0 + ( 3.0 - 3.0 *x)* x)* x
      polf1(x) =                                x * x * x

      x = (xx(1) - xmin(1)) * rhx(1) + 2.0
      y = (xx(2) - xmin(2)) * rhx(2) + 2.0
      z = (xx(3) - xmin(3)) * rhx(3) + 2.0
      ix = min0 (max0 (ifix(x), 2), nx-2)
      iy = min0 (max0 (ifix(y), 2), ny-2)
      iz = min0 (max0 (ifix(z), 2), nz-2)
      cdx = x - float(ix)
      cdy = y - float(iy)
      cdz = z - float(iz)

      xt(1) = polf4(cdx)
      xt(2) = polf3(cdx)
      xt(3) = polf2(cdx)
      xt(4) = polf1(cdx)

      yt(1) = polf4(cdy)
      yt(2) = polf3(cdy)
```

```
      yt(3) = polf2(cdy)
      yt(4) = polf1(cdy)

      zt(1) = polf4(cdz)
      zt(2) = polf3(cdz)
      zt(3) = polf2(cdz)
      zt(4) = polf1(cdz)

      s3val = 0.
      do 1 kz = 1, 4
         do 1 ky = 1, 4
            do 1 kx = 1, 4
1              s3val = s3val + alpha(ix-2+kx,iy-2+ky,iz-2+kz) *
     %                          xt(kx)*yt(ky)*zt(kz)
      s3val = c116 * s3val
      return
      end
```

8. Error and Sensitivity Analysis

Sooner or later someone is going to ask – "how good is this approximation?" To answer this properly it is necessary to know something about both the theory of error in polynomial interpolation, and how to test a specific application for error. The question can be put in slightly sounder terms by posing it in three parts.

- If some properties of the "ideal" or "real" function are known, what's the worst error to expect in the approximation, compared to this?
- How much improvement can be achieved by increasing the number of data input to the approximation procedure?
- If some of the data input to the approximating algorithm change, how does the approximation change?

(There is no hope of doing justice to the mathematics of approximation theory in this small a space; rather certain tools with which to examine approximation algorithms will simply be mentioned.)

The concept of *norm* is introduced first. This leads to the notion of a *best approximation*, which is a standard with which to compare different approximation techniques. As mentioned at the end of section 6, we will need to measure the "bumpiness" of a function, or how much the function changes over various scale lengths. This is done with the *modulus of continuity*.

Next, *cardinal functions* are described. These functions provide a tool to examine the sensitivity of an approximation to changes in the data. Finally, some practical techniques to measure the error in a magnetic field approximation are mentioned.

Norms

A norm is a measure of distance in a space. The norm of something is a non-negative real number. In a space, there may be many possible norms; when a norm is used, it may refer to any norm, or it may be defined in a particular way. (Commonly used norms are often named.) Regardless of which norm is used, a norm must have three properties. They are: *uniqueness of the zero element*

$$||a|| \geq 0\,, \quad ||a|| = 0 \text{ iff } a = 0, \tag{8.1}$$

the homogeneity condition

$$||\lambda a|| = \lambda ||a|| \tag{8.2}$$

where λ is a scalar; and *the triangle inequality*

$$||a + b|| \leq ||a|| + ||b|| \tag{8.3}$$

so called because the vectors making up the sides of a triangle provide the most familiar example of this. By defining a norm in this way, $||a - b||$ is a suitable measure of the distance between a and b.

Aside for the Mathematically Inclined. A normed space is more specialized than a metric space. The distance function in a metric space must satisfy the triangle inequality, but the homogeneity condition on the norm implies a zero element in the normed space.

One norm that most people are familiar with (if not by name) is the *2-norm.* This corresponds to the usual notion of distance in Euclidean space. For a vector x

$$||x||_2 := \left(\sum_j {x_j}^2 \right)^{\frac{1}{2}}, \tag{8.4}$$

or for a function f on the range $[a, b]$

$$\|f\|_2 := \left(\frac{1}{b-a}\int_a^b f(x)^2\,dx\right)^{\frac{1}{2}}. \tag{8.5}$$

The only other norm that will be mentioned specifically here is the *infinity-norm*:

$$\|x\|_\infty := \max_j \left|x_j\right|, \tag{8.6}$$

or

$$\|x\|_\infty := \max\{f(x) : x \in [a, b]\}. \tag{8.7}$$

The error in an approximation is simply the norm of the difference between the original function and the approximation.

The Error in the Interpolating Polynomial

The expression for a single polynomial to interpolate a series of points was shown in section 4 in the Lagrange form. It is shown here in the Newton form. (Many elementary texts [6,7] show the equivalence between the two forms.)

$$Pf(x) = \sum_{i=0}^{n}(x-x_0)(x-x_1)\cdots(x-x_n)f[x_0, x_1, \ldots, x_n] \tag{8.8}$$

The Newton form is more general. If any of the x_i approach each other, we get higher order interpolation at that point without any changes to the representation. In particular, if all the x_i approach one point, we have the Taylor expansion of the function about that point.

Using Leibniz' formula (section 4), one can also show that the error can be written in a very suggestive form:

$$e(x) = f(x) - Pf(x) = (x-x_0)(x-x_1)\cdots(x-x_n)f[x_0, x_1, \ldots, x_n, x] \tag{8.9}$$

which looks like the "next" term in the Newton form, should it be expanded one more term. More importantly, it can be shown also that even if the x_i remain distinct, there exists a point ξ such that

$$f[x_0, x_1, \ldots, x_n, x] = \frac{f^{(n+1)}(\xi)}{(n+1)!} \tag{8.10}$$

where ξ is within the smallest interval containing all the values $x_0, x_1, \ldots, x_n, x$. Again, if all the x_i approach each other, we get the standard remainder term for a Taylor series.

This term will let us relate the error in an interpolating polynomial to the norm of a higher derivative of the function.

Moduli of Smoothness

The modulus of continuity, which describes how a function varies over an interval, is defined in an straightforward way. If the function f is defined in the range $[a, b]$:

$$\omega(f; h) := \max\{|f(x) - f(y)| : x, y \in [a, b]\ ,\ |x - y| \leq h\}. \tag{8.11}$$

This is said *"the modulus of continuity of f at h".*

If we generalize this and take the maximum of the rth forward difference we obtain the *rth modulus of smoothness*;

$$\omega_r(f; h) := \max\{\left|\Delta_j^r\right| f(x) : x \in [a, b - rj]\ ,\ 0 \leq j \leq h\}. \tag{8.12}$$

The modulus of continuity is therefore the first modulus of smoothness.

The modulus of continuity approaches zero as the interval approaches zero:

$$\lim_{h \to 0} \omega(f; h) = 0 \tag{8.13}$$

and for a function with a continuous first derivative we can bound it:

$$\omega(f; h) \leq h\|f'\|. \tag{8.14}$$

Best Approximations and other Standards for Comparison

One of the ways to approach the question of "how good is this approximation?", is in a relative way. Once you have decided to approximate f with, let us say, a piece-wise polynomial, you somehow conjure up an operator (an algorithm, if you wish) to apply to f, that delivers a piece-wise polynomial for your approval. Now you can ask "how good is this piece-wise polynomial approximation, compared to the best piece-wise polynomial approximation around?" Of course, now you are left to define "best", but this approach has some merit.

In typical mathematical fashion, the notion of a "best" approximation is defined in a slippery sort of way. The best approximation is simply the approximation which minimizes the norm of the difference between it and the original function. For a proper definition, and proof that a best approximation exists in a normed space, see Powell [14]. The question of what makes a best approximation has been reduced to the choice of norm. Often it is possible to find the error of a best approximation without constructing the best approximation itself.

This two-stage approach, finding the error of a best approximation and then finding the error of a practical approximation compared to the best one, is the standard way of establishing upper bounds for error.

For instance, it is well known [7, 10, 11, 14] that if we chose to approximate by interpolating a polynomial at certain points of a function, then using the Chebyshev points as the interpolation points yields the best approximation under the infinity-norm. Moreover, it is also known that choosing a "better" polynomial by *any* other means cannot improve the approximation by more than a multiplicative factor of 4. Therefore by comparing any other approximation with polynomial interpolation at Chebyshev points, one can measure the error in that approximation.

Sometimes it is useful to write the error in the best approximation as an algebraic quantity. This is done by using the notion of distance from a set: that is distance

from a function f to a set S is

$$\mathrm{dist}(f, S) := \min\{\|f - s\| : s \in S\}. \tag{8.15}$$

The Error in Approximating Functions with Splines

When measuring the error in a spline approximation, one must first describe how the approximation is made. While the collocation methods described in most of this report are not hard to understand, their properties are too complicated for some error analyses. A method known as Schoenberg's variation diminishing approximation [10] is used as a highly tractable approximation method instead.

Let $f(x)$ be the original function and $Sf(x)$ be its kth order spline approximation on knots y_i. The variation diminishing approximation simply uses a point value of the function "somewhere near" the support of the B-spline as the coefficient of that B-spline.

$$Sf(x) := \sum_i f(x_i)B_i(x) \tag{8.16}$$

This approximation, poor though it may appear at first glance, has many useful properties, including the property that it will reproduce a constant function exactly. See de Boor [10] for many more properties, including an explanation of "variation diminishing".

Following de Boor [10], we consider some specific $\tilde{x}$ in the range $x_j \le \tilde{x} < x_{j+1}$. From the local nature of the B-splines we know

$$Sf(\tilde{x}) = \sum_{i=j-k+1}^{j} f(x_i)B_i(\tilde{x}) \tag{8.17}$$

but from the fact that the B-splines sum to 1, we can write

$$f(\tilde{x}) = f(\tilde{x}) \sum_{i=j-k+1}^{j} B_i(\tilde{x}) = \sum_{i=j-k+1}^{j} f(\tilde{x})B_i(\tilde{x}) \tag{8.18}$$

and therefore we can subtract and find the error

$$f(\tilde{x}) - Sf(\tilde{x}) = \sum_{i=j-k+1}^{j} (f(\tilde{x}) - f(x_i))\, B_i(\tilde{x}) \tag{8.19}$$

and since $B_i(x) \geq 0$ always, we can take the absolute value on both sides

$$\begin{aligned} |f(\tilde{x}) - Sf(\tilde{x})| &\leq \sum_{i=j-k+1}^{j} |f(\tilde{x}) - f(x_i)|\, B_i(\tilde{x}) \\ &\leq \max\{|f(\tilde{x}) - f(x_i)|\} \end{aligned} \tag{8.20}$$

to get a bound for the error.

Now, having chosen each x_i to fall "somewhere near" the support of B_i, let us be more specific and chose

$$x_i = \frac{1}{2}(y_{i-k+1} + y_i). \tag{8.21}$$

Then

$$\begin{aligned} \max\{|f(\tilde{x}) - f(x_i)|\} &\leq \max\{|f(x) - f(y)|\} \\ &;x, y \in \{y_{i-k+1}, x_i\} \quad \text{or} \quad x, y \in \{x_i, y_i\} \end{aligned} \tag{8.22}$$

which, now looks like the modulus of continuity

$$\begin{aligned} \max\{|f(\tilde{x}) - f(x_i)|\} &\leq \omega(f; k|y_i|/2) \\ &\leq \frac{k}{2}\omega(f; |y_i|) \end{aligned} \tag{8.23}$$

where $|y_i|$ refers to the maximum knot spacing

$$|y_i| := \max_i (y_i - y_{i-1}).$$

We can conclude, therefore that

$$\text{dist}(f, S) \leq \frac{k}{2}\omega(f; |y_i|). \tag{8.24}$$

The Error in Approximating Continuous Functions with Splines

So far, we have not introduced any properties of the function f, such as continuity. If f and the spline Sf have continuous derivatives, then improved estimates of the error can be made using a procedure such as the one that follows.

Consider the set S_k of all splines of order k on a set of knots. (It is understood that this is the set of all splines on a particular set of knots y_i, say, and not the set of *all* splines, even though the set of knots is not mentioned explicitly.) If the function f is a spline, then the error in its best approximation will be zero. Therefore

$$\mathrm{dist}(f, S_k) = \mathrm{dist}(f - s, S_k) \quad \text{for all } s \in S_k. \tag{8.25}$$

Using the limit on the the modulus of continuity for continuous functions

$$\omega(f - s; h) \leq h \, \|f' - Ds\| \tag{8.26}$$

where D is the derivative operator, we can conclude that

$$\mathrm{dist}(f, S_k) \leq \frac{k}{2} |y_i| \, \|f' - Ds\|. \tag{8.27}$$

The set of all splines of lower order can produced by differentiating the continuous members of the higher order splines on the same knots.

$$S_{k-1} = \{Ds : s \in S_k \bigcap C_0\}$$

So using equation (8.24) again,

$$\mathrm{dist}(f, S_k) \leq \frac{k}{2} |y_i| \ \mathrm{dist}(f', S_{k-1}) \tag{8.28}$$

$$\mathrm{dist}(f, S_k) \leq \frac{k(k-1)}{4} |y_i| \ \omega(f'; |y_i|) \tag{8.29}$$

This procedure may be applied recursively, until the requirement of continuity in the function f or the spline cannot be met. Usually, we will end up with a limit like:

$$\mathrm{dist}(f, S_k) \leq \mathrm{constant} \, |y_i|^k \|f^{(k)}\|. \tag{8.30}$$

Cardinal Functions

An interpolation method that is linear in the data $f_i = f(x_i)$ can be expressed in the following form:

$$S(x) = \sum_j l_j(x) \, f_j \tag{8.31}$$

where $S(x)$ represents any approximation to $f(x)$, not necessarily a spline. The l_j are called cardinal functions – they depend on the positions of the x_i, but not on the data values f_i. Because of the linearity of equation (8.31) the cardinal function l_k can be obtained by applying the interpolation method to the points $l_k(x_j) = \delta_{jk}$.

Let us examine the cardinal functions for cubic spline interpolation on evenly spaced knots, where the approximation scheme matches function values on knots. If each piece of spline was represented in the Hermite form (equation (4.6) or (4.7)), then in the form of equation (4.8), continuity of the second derivative gives equations for the unknown first derivative values. For evenly spaced knots we will have a system of the form

$$l'_{i-1} + 4l'_i + l'_{i+1} = -\frac{3}{h} f_{i-1} + \frac{3}{h} f_{i+1}. \tag{8.32}$$

Consider what happens on an infinite set of knots. Equation (8.32) can be rewritten in a recursive form:

$$\begin{pmatrix} l'_{j+1} \\ l'_j \end{pmatrix} = \begin{pmatrix} -4 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} l'_j \\ l'_{j-1} \end{pmatrix} + \begin{pmatrix} -\frac{3}{h} f_{i-1} + \frac{3}{h} f_{i+1} \\ 0 \end{pmatrix} \tag{8.33}$$

and the stability of this recursion can be examined. The eigenvalues of the 2 by 2 matrix above are $\lambda = -2 \pm \sqrt{3}$; one gives an oscillating function with decaying magnitude, the other a oscillating function with growing magnitude. For the moment, the functions will be referred to as l^D and l^G respectively. These cardinal splines are shown in figure 8.1.

Both these cardinal splines pass through zero at all knots, except the central knot where their value is unity. This fact, together with the slopes from the matrix equation (8.33), defines the cardinal splines in terms of cubic Hermite interpolation polynomials.

Each cardinal spline $l_j(x)$, of a particular spline approximation (8.29), may be any linear combination

$$\lambda_j l_j^G(x) + (1 - \lambda_j) l_j^D(x) \tag{8.34}$$

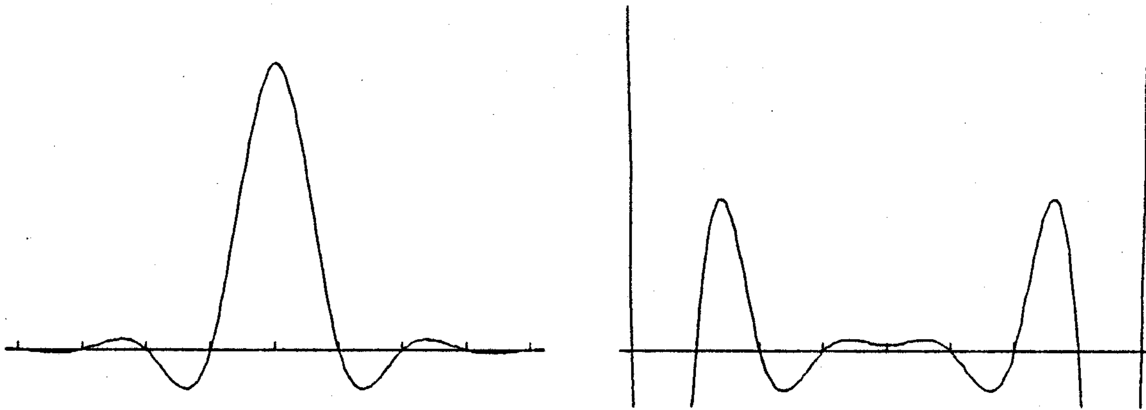

Figure 8.1: The decaying cardinal spline for cubic splines on infinite knots l^D, on the left, and the growing one l^G, on the right. The vertical scale for the growing function, on the right, has been shrunk by two orders of magnitude, compared to the one on the left, to show the shape. Both these cardinal splines have a value of 1 on their central knot.

of the two functions above. This combination is determined by the approximating algorithm; not by the specific data values. In general, we want a particular approximation to deliver as much of the decaying function and as little of the growing one as possible. This will ensure that small changes in the input data do not cause large oscillations of the spline.

The Effect of Boundary Conditions

An actual approximating algorithm, of course, must deal with boundary conditions. When boundary knots are considered, it is easier to lump all the growing cardinal splines for all the knots into two new cardinal splines. These two new cardinal functions will be called left and right (l^L and l^R) because they decay from their respective boundaries.

The linear combination (8.34) above can be rewritten as

$$l_j^D(x) + \lambda_j(l_j^G(x) - l_j^D(x)) \tag{8.35}$$

showing explicitly the desirable and undesirable parts. Let us show the slopes of l^G and l^D at a series of knots. Let λ be the stable eigenvalue $\sqrt{3} - 2$.

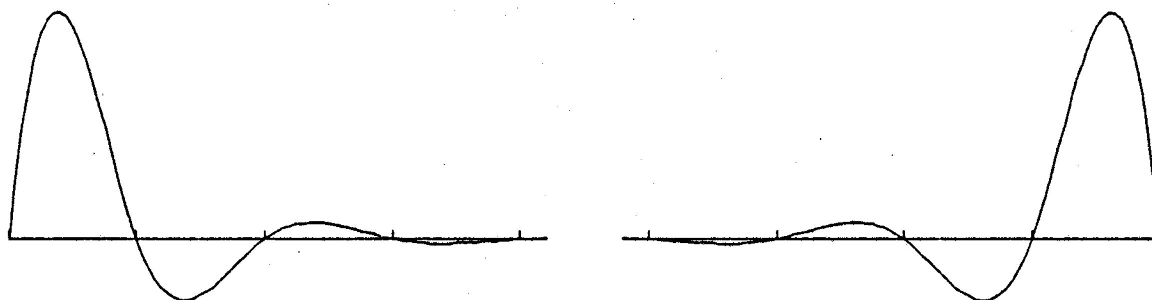

Figure 8.2: Left and right cardinal splines for boundaries. Note they have zero value at all knots.

knot	$j-3$	$j-2$	$j-1$	j	$j+1$	$j+2$	$j+3$
l^G	$-\lambda^{-3}$	$-\lambda^{-2}$	$-\lambda^{-1}$	0	λ^{-1}	λ^{-2}	λ^{-3}
l^D	$-\lambda^{3}$	$-\lambda^{2}$	$-\lambda^{1}$	0	λ^{1}	λ^{2}	λ^{3}

(8.36)

Now define l^L and l^R as oscillating splines in the same way. They pass through zero at *all* knots. Their slopes at consecutive knots have ratio λ. Let us chose their magnitudes arbitrarily to have slope 1 at knot j. Then the slopes of l^L and l^R are

knot	$j-3$	$j-2$	$j-1$	j	$j+1$	$j+2$	$j+3$
l^L	λ^{-3}	λ^{-2}	λ^{-1}	1	λ^{1}	λ^{2}	λ^{3}
l^R	λ^{3}	λ^{2}	λ^{1}	1	λ^{-1}	λ^{-2}	λ^{-3}

(8.37)

and we can see, by comparing column by column, that $l^G - l^D$ can be replaced by $l^R - l^L$. Different magnitudes for l^L and l^R will match the growing cardinal spline for different knots. Therefore the effect of all the growing cardinal functions l^G at the different knots can be replaced by the two new cardinal functions l^L and l^R.

The shape of l^L and l^R shows why poorly chosen fitting procedures produce their worst effects at the ends of the interpolation range.

Cardinal functions provide an explicit way to see the tradeoff between continuity and locality in the approximation scheme. As the order of the spline increases, the order

of the derivative that may have a discontinuity at the knot increases. So the process of damping out the oscillations of the cardinal function takes longer. In general, lower order splines have faster decaying cardinal functions.

A Cautionary Example

The example above showed the derivation of cardinal splines for cubic spline approximation with collocation *on* the knots. Placing knots *between* interpolation points can change an approximation algorithm greatly from the case where knots are placed *on* interpolation points. This happens in the case of quadratic (3rd order) splines.

The cardinal spline for quadratic spline interpolation on points spaced half-way between equidistant knots can be derived in a similar manner to the case above. In this case, there is a single zero crossing in each polynomial "loop" of the function. The resulting cardinal splines look similar to the cubic case. There is one with a decaying amplitude, and one with a growing amplitude; in this case the eigenvalues are $\lambda = -3 \pm 2\sqrt{2}$.

The cardinal spline for quadratic spline interpolation on equidistant knots has fewer degrees of freedom than the previous case, because there are two zero crossings in each piecewise polynomial. This yields a two term recursion relation for the derivatives at the knots and a purely oscillating cardinal function.

A purely oscillating cardinal function implies that changing one input data value will have an equal effect on all polynomial pieces throughout the range. This is generally undesirable in a spline interpolation, and should be avoided.

The Error in Magnetic Field Approximation

Error is best tested after a working program has been debugged. This final section therefore contains a collection of hints and techniques for after-the-fact checking of error.

It is useful to distinguish between *explicit* and *implicit* error computation. Explicit error checking involves explicit knowledge of the original exact function and computing a direct measure of the difference between the original and the approximation. For magnetic field approximation this usually means comparing field values from the Biot-Savart (or analytic, or other) model with those from the spline model. Implicit error checking usually means comparing the results of some application of the two models. An example would be to compare field lines computed from both models.

Examples of explicit error measurement could include:

- Direct differences between component values derived from original and spline models.
- Angular error between field direction from the two models.
- The finite amount of divergence and curl in the spline fields.

Examples of implicit error measurement could include:

- The difference between field lines or particle trajectories as a function of path length through the two models.
- The difference between integral properties such as average field strength and enclosed flux between surfaces in the two models.

Advantages of explicit measurement are:

- It is clear *what* is being measured (*i.e.* in some of the implicit measurements it may be hard to separate numerical error in an integrator, for instance, from error in the model).
- The scaling of such error, and therefore the costs of improving it, are known more easily from theory.

Advantages of implicit measurement are:

- The effect of error upon the application will be clearer. In fact, proper modular programming can let you run your application in both models "side by side" and assemble a collection of benchmarks.

- The "point of diminishing returns" – *ie* when error in the spline model becomes negligible compared to error in the other approximations involved – is more easily identified.

References

1. Barsky, Brian A. and Spencer W. Thomas. "TRANSPLINE – A System for Representing Curves Using Transformations among Four Spline Representations." *The Computer Journal* 24 3 (1981) p 271.

2. Boehm, Wolfgang. "On Cubics: A Survey." *Computer Graphics and Image Processing* 19 (1982) p 201.

3. Boehm, Wolfgang. "Inserting New Knots into B-spline Curves." *Computer Aided Design* 12 4 (1980) pp 199–201.

4. Brodlie, K.W. "A Review of Methods for Curve and Function Drawing." *Mathematical Methods in Computer Graphics and Design (Conference).* Academic Press, New York, N.Y., (1980).

5. Cline, A.K. "Scalar- and Planar- Valued Curve Fitting Using Splines Under Tension." *Communications of the ACM* 17 4 (April, 1974) p 281.

6. Conte, S.D., and Carl de Boor. *Elementary Numerical Analysis: an algorithmic approach.* McGraw-Hill, New York, N.Y., (1972).

7. Dahlquist, Germund and Ake Bjorck. *Numerical Methods.* Prentice Hall, Englewood Cliffs, N.J., (1974).

8. de Boor, Carl. "On Calculating with B-splines." *Journal of Approximation Theory.* 6 (1972) pp 50–62.

9. de Boor, Carl. "Package for Calculating with B-splines." *SIAM J. Numer. Anal.* 14 3 (June, 1977) p 441.

10. de Boor, Carl. *A Practical Guide to Splines.* Springer-Verlag, New York, N.Y., (1978).

11. Holland, A.S.B. and B.N. Sahney. *The General Problem of Approximation and Spline Functions.* Kreiger, Huntington, N.Y., (1979).

12. Lee, E.T.Y. "A Simplified B-Spline Computation Routine." *Computing.* 29 (1982) pp 365–371.

13. Nielson, Gregory M. "Some Piecewise Polynomial Alternatives to Splines Under Tension." *Computer Aided Geometric Design (Conference).* Academic Press, New York, N.Y., (1974) pp 209–236.

14. Powell, M.J.D. *Approximation Theory and Methods.* Cambridge University Press, Cambridge., (1981).

15. Schoenberg, I.J. *Cardinal Spline Interpolation.* S.I.A.M., Phila. Penn. (1973).

16. Schumaker, Larry L. *Spline Functions: Basic Theory.* John Wiley and Sons, New York, N.Y., (1981).