CS515, F03, Spline Lecture Notes

These are unedited notes of Carl de Boor from his Spring 2000 lectures

lecture 7apr00 :data maps, column maps, recovery, projectors, ...

Recall the setup so far. We are hoping to recover an element f of the ls F from information $\Lambda' f = (\lambda_i f : i \in I)$ about it, with each λ_i a lfnl, hence

$$\Lambda' : F \to \mathbb{F}^I$$

a linear map. We assume that Λ' is 1-1.

We only looked at the finite-dimensional case:

$$\Lambda' f = (\lambda_i : i = 1, \dots, n),$$

and assumed that Λ' is onto.

Then F is necessarily n-dimensional, hence has a basis consisting of n elements, or, equivalently, there is a reconstruction map, or column map,

$$W : \mathbb{F}^n \to F : a \mapsto \sum_{j=1}^{n} w_j a(j) =: [w_1, \dots, w_n]a$$

that is invertible.

Under that assumption, the Gram matrix $\Lambda' W = (\lambda_i w_j : i, j = 1, \dots, n)$ is invertible, and the formula

$$f = W(\Lambda' W)^{-1} \Lambda' f, \qquad \forall f \in F,$$

provides the complete reconstruction of $f \in F$ from the information $\Lambda' f$ about it.

We observed that this amounts to nothing more than a change of basis: Since $\Lambda' : F \to \mathbb{F}^n$ is invertible, its inverse, $V := (\Lambda')^{-1}$, is a linear map, hence necessarily of the form

$$V : \mathbb{F}^n \to F : a \mapsto \sum_{j=1}^{n} v_j a(j) =: [v_1, \dots, v_n]a,$$

and, necessarily,

$$\mathrm{id} = \Lambda' V = (\lambda_i v_j : i, j = 1, \dots, n)$$

the identity matrix, i.e., the two sequences (λ_i) and (v_j) are biorthonormal and, correspondingly,

$$f = V \Lambda' f = \sum_{j=1}^{n} v_j \lambda_j f, \qquad \forall f \in F.$$

In fact, necessarily

$$V = W(\Lambda' W)^{-1}.$$

 ©2000 Carl de Boor

Hence, our reconstruction consisted of the following: obtain, from the coordinates $\Lambda' f$ of f wrto the basis V, the coordinates $(\Lambda' W)^{-1}\Lambda' f$ of f wrto the basis W.

Nevertheless, we counted this a success to the extent that the coordinates of f wrto the basis W, more readily than $\Lambda' f$, provide the information about f we are after.

This led us to the basic question: *What information about $f \in F$ is readily obtainable from its coordinates wrto a given basis W for F?*

We are going to look in detail into this question for the case that F is a spline space, and the basis in question is its B-spline basis.

But before starting that discussion, here are two more general observations at this more abstract level, that will be important.

1. **linear projectors**: Extend our setup slightly, by assuming that our linear space F is actually a linear subspace of some linear space X. A concrete example would be

$$F = \Pi_{<n} \ \subset C[0\,..\,1] =: X.$$

Correspondingly, assume that our data map, Λ', is actually defined on all of X. For our earlier concrete example,

$$\Lambda' f = (f(\tau_i) : i = 1, \ldots, n),$$

this is obviously true (provided $\tau_i \in [0\,..\,1]$, all i). Then we can construct

$$Pg := W(\Lambda' W)^{-1}\Lambda' g$$

for every $g \in X$, but, while $g = Pg$ whenever $g \in F$, this need not hold for every $g \in X$. In fact, since necessarily $Pg \in F$ (why?), it follows that

$$g = Pg \quad \iff \quad g \in F.$$

In other words,

$$\operatorname{ran} P = F, \quad PP = P.$$

A linear map with the property $P^2 = P$ is called **idempotent** or a **projector**. So, P is a linear projector onto F.

What is the relationship between $g \in X$ and Pg? Since

$$\Lambda' Pg = \Lambda' W(\Lambda' W)^{-1}\Lambda' g = \Lambda' g,$$

we recognize Pg as the *unique element of F that agrees with g on Λ'* and therefore call it the unique **interpolant from F to the data $\Lambda' g$**. The most commonly used approximation schemes are all of this form, with the only difference being the choice of the data map, i.e., the choice of the information to be matched.

By considering g from a superspace for F, we cannot hope any more for lossless reconstruction; still, it is reconstruction of a sort.

2. In the first part of this course, a situation more special and more general was considered. The ls F was usually infinite-dimensional, but a Hilbert space. But, even when when F was finite-dimensional, the data map, while 1-1, was not assumed to be onto. This provided some extra flexibility, but it is important to realize what was lost.

 ©2000 Carl de Boor

If $\Lambda' = [\lambda_1, \ldots, \lambda_n]' : F \to \mathbb{F}^n$ is not onto, then

$$m := \dim F < n.$$

In particular, our basis W for F now is a map from $\mathbb{F}^m$ rather $\mathbb{F}^n$, hence, offhand, not helpful for the reconstruction. Rather, we now must choose some column map $U : \mathbb{F}^n \to F$, necessarily not 1-1, so that

$$f = U\Lambda' f, \quad \forall f \in F,$$

but it is not obvious how to do this. If U is such a map, then, certainly, the Gramian $\Lambda' U$ must be the identity on $\operatorname{ran} \Lambda'$, but that is far from having it be the identity matrix.

In contrast, if all we know is that Λ' is some data map to $\mathbb{F}^n$ and that $F = \operatorname{ran} W$ for some reconstruction map $W : \mathbb{F}^n \to X$, then just the fact that the Gramian $\Lambda' W$, which is square, is either 1-1 or onto tells us already that it must be invertible, and Λ' must be onto even when restricted to F, and W must be a basis for F, and we get the earlier reconstruction formula for all $f \in F$.

A standard sufficient condition for a square matrix to be invertible is for the matrix to be triangular with nonzero diagonal entries.

lecture 10apr00 – ??: B-splines defined

The traditional definition of a B-spline is in terms of the divided difference of the truncated power function

$$()_+^j : x \mapsto x_+^j := \begin{cases} x^j, & x \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

However, since I did not take the time to introduce divided differences (so far), I'll introduce B-splines instead 'out of the blue' via their recurrence, with the B-spline of order 1 the starting point for that recurrence.

Definition. *The* **B-spline with knots** *a, b is the characteristic function of the half-open interval $[a\,..\,b)$, i.e.,*

$$B(x|a,b) := \chi_{[a..b)}(x) = \begin{cases} 1, & a \leq x < b; \\ 0, & \textit{otherwise} \end{cases}.$$

Observations

- $B(\cdot|a,b)$ is piecewise constant, with a breaks a and b (and nowhere else), in case $a < b$; in particular, it is positive on the interval $[a\,..\,b)$, and is zero elsewhere.
- $B(\cdot|a,b)$ is the zero function in case $a \geq b$.
- $B(\cdot|a,b)$ is **right-continuous**, i.e.,

$$B(x|a,b) = B(x^+|a,b) := \lim_{h \downarrow 0} B(x+h|a,b).$$

The choice of making the B-spline right-continuous is just that, a choice, since some choice has to be made, and other choices would have been possible, e.g., the more symmetric choice

$$f(x) := (f(x-) + f(x+))/2.$$

 ©2000 Carl de Boor

The particular choice made is connected to the ppform, a standard way to represent piecewise polynomials.

If now

$$\mathbf{t} := (\cdots \le t_i \le t_{i+1} \le \cdots)$$

is a given nondecreasing sequence, we associate with it the sequence

$$B_i = B_{i,1,\mathbf{t}} := B(\cdot|t_i, t_{i+1}), \quad \forall i,$$

of first-order B-splines, and find that this sequence is a **partition of unity**, i.e.,

$$\sum_i B_i(x) = 1, \quad \inf_i t_i < x < \sup_i t_i.$$

Further, if $t_i < t_{i+1}$ for all i, then $(B_i : i)$ provides a 'basis' for the space

$$\Pi_{<1,\mathbf{t}}$$

of all (right-continuous) piecewise constant functions with possible breaks at the t_i's and nowhere else. Here, I have put 'basis' in quotes since I don't exclude the possibility that the sequence $\mathbf{t}$ is infinite or even bi-infinite, hence correspondingly consider the infinite or even bi-infinite sum

$$f = \sum_i a(i)B_i : x \mapsto \sum_i a(i)B_i(x) = \begin{cases} a(j), & t_j \le x < t_{j+1}; \\ 0, & \text{otherwise,} \end{cases}$$

i.e., as a pointwise sum.

In practice, one usually only considers finite sequences $\mathbf{t}$, but it is convenient to permit $\mathbf{t}$ to be nonfinite. In fact, in your consideration of B-splines so far, the sequence $\mathbf{t}$ was always bi-infinite, namely the sequence $\mathbf{t} = \mathbb{Z}$.

Definition. *With* $\mathbf{t}$ *continuing to denote a nondecreasing sequence,*

$$B_i := B_{i,2} := B_{i,2,\mathbf{t}} := B(\cdot|t_i, t_{i+1}, t_{i+2})x \mapsto \begin{cases} \frac{x-t_i}{t_{i+1}-t_i}, & t_i \le x < t_{i+1} \\ \frac{t_{i+2}-x}{t_{i+2}-t_{i+1}}, & t_{i+1} \le x < t_{i+2} \\ 0, & \textit{otherwise,} \end{cases}$$

is the **B-spline with knots** t_i, t_{i+1}, t_{i+2}.

Observations

- $B_{i,2} = B(\cdot|t_i, t_{i+1}, t_{i+2})$ is piecewise linear, with breaks at t_i, t_{i+1}, t_{i+2}; it is positive on $(t_i \,..\, t_{i+2})$ and is zero off $[t_i \,..\, t_{i+2}]$.
- $B_{i,2}$ is continuous from the right and it continuous at any of its breaks if and only if that break appears only once the the sequence $\mathbf{t}$.

 ©2000 Carl de Boor

It is still true that

$$\sum_i B_{i,2}(x) = 1, \quad \inf_i t_i < x < \sup_i t_i,$$

except for some possible trouble near the first or last t_i, if there is one. It is a little bit harder, though, to describe precisely the span of $(B_{i,2} : i)$.

Now the following observation: *With $\omega_{i,2}$ the linear polynomial*

$$\omega_{i,2} : x \mapsto \frac{x - t_i}{t_{i+1} - t_i}, \quad \forall i,$$

we have

$$B_{i,2} = \omega_{i,2} B_{i,1} + (1 - \omega_{i+1,2}) B_{i+1,1}.$$

Thus $B_{2,i}$ fits the following **recursive definition** of the B-spline.

Definition. *With* $\mathbf{t}$ *a nondecreasing sequence and* k *a positive integer, the B-spline with knots* $t_i, \ldots, t_{i+k}$ *is*

$$B(\cdot|t_i, \ldots, t_{i+k}) := B_{i,k} := B_{i,k,\mathbf{t}} := \begin{cases} \chi_{[t_i \ldots t_{i+1})}, & k = 1; \\ \omega_{i,k} B_{i,k-1} + (1 - \omega_{i+1,k}) B_{i+1,k-1}, & k > 1, \end{cases}$$

where $\omega_{i,k}$ is the linear polynomial

$$\omega_{i,k} : x \mapsto \frac{x - t_i}{t_{i+k-1} - t_i}.$$

Use the `MATLAB Spline Toolbox` command `bspligui` to become more familiar with just how $B_{i,k,\mathbf{t}}$ depends on its $k+1$ knots.

Also, marvel at the fact that the recurrence relation takes two functions of a certain smoothness, multiplies each of them by some linear polynomial, and then adds the resulting products and, in this way, obtains a function with higher smoothness than either summand.

The discussion of B-spline properties to follow is taken from the book (though I won't necessarily go through the proofs), hence I won't prepare class notes for it.

lecture 14apr00 – ??: B-spline recurrence used

Let

$$f := \sum_j a_j B_{jk}.$$

If $k > 1$, then, from the recurrence relations,

$$\begin{aligned} f &= \sum_j a_j (\omega_{jk} B_{j,k-1} + (1 - \omega_{j+1,k}) B_{j+1,k-1}) \\ &= \sum_j \left(a_j \omega_{jk} + a_{j-1}(1 - \omega_{j,k})\right) B_{j,k-1}. \end{aligned}$$

 ©2000 Carl de Boor

Hence, with

$$a_j^{[i+1]} := \begin{cases} a_j, & i = 0; \\ a_j^{[i]}\omega_{j,k-i+1} + a_{j-1}^{[i]}(1-\omega_{j,k-i+1}) = \dfrac{(\cdot - t_j)a_j^{[i]} + (t_{j+k-i} - \cdot)a_{j-1}^{[i]}}{t_{j+k-i} - t_j}, & i > 0, \end{cases}$$

we get

$$f \;=\; \sum_j a_j^{[i]} B_{j,k-i+1}, \qquad i = 1{:}k.$$

In particular,

$$f \;=\; \sum_j a_j^{[k]} B_{j,1},$$

with each $a_j^{[k]}$ a polynomial of degree $< k$, i.e., a polynomial of **order** k,

$$a_j^{[k]} \in \Pi_{<k}.$$

Consider some specific sequences $a = (a_j : j)$.

1. $a = \delta_j$, i.e., $f = B_{jk}$. Then, by induction, $a_j^{[i]}, \ldots, a_{j+i-1}^{[i]}$ are the only nonzero entries in $a^{[i]}$. In particular, B_{jk} has its support in the interval $[t_j \,..\, t_{j+k})$.

2. $a_j = 1$, all j. Then also $a_j^{[i]} = 1$ for all j, therefore

$$\sum_j B_{jk} = \sum_j B_{j1} = 1,$$

showing that $(B_{jk} : j)$ forms a (positive and local) partition of unity (B-spline property (ii)).

3. This example is the prettiest:

$$a_j = (t_{j+1} - \tau) \cdots (t_{j+k-1} - \tau) =: \psi_{jk}(\tau), \quad \forall j,$$

with τ arbitrary. Then

$$\begin{aligned} a_j^{[2]} &= \psi_{jk}(\tau)\omega_{jk} + \psi_{j-1,k}(\tau)(1-\omega_{jk}) \\ &= \psi_{j,k-1}(\tau)\left((t_{j+k-1} - \tau)\omega_{jk} + (t_j - \tau)(1-\omega_{jk})\right) \\ &= \psi_{j,k-1}(\tau)(\cdot - \tau). \end{aligned}$$

In other words,

$$\sum_j \psi_{jk}(\tau)B_{jk} = (\cdot - \tau)\sum_j \psi_{j,k-1}(\tau)B_{j,k-1} = \cdots = (\cdot - \tau)^{k-1}\sum_j \psi_{j,1}(\tau)B_{j,1} = (\cdot - \tau)^{k-1}.$$

This is **Marsden's identity** (B-spline property (iii)):

$$(\cdot - \tau)^{k-1} = \sum_j \psi_{jk}(\tau)B_{jk}.$$

 ©2000 Carl de Boor

From this, we get a formula for writing any $p \in \Pi_{<k}$ as a weighted sum of the B_{jk} by going through the following steps:

1. divide both sides by $(k-1)!$ (to make differentiation easier)
2. differentiate both sides $\nu - 1$ times with respect to $-\tau$
3. multiply both sides by $D^{k-\nu}p(\tau)$
4. sum all these equations for $\nu = 1, \dots, k$.

This gives the (horrendous) equation

$$\sum_{\nu=1}^{k} D^{k-\nu}p(\tau)\frac{(\cdot-\tau)^{k-\nu}}{(k-\nu)!} = \sum_{\nu=1}^{k} D^{k-\nu}p(\tau)\sum_j \frac{(-D)^{\nu-1}\psi_{jk}(\tau)}{(k-1)!}B_{jk}.$$

However, since we assume that p is of degree $< k$, the left side is just the Taylor expansion for p (around the point τ), hence equal to p. Further, since we agreed that the sum over j is to be taken pointwise and, at any point x, at most k of the $B_{jk}(x)$ are not zero, there is no difficulty with interchanging those two summations. After that, we finally get that

$$p = \sum_j \lambda_{jk}p\, B_{jk}, \qquad \forall p \in \Pi_{<k}, \tag{0.1}$$

with

$$\lambda_{jk} : f \mapsto \sum_{\nu=1}^{k} \frac{(-D)^{\nu-1}\psi_{jk}(\tau)}{(k-1)!} D^{k-\nu}f(\tau). \tag{0.2}$$

Take in the fact that this holds for an arbitrary τ. (In fact, it is easy to verify that, *for any $f \in \Pi_{<k}$, $\lambda_{jk}f$ is independent of τ*).

As a quick check, take for p a constant polynomial. Then all derivatives of p are zero everywhere, hence

$$\lambda_{jk}p = \frac{(-D)^{k-1}\psi_{jk}(\tau)}{(k-1)!}p(\tau),$$

and this equals $p(\tau)$ since $\psi_{jk}(\tau) = (-\tau)^{k-1} + \text{l.o.t.}$, hence $(-D)^{k-1}\psi_{jk} = (k-1)!$. We conclude that

$$\lambda_{jk}p = p(\tau), \qquad p \in \Pi_0.$$

A more interesting case occurs when p is a linear polynomial, say $p = \ell \in \Pi_1$. Now $D^i p(\tau) = 0$ for any $i > 1$. Therefore,

$$\lambda_{jk}\ell = \ell(\tau) + \frac{(-D)^{k-2}\psi_{jk}(\tau)}{(k-1)!}D\ell(\tau).$$

Now, since ψ_{jk} is a polynomial of exact degree $k-1$, its $(k-2)$nd derivative is a polynomial of exact degree 1, hence has exactly one zero. This zero turns out to be the point

$$t^*_{jk} := (t_{j+1} + \cdots + t_{j+k-1})/(k-1).$$

 ©2000 Carl de Boor

In other words

$$\lambda_{jk}\ell = \ell(t^*_{jk}), \qquad \forall \ell \in \Pi_1,$$

hence (B-spline property (v))

$$\ell = \sum_j \ell(t^*_{jk}) B_{jk}, \qquad \forall \ell \in \Pi_1.$$

It turns out that the formula (0.1) holds not just for $p \in \Pi_{<k}$, but for every $p = \sum_j a_j B_{jk}$ with arbitrary coefficient sequence $(a_j : j)$, provided only that the τ appearing in the definition (0.2) of the linear functional λ_{jk} be, more precisely, some point τ_j in the support of B_{jk}, i.e., from the interval $(t_j \,..\, t_{j+k})$.

lecture 17apr00 – ??: spline := linear combination of B-splines

By definition, *a spline of order k with knot sequence* $\mathbf{t}$ is a weighted sum of the B-splines $B_{j,k,\mathbf{t}}$, i.e., an element of

$$\$:= \$_{k,\mathbf{t}} := \operatorname{ran}(B_{j,k,\mathbf{t}} : j) = \{\sum_j a_j B_{j,k,\mathbf{t}} : a_j \in \mathbb{R}\}.$$

Exactly what functions are in $\$_{k,\mathbf{t}}$?

Pick a nontrivial knot interval,

$$I_j := (t_j \,..\, t_{j+1}) \neq \emptyset,$$

say. Then, on such an interval, every $B_i = B_{i,k,\mathbf{t}}$ is just a polynomial of order k, i.e., of degree $< k$, hence every $f \in \$_{k,\mathbf{t}}$ has the same property:

$$\$_{k,\mathbf{t}}|_{I_j} \subset \Pi_{<k}|_{I_j}.$$

Is there equality? That depends. For each i, let p_i be the polynomial that agrees with B_i on I_j. Then at most k of these are nonzero, namely $p_{j+1-k}, \ldots, p_j$, hence, since $\dim \Pi_{<k} = k$, there cannot be equality unless all k of these p_i actually exist. Note that there could be fewer. If, e.g., $\mathbf{t}$ has a first entry, $\mathbf{t} = (t_1, t_2, \ldots)$, say, then, for $j = 1$, only p_j has a chance of being defined. For that reason, I work with the **basic interval** for $\$_{k,\mathbf{t}}$ (see B-spline properties hand-out)

$$I_{k,\mathbf{t}} := (\mathbf{t}_- \,..\, \mathbf{t}_+),$$

with

$$\mathbf{t}_- := \begin{cases} t_k, & \text{if } \mathbf{t} = (t_1, \ldots); \\ \inf_j t_j, & \text{otherwise,} \end{cases} \qquad \mathbf{t}_+ := \begin{cases} t_{n+1}, & \text{if } \mathbf{t} = (\ldots, t_{n+k}); \\ \sup_j t_j, & \text{otherwise.} \end{cases}$$

For each $I_j \subset I_{k,\mathbf{t}}$, the full sequence $(p_{j+1-k}, \ldots, p_j)$ is defined. Moreover, we already know that

$$p = \sum_{i=j+1-k}^{j} (\lambda_{ik} p)\, p_i, \qquad \forall p \in \Pi_{<k}.$$

 ©2000 Carl de Boor

Since there are exactly k of these p_i involved here, and $\dim \Pi_{<k} = k$, we conclude that $[p_{j+1-k}, \ldots, p_j]$ is a basis for $\Pi_{<k}$ and that

$$\lambda_{ik} p_h = \delta_{ih}, \qquad i, h = j+1-k, \ldots, j,$$

i.e., the data map $\Pi_{<k} \to \mathbb{R}^k : p \mapsto (\lambda_{ik} p : i = j+1-k, \ldots, j)$ is its inverse.

The linear functionals

$$\lambda_{ik} : f \mapsto \sum_{\nu=1}^{k} \frac{(-D)^{\nu-1} \psi_{ik}(\tau_i)}{(k-1)!} D^{k-\nu} f(\tau_i) \tag{0.1}$$

involved here depend on the choice of the points τ_i. As already remarked, $\lambda_{ik} f$ is independent of the choice of τ_i in case $f \in \Pi_{<k}$. But this independence cannot hold for every $f \in \$_{k,\mathbf{t}}$: E.g., if $f = B_{ik}$ for $i = j+1-k, \ldots, j$, then we just saw that $\lambda_{ik} f = 1$ if $\tau_i \in I_j$, while it is certainly 0 in case τ_i is outside the interval $(t_i \,..\, t_{i+k})$. Remarkably, that is the *only* restriction. In particular, assuming from now on that

$$t_i < \tau_i < t_{i+k}, \quad \forall i,$$

we have

$$\lambda_{ik} B_{jk} = \delta_{ij}, \quad \forall i, j.$$

For this reason, the λ_{ik} are called the **dual functionals** (for the corresponding B-spline sequence). See B-spline property (vii).

In particular $(B_{jk} : j)$ is linearly independent, hence a basis for its span, $\$_{k,\mathbf{t}}$. It is for this reason that their creator, I. J. Schoenberg, gave them the letter 'B', as an acronym for 'Basis' or 'basic'.

By now, we know that each $f \in \$_{k,\mathbf{t}}$ is piecewise polynomial of order k with breaks at the t_i (and nowhere else). For a fuller description, return once more to Marsden's Identity:

$$(\cdot - \tau)^{k-1} = \sum_j \psi_{jk}(\tau) B_{jk},$$

but choose $\tau = t_i$ for some i, and observe that

$$\psi_{jk}(t_i) B_{jk}(t_i) = 0 \quad \forall i.$$

Indeed, if $B_{jk}(t_i) \neq 0$, then necessarily $t_j < t_i < t_{j+k}$, hence t_i is one of the zeros of ψ_{jk}. Hence,

$$(\cdot - t_i)^{k-1} = \sum_{B_{jk}(t_i)=0} \psi_{jk}(\tau) B_{jk}.$$

This sum neatly splits into two, one involving all the B-splines with support to the left of t_i, the other involving all the B-splines with support to the right of t_i. In particular,

$$(\cdot - t_i)_+^{k-1} = \sum_{j \geq i} \psi_{jk}(\tau) B_{jk}.$$

 ©2000 Carl de Boor

So, in addition to $\Pi_{<k}$, $\$_{k,\mathbf{t}}$ also contains all the truncated powers

$$(\cdot - t_i)_+^{k-1}, \qquad t_i < \mathbf{t}_+.$$

One shows in a similar way that, more generally, $\$_{k,\mathbf{t}}$ contains all the truncated powers

$$(\cdot - t_i)_+^{k-\nu}, \qquad 1 \le \nu \le \#t_i := \#\{j : t_i = t_j\},\ t_i < \mathbf{t}_+.$$

Note here the appearance of the **knot multiplicity** $\#t_i$ which counts the number of times the number t_i appears in the knot sequence $\mathbf{t}$.

And that's it, as is made clear in B-spline property (vi). Specifically, if you take any interval $I := [a\,..\,b] \subset I_{k,\mathbf{t}}$, then we now know that the entire sequence

$$(((\cdot - a)^{k-\nu} : \nu = 1{:}k), ((\cdot - t_i)_+^{k-\nu} : 1 \le \nu \le \#t_i, a < t_i < b))\big|_I$$

of $k + \#\{i : a < t_i < b\}$ functions is in $\$_{k,\mathbf{t}}|_I$ and is also easily seen to be linearly independent. On the other hand, $\$_{k,\mathbf{t}}|_I$ is spanned by the sequence $(B_{jk}|_I \neq 0)$ and this sequence also contains exactly $k + \#\{i : a < t_i < b\}$ functions. Hence, Linear Algebra tells us that both sequences must be a basis for $\$_{k,\mathbf{t}}|_I$. In particular, *each* $f \in \$_{k,\mathbf{t}}$ *satisfies (at least)* $k - \#t_i$ *smoothness conditions across the knot* t_i*, all* i, i.e., $\mathrm{jump}_{t_i} D^{r-1} f = 0$ for $r = 1{:}(k - \#t_i)$:

knot multiplicity + smoothness = order

Next: What information about $f = \sum_j a_j B_{jk}$ is 'easily' obtained from its B-spline coefficients $(a_j : j)$?

1. evaluation: . To compute $f(x)$, (i) determine j such that $t_j \le x < t_{j+1}$ and initialize $b := (a_{j+1-k}, \ldots, a_j)$; then (ii) use the recurrence:

```
for i=2:k
| for r=k:-1:i
| b(r) = ((x-t(j-k+r))*b(r) + (t(j+r-i+1)-x)*b(r-1))/...
| ((x-t(j-k+r)) + (t(j+r-i+1)-x) );
| end
end
```

After this, $b(k)$ contains the value of f at x. Note that the index for the inner loop runs down rather than up (why?). To be sure, a preferable implementation would compute the quantities `x-t(i)` and `t(k+i)-x`, `i=1:k`, needed here outside the double loop, in which case computation of the denominator is no more costly than in its simpler form `-t(j-k+r) + t(j+r-i+1)`. The present form is preferable for rounding-error control.

 ©2000 Carl de Boor

What about $Df(x), D^2f(x), \ldots$? Simply 'differentiate' the above double loop, generating with each quantity also its derivatives with respect to x. For example, if we also want the first derivative, we make b a matrix of size `[2,k]`, with $b(2,i)$ the derivative with respect to x of the quantity in $b(1,r)$, $r = 1{:}k$. This means that we initialize `b` as the 2-row matrix

$$b = \begin{bmatrix} a_{j-k+1} & \cdots & a_j \\ 0 & \cdots & 0 \end{bmatrix}$$

(since the coefficients are just constants as far as x is concerned).

The only line of code explicitly involving x is easily differentiated wrto x (note that the denominator is, in fact, independent of x) and so gives the enlarged code:

```
for i=2:k
| for r=k:-1:i
| b(2,r) = (b(1,r)-b(1,r-1) + ...
| (x-t(j-k+r))*b(2,r) + (t(j+r-i+1)-x)*b(2,r-1))/...
| ((x-t(j-k+r)) + (t(j+r-i+1)-x) );
| b(1,r) = ((x-t(j-k+r))*b(1,r) + (t(j+r-i+1)-x)*b(1,r-1))/...
| ((x-t(j-k+r)) + (t(j+r-i+1)-x) );
| end
end
```

Note that the differentiated line is executed first, thus updating the derivative values before updating the function values.

What if x does not lie in the interval $[t_j \,..\, t_{j+1})$? Then $b(k)$ (or, more generally, $b(1,k)$) contains the value at x of the polynomial that agrees with f on that interval.

2. Differentiation (B-spline property (viii)) The derivative of a spline $f = \sum_j a_j B_{jk}$ is a spline of one order lower, and its coefficients are difference quotients of the coefficients of the spline itself:

$$D\Big(\sum_j a_j B_{jk}\Big) = \sum_j \frac{a_j - a_{j-1}}{(t_{j+k-1} - t_j)/(k-1)} B_{j,k-1}.$$

To be sure, if, e.g., $t_{j+k-1} = t_j$, then that quotient multiplying $B_{j,k-1}$ is not defined. However, in that case, $B_{j,k-1}$ is the zero function, and we don't care.

Note that, in this case, $\#t_j \geq k$, i.e., f itself may have a jump discontinuity across t_j, and is not even differentiable at t_j. In effect, we ignore that, by taking the derivative here piecewise-polynomial style, i.e., for each polynomial piece separately.

As a consequence, $\int_x^y (Df)(s)\,\mathrm{d}s$ will equal $f(y) - f(x)$ in general only if the spline f is continuous on the interval $[x \,..\, y]$, for example if $\#t_i < k$ for all $t_i \in (x \,..\, y)$.

3. Good condition aka stable basis (B-spline property (x)) We already saw that, for $t_j \leq x < t_{j+1}$, the value $f(x) = \sum_{i=j-k+1}^{j} a_i B_{ik}(x)$ is a *convex* combination of the k coefficients a_{j-k+i}, $i = 1{:}k$. In particular, the value $f(x)$ must lie between the smallest

and the largest of these k coefficients. On the other hand, at least for modest k, none of these k coefficients can be too far from the value $f(x)$. Precisely,

$$|a_i| \le D_{k,\infty} \| \sum_j a_j B_{jk} \|_{[t_{i+1}..t_{i+k-1}]},$$

with $D_{k,\infty} \approx 2^{k-3/2}$.

This makes (B_{jk}) a **stable** basis (or Riesz basis) in the uniform norm in the sense that

$$(1/D_{k,\infty})\|a\|_\infty \le \| \sum_j a_j B_{jk} \|_\infty \le \|a\|_\infty.$$

But the B-spline basis has this property even *locally*.

lecture 21apr00 – ??: B-splines are refinable

The close connection between the value $f(x)$ of $f = \sum_j a_j B_{jk}$ and the 'nearby' coefficients $(a_i : B_{ik}(x) \neq 0)$ is made visible in CAGD by considering the *curve*

$$x \mapsto (x, f(x)) = (\sum_j t^*_{jk} B_{jk}, \sum_j a_j B_{jk}) =: \sum_j P_j B_{jk}$$

(note the use of B-spline property (v) here!), with

$$P_j = P_{j,k,\mathbf{t}} f := (t^*_{jk}, a_j) \in \mathbb{R}^2$$

called the **control point**s, and the broken line connecting these control points, and denoted here by

$$C_{k,\mathbf{t}} f,$$

called the **control polygon**.

This nomenclature arose in CAGD (:= Computer-Aided Geometric Design), where one considers, more generally, **spline curves**, i.e., curves of the form $x \mapsto \sum_{jk} P_j B_{jk}(x)$ with P_j arbitrary vectors in the plane (or even in 3-space or higher dimensions) and, correspondingly, its control polygon, i.e., the piecewise linear curve $x \mapsto \sum_j P_j B_{jk}(x)$.

The control polygon provides a rough outline or caricature of the spline itself. At the same time, by B-spline property (x), for modest order k, this control polygon cannot be too far from the curve itself. Sticking with a spline *function*, i.e., our scalar-valued spline $f = \sum_j a_j B_{jk}$, one infers directly from the dual functionals that

$$a_j = f(t^*_{jk}) + O((t_{j+k-1} - t_{j+1})^2 \|D^2 f\|_{[t_{j+1}..t_{j+k-1}]}).$$

This implies that the control polygon is close to the spline itself when the **mesh spacing**

$$|\mathbf{t}| := \sup_i (t_{i+1} - t_i)$$

is sufficiently small.

E.g., try out this simple example, in which a cubic spline is generated by interpolation, then plotted, along with its control polygon:

```
x = sort(rand(1,21))*4*pi; k = 4; sp = spapi(k,x,sin(x)./(.3+x));
fnplt(sp)
hold on, plot(aveknt(fnbrk(sp,'knots'),k), fnbrk(sp,'coef') ,'k'), hold
off
```

If the mesh spacing isn't small enough, we can make it smaller simply by inserting more knots. After all, if $\mathbf{t}$ is a subsequence of $\widehat{\mathbf{t}}$, then $\$_{k,\mathbf{t}}$ is a subset of $\$_{k,\widehat{\mathbf{t}}}$, i.e.,

$$\mathbf{t} \subset \widehat{\mathbf{t}} \quad \Longrightarrow \quad \$_{k,\mathbf{t}} \subset \$_{k,\widehat{\mathbf{t}}},$$

hence, in that case, each $f \in \$_{k,\mathbf{t}}$ is also uniquely writeable as a weighted sum of the $\widehat{B}_{jk} := B_{j,k,\widehat{\mathbf{t}}}$:

$$\sum_j a_j B_{jk} = f = \sum_j \widehat{a}_j \widehat{B}_{jk}. \tag{0.1}$$

E.g., continue the example:

```
sp = fnrfn(sp,aveknt(x,3));
hold on, plot(aveknt(fnbrk(sp,'knots'),k), fnbrk(sp,'coef') ,'r'), hold
off
```

The formula for the $\widehat{a}_j$ can be quite involved. However, we can obtain any **refinement** $\widehat{\mathbf{t}}$ of $\mathbf{t}$ in a sequence of steps, each of which consists of adding just *one* knot. Hence, it is sufficient to know the formula for $\widehat{a}_j$ in the special case that $\widehat{\mathbf{t}}$ is obtained from $\mathbf{t}$ by the insertion of just one additional knot. This formula constitutes B-spline property (xi). *If $\widehat{\mathbf{t}}$ is obtained from $\mathbf{t}$ by insertion of the point x, then (0.1) holds with*

$$\widehat{a}_j = \widehat{\omega}_{jk}(x) a_j + (1 - \widehat{\omega}_{jk}(x)) a_{j-1},$$

where

$$\widehat{\omega}_{jk}(x) := \max\{0, \min\{1, \omega_{jk}(x)\}\}.$$

lecture 24apr00: an aside: the ppform

Here is a standard method for the evaluation of a *cardinal* cubic spline, i.e., a cubic spline with the uniform knot sequence $\mathbb{Z}$.

Assume that we are interested in the value of $f = \sum_j a_j B_j$ at some $x \in [0\,..\,1)$. Then

$$f(x) = \sum_{i=-3}^{0} B_i(x) a_i = \sum_{i=-3}^{0} p_i(x) a_i,$$

with p_i the polynomial that agrees with B_i on the interval $(0\,..\,1)$. Let

$$p_i =: \sum_{r=1}^{4} ()^{4-r} C(r, 4+i), \quad i = -3{:}0.$$

 ©2000 Carl de Boor

Then C is the change-of-basis matrix (or transition matrix) for going from the basis $[p_{-3},\dots,p_0]$ for Π_3 to the power basis $[()^3,\dots,()^0]$, i.e.,

$$C = [()^3,\dots,()^0]^{-1}[p_{-3},\dots,p_0],$$

and

$$\begin{aligned} f(x) &= [p_{-3}(x),\dots,p_0(x)](a_{-3},\dots,a_0) \\ &= [x^3,\dots,x^0]\, C\, (a_{-3},\dots,a_0). \end{aligned} \tag{0.1}$$

In particular, the vector $\mathbf{b} := C\,(a_{-3},\dots,a_0)$ provides the coefficients in the **power form** for the cubic polynomial that agrees with f on the interval $(0\,..\,1)$. With this vector in hand, we would now compute $f(x)$, not by forming $[x^3,\dots,x^0]$ and then multiplying $\mathbf{b}$ by it, but by using Nested Multiplication (remember Nested Multiplication??), i.e., by computing

$$f(x) = ((x * b_1 + b_2) * x + b_3) * x + b_4.$$

Once $\mathbf{b}$ is known, then this calculation takes only 3 adds and 3 multiplies, considerably less than working directly with the recurrence relation. (Part of this advantage is thrown away if one follows the CAGD habit of explicitly forming the vector $[x^3,\dots,x^0]$.) But this gain comes at the cost of computing $\mathbf{b}$ from $\mathbf{a}$, hence pays off only if evaluation at more than just one x is needed.

In `Matlab`, this calculation can be carried out simultaneously for a whole vector or even a matrix x.

If we need $f(x)$ for $x \in [j\,..\,j+1)$ for some $j \neq 0$, then the needed changes are quite minor: the relevant coefficient segment becomes $(a_{j-3},\dots,a_j)$ and we would replace x by $x-j$, i.e., we would, in effect, write f on $[j\,..\,j+1)$ in **local power form**:

$$f(x) = \sum_{r=1}^{k} (x-j)^{k-r} b_r.$$

In particular, the matrix C remains unchanged, but this is a special feature of using the *uniform* knot sequence $\mathbb{Z}$.

For an *arbitrary* knot sequence $\mathbf{t}$ and arbitrary order k, the evaluation of $f = \sum_i a_i B_i$ at some $x \in [t_j\,..\,t_{j+1})$ still can be handled this way, i.e., by writing

$$\begin{aligned} f(x) &= [p_{j-k+1}(x),\dots,p_j(x)](a_{j-k+1},\dots,a_j] \\ &= [(x-t_j)^{k-1},\dots,(x-t_j)^0]\, C\, (a_{j-k+1},\dots,a_j). \end{aligned} \tag{0.2}$$

But now the change-of-basis matrix C depends on j and $\mathbf{t}$ as well as on k. There are several ways in use (and knot insertion is one of them) to carry out such a conversion, from the **B-form** for f to the corresponding **ppform**. The latter describes f in terms of its **breaks** $(\xi_i : i)$, i.e., the largest strictly increasing subsequence of the knot sequence $\mathbf{t}$, and the coefficient matrix c for its **local power form**, i.e.,

$$f(x) = \sum_{r=1}^{k} (x-\xi_j)^{k-r} c_{j,r}, \quad \xi_j \le x < \xi_{j+1}, \quad \forall j.$$

 ©2000 Carl de Boor

E.g., the command `ppB = fn2fm(spmak(0:4,1),'pp')` provides the ppform of the cardinal cubic B-spline, and `fnbrk(ppB,'coe')` provides the entries for the matrix C appearing in (0.1), though differently ordered.

lecture 26apr00: an aside: the ppform, continued

Here is a formal description of the two forms for describing splines in `Matlab`. A specific example for each is given below.

The **B-form** describes $f \in \$_{k,\mathbf{t}}$ in terms of its
knots $\mathbf{t} = (t_1 \le \cdots < t_{n+k})$
coefficients $a = (a_i : i = 1{:}n)$
degrees of freedom $n =$ `length(a)` $=$ `length(t)` - order
order k.

The **ppform** describes $f \in \Pi_{k,\boldsymbol{\xi}}$ in terms of its
breaks $\boldsymbol{\xi} = (\xi_1 < \cdots < \xi_{\ell+1})$
coefficients $(c(i,r) : i = 1{:}\ell, r = 1{:}k)$
number of pieces ℓ
order k
in the sense that

$$f(x) = \begin{cases} \sum_{r=1}^{k} (x-\xi_1)^{k-r} c(1,r), & x < \xi_2; \\ \sum_{r=1}^{k} (x-\xi_i)^{k-r} c(i,r), & \xi_i \le x < \xi_{i+1},\ i = 2{:}\ell{-}1; \\ \sum_{r=1}^{k} (x-\xi_\ell)^{k-r} c(\ell,r), & \xi_\ell \le x. \end{cases}$$

In other words, outside its **basic interval**, $[\xi_1 .. \xi_{\ell+1}]$, such a piecewise polynomial function is defined by extension of its first, respectively its last, polynomial piece. Correspondingly, neither ξ_1 nor $\xi_{\ell+1}$ is an active break.

We are usually not interested in the whole space $\Pi_{k,\boldsymbol{\xi}}$, but in its subspaces

$$\Pi_{k,\boldsymbol{\xi},\boldsymbol{\nu}} := \{f \in \Pi_{k,\boldsymbol{\xi}} : \operatorname*{jump}_{\xi} D^{r-1} f = 0, r = 1{:}\nu_i, i = 2 : \ell\}.$$

On the basic interval $[\xi_1 \,..\, \xi_{\ell+1}]$, this space coincides with the spline space $\$_{k,\mathbf{t}}$, with $\mathbf{t}$ the nondecreasing sequence that contains ξ exactly $k - \nu_i$ times, $i = 1{:}(\ell+1)$, and $\nu_1 = 0 = \nu_{\ell+1}$. In other words, each such piecewise polynomial space has a B-spline basis. This is a reiteration of B-spline property (vi).

The `Matlab` command `spline(x,y)` constructs a cubic spline, f, that interpolates the given data in the sense that $f(x(i)) = y(i)$ for all relevant i. It returns this 4th order spline in ppform, as is evident from the following example:

```
x = linspace(0,2*pi,5); y = sin(x);
cs = spline(x,y)
cs =
form:  'pp'
breaks:  [0 1.5708 3.1416 4.7124 6.2832]
coefs:  [4x4 double]
pieces:  4
```

 ©2000 Carl de Boor

```
order:  4
dim:  1
coefs = fnbrk(cs,'coe')
coefs =
0.0860 -0.8106 1.6977 0
0.0860 -0.4053 -0.2122 1.0000
0.0860 -0.0000 -0.8488 0.0000
0.0860 0.4053 -0.2122 -1.0000
```

Notice the 5 breaks and, correspondingly, the four polynomial pieces, as evidenced by the four rows of coefficients. If you take in that the ith row describes the coefficients for the ith piece, on the interval $[\xi_i \,..\, \xi_{i+1}] = [(i-1)\pi/2 \,..\, i\pi_2]$, and that the coefficients are ordered from highest to lowest, then you can read off that this spline has the value 0 at 0, 1 at $\pi/2$, 0 at π, and -1 at $3\pi/2$ (as it should, since these are the values the sine function takes at those points).

The command `fn2fm` provides conversion from one form to another. Here is our cubic interpolating spline written in an appropriate B-form:

```
bs = fn2fm(cs,'B-')
bs =
form:  'B-'
knots:  [0 0 0 0 6.2832 6.2832 6.2832 6.2832]
coefs:  [-6.2172e-015 3.5556 -3.5556 -5.5511e-016]
number:  4
order:  4
dim:  1
```

Note that the first and the last knot appear with 'full multiplicity', i.e., 4-fold. Note also that there are no interior knots. This is due to the fact that this particular spline has no active breaks (and the `fn2fm` command was able to detect and act on that). Here is the explanation.

The cubic spline interpolant f provided by `spline(x,y)` to data $((x_i, y_i) : i = 1 : \ell{+}1)$ is constructed as a C^2 piecewise cubic, hence satisfies 3 smoothness conditions across each interior break, with breaks at the data sites x_i. Hence

$$f \in \Pi_{4,\mathbf{x},\boldsymbol{\nu}},$$

with $\nu_i = 3$, $i = 2{:}\ell$. But, it so happens that, with this choice of $\boldsymbol{\nu}$, the linear space $\Pi_{4,\mathbf{x},\boldsymbol{\nu}}$ has dimension $\ell + 3$, i.e., two more degrees of freedom than there are data points. There are many different ways to make use of these two extra degrees of freedom (look at cubic spline interpolation in `splinetool`, under **end conditions**). The `spline` command uses the so-called **not-a-knot** conditions, which amount to choosing $\nu_2 = 4 = \nu_\ell$ and so make x_2 and x_ℓ inactive breaks and, correspondingly, $\dim \Pi_{4,\mathbf{x},\boldsymbol{\nu}} = \ell + 1 = \#\mathbf{x}$.

With $\boldsymbol{\nu}$ so chosen, the B-spline basis for $\Pi_{4,\mathbf{x},\boldsymbol{\nu}}$, as per the earlier recipe, has as its knot sequence

$$(x_1, x_1, x_1, x_1, x_3, \ldots, x_{\ell-1}, x_{\ell+1}, x_{\ell+1}, x_{\ell+1}, x_{\ell+1}).$$

For our specific example, this reduces to the sequence

$$(0, 0, 0, 0, \pi, 2\pi, 2\pi, 2\pi, 2\pi).$$

However, as is evident from the coefficients in the ppform, the third derivative of the interpolant to our particular data has no jump across π (and shouldn't, since the problem is antisymmetric around that point, hence we must have $f(\pi + s) = -f(\pi - s)$, hence $D^3 f(\pi+) = -(-1)^3 D^3 f(\pi-) = D^3 f(\pi-)$). Therefore, also the break π is inactive for this particular f, and the command `fn2fm` was able to detect and act on that, too.

Note also that, in this example, the ppform involves 16 coefficients, while the B-form involves only 4, a remarkable amount of *compression*. To be sure, we could have represented this f as one cubic piece, requiring also only 4 coefficients. In general, though, a C^2 piecewise cubic function with ℓ interior breaks involves 4ℓ coefficients, while the corresponding B-form only involves $3 + \ell$ coefficients. It is in this sense, that the B-form is much more efficient than the ppform. On the other hand, evaluation from the ppform is much faster than from the B-form. Hence, if the spline is to be evaluated at many points, conversion to ppform would be indicated. Of course, `cs = spline(x,y)` already delivers its answer in ppform, and this is taken advantage of in the evaluation command `ppval(cs,xx)` that replaces each entry of `xx` by the value at that entry of the spline described by `cs`.

Finally, notice that both forms have an additional parameter, labeled `dim` in the examples given. In both examples, this number is 1, corresponding to the fact that the spline in question is *scalar-valued*. The command `spline` is also prepared to deal with *vector-valued* functions. Have a look at the last example in the help from `spline`, obtainable by the command `help spline`.

lecture 28apr00: knot insertion

Recall B-spline property (xi). *If $\widehat{\mathbf{t}}$ is obtained from $\mathbf{t}$ by insertion of the point x, and, correspondingly,*

$$\widehat{B}_{jk} := B_{j,k,\widehat{\mathbf{t}}}\,,$$

then

$$\sum_j a_j B_{jk} = f = \sum_j \widehat{a}_j \widehat{B}_{jk}, \tag{0.1}$$

with

$$\widehat{a}_j = \widehat{\omega}_{jk}(x) a_j + (1 - \widehat{\omega}_{jk}(x)) a_{j-1},$$

where

$$\widehat{\omega}_{jk}(x) := \max\{0, \min\{1, \omega_{jk}(x)\}\}.$$

Recall also the *control polygon* for $f = \sum_j a_j B_{j,k,\mathbf{t}}$ as the broken line with vertices the *control point*s

$$P_j = (t^*_{jk}, a_j) \in \mathbb{R}^2,$$

i.e., the coefficients of the spline *curve*

$$x \mapsto (x, f(x)) = \sum_j P_j B_{j,k,\mathbf{t}}(x)$$

that is the graph of f. It follows that

$$\sum_j P_j B_{j,k,\mathbf{t}} = \sum_j \widehat{P}_j \widehat{B}_{jk},$$

with

$$\widehat{P}_j = \widehat{\omega}_{jk}(x) P_j + (1 - \widehat{\omega}_{jk}(x)) P_{j-1}.$$

This shows that the refined control polygon has its vertices on the control polygon we started with.

This is worth a picture. I'll make up a (cubic) spline in B-form consisting of just one cubic piece, pretty much like the interpolant to sin, i.e., with B-spline coefficients $(0, 32/9, -32/9, 0)$, except that I'll rescale the interval of interest to be $[0\,..\,4]$. Then I'll insert the point 1 three times into the knot sequence, each time plotting the control polygon a little bit thicker. I also circle the final set of control points:

```
k = 4; sp = spmak(augknt([0 4],k),[0 32/9 -32/9 0]);
fnplt(sp), hold on
plot(aveknt(sp.knots,k),sp.coefs,'-r','linew',.5)
sp = fnrfn(sp,1); plot(aveknt(sp.knots,k),sp.coefs,'-r','linew',2)
sp = fnrfn(sp,1); plot(aveknt(sp.knots,k),sp.coefs,'-r','linew',4)
sp = fnrfn(sp,1); plot(aveknt(sp.knots,k),sp.coefs,'-w','linew',2)
plot(aveknt(sp.knots,k),sp.coefs,'ok'), hold off
```

Note that the refined control points lie exactly where they are supposed to lie: Since 1 is the quarter-point on the way from $0 = t_j$ to $4 = t_{j+k-1}$, each new $\widehat{P}_j$ lies at the quarter-point from P_{j-1} to P_j.

Also, notice that the initial (and the final) slope of f and its control polygon coincide (why??).

Also, the second time, only the first part of the control polygon changes.

Finally, on the third time, the polygon doesn't change; it only acquires one more vertex, exactly at the point where the control polygon touches the spline. This is not surprising because, by this time, the number 1 is a three-fold knot for this cubic spline, hence there is only one cubic B-spline not zero at that point and its coefficient therefore necessarily equals f(1).

In general, *if $\#t_{j+1} = k - 1$, then B_{jk} is the only $B_{i,k,\mathbf{t}}$ not zero at $x := t_{j+1}$, hence necessarily has the value 1 there and, correspondingly, $a_j = f(x)$.* In other words, we can use repeated knot insertion as a way to evaluate splines.

This fact together with the fact that the refined control polygon has its vertices on the original control polygon is at the basis of B-spline property (xii): *If $\sum_j a_j B_j$ and $\tau_1 < \cdots < \tau_r$ are such that $f(\tau_{i-1}) f(\tau_i) < 0$ for all i, then there exist indices $j(1) < \cdots < j(r)$ so that*

$$f(\tau_i) a_{j(i)} B_{j(i)}(\tau_i) > 0, \quad i = 1{:}r.$$

This property is called **variation diminution** since it says, offhand, that the spline cannot oscillate any more times than its control polygon.

But it says much more than that. Since, by B-spline property (v), any linear polynomial is its own control polygon, it says that a spline cannot oscillate across any linear polynomial any more times than its control polygon does.

In particular, if the control polygon is monotone, so is the spline. If the control polygon is convex, so is the spline. This property is also called **shape-preserving**, and the wide use of splines in CAGD is almost entirely due to this property. For it makes it easy to construct a smooth spline curve of a desired shape simply by sketching that shape as a control polygon and then use the corresponding spline.

As an example, take the corners of the unit square as control points, i.e., consider the spline curve

$$x \mapsto \sum_{j=1}^{5} P_j B_{j,k,\mathbf{t}},$$

with

$$P = \begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 \end{bmatrix},$$

hence $\mathbf{t}$ any knot sequence of length $5+k$. To be definite, take $k = 4$, hence $\mathbf{t}$ should have length 9. If we also want the first and last knot have multiplicity $k = 4$, that leaves us with just one interior knot. E.g.,

```
t = [0 0 0 0 1 2 2 2 2];
a = [1 -1 -1 1 1; ...
1 1 -1 -1 1];
sp = spmak(t,a);
fnplt(sp), hold on, plot(a(1,:),a(2,:),'-o'), hold off, axis equal, axis
off
```

Because of the end knots of multiplicity k, the curve begins at P_1 and ends at $P_5 (= P_1)$.

But don't be misled by the terms *control* point and *control* polygon. As my performance at the end of class showed, it is not all that easy to produce a curve of some desired shape by playing with the control points (contrary what some books might want you to believe). The *construction* of a spline with desired properties (i.e., the *reconstruction* of a spline from information about it) is best done by direct interpolation or other fitting of a suitable spline model to specific data, our next (and final) topic.

lecture 03may00: spline interpolation

We are now ready to use splines to recover (not necessarily perfectly) a function from some information about it, using splines.

To be specific, we choose some real interval I, some sequence $\tau = (\tau_1 < \cdots < \tau_n)$ in I, and consider the data

$$\Lambda'_\tau g := (g(\tau_i) : i = 1{:}n)$$

for some real-valued function g on I, i.e., some

$$g \in X := \mathbb{R}^I.$$

We also choose some linear subspace $F \subset X$, of dimension n, i.e., with some basis $V : \mathbb{R}^n \to F$, and assume that the Gramian matrix $\Lambda_\tau' V$ is invertible. Then we know that

$$f = P_\tau g := V(\Lambda_\tau' V)^{-1} \Lambda_\tau' g$$

is the unique element of F that matches the given information about g, i.e., satisfies

$$\Lambda_\tau' f = \Lambda_\tau' g.$$

The classical choice for F is $\Pi_{<n}$, the space of polynomials of degree $< n$. As discussed earlier, $\Lambda_\tau' V$ is invertible for any choice of basis V for this space. However, there are some problems, as the following calculations show.

Take, specifically, $I = [-1 \,..\, 1]$, and choose

$$\tau = (-1 = \tau_1 < \cdots < \tau_n = 1)$$

uniformly spaced (i.e., $\tau_i = -1 + 2(i-1)/(n-1)$, $i = 1{:}n$). Also choose the **Runge example**

$$g : x \mapsto \frac{1}{1 + (5x)^2}.$$

We can compute and plot the polynomial interpolant with the aid of the command `spapi` as follows (choosing $n = 15$, to be specific):

```
g = inline('1./(1+(5*x).^2)');
xx = linspace(-1,1,101); plot(xx,g(xx)), hold on
n = 15; tau = linspace(-1,1,n); pol = spapi(n,tau,g(tau));
fnplt(pol,'k')
```

Near the end points, this polynomial interpolant overshoots the function g by more than 7.

For comparison, we also construct and plot a cubic spline interpolant to the same data:

```
cs = spapi(4,tau,g(tau)); fnplt(cs,'r'), hold off
```

It agrees with g to graphic accuracy.

What is going wrong with the polynomial interpolant? The main difficulty appears to be that the polynomial interpolant can be much bigger than the function it interpolates.

To make this precise, we introduce the *size* or *norm* of a function:

$$\|g\| := \sup_{x \in I} |g(x)|, \qquad g \in X = \mathbb{R}^I.$$

Further, we introduce

$$\|P\| := \sup_{g \in X} \frac{\|Pg\|}{\|g\|}$$

as a measure of the worst-case scenario, i.e., the biggest possible ratio of the size of Pg compared to the size of g. The larger $\|P\|$, the bigger can be the difference between some g and its interpolant.

As it turns out, there is a convenient formula for $\|P\|$, namely

$$\|P\| = \sup_{x\in I} \sum_{j=1}^{n} |\ell_j(x)|,$$

with $[\ell_1, \ldots, \ell_n]$ the basis for F dual to Λ'_τ (also called the Lagrange basis), i.e.,

$$\ell_i(\tau_j) = \delta_{ij}.$$

This means that we can estimate $\|P\|$ quite closely in the following way, first for polynomial interpolation and then for cubic spline interpolation:

```
max(sum(abs(fnval(spapi(n,tau,eye(n)),xx))))
ans = 283.1809
max(sum(abs(fnval(spapi(4,tau,eye(n)),xx))))
ans = 1.9698
```

This shows a great difference. Moreover, one can work out that, as n increases, $\|P\|$ grows like $\exp(n/2)$ for the polynomial case, while, for cubic spline interpolation, it stays around 2 regardless of n.

This very different behavior is a main reason we use splines rather than just one polynomial when trying to recover a function from function values. But this is not the whole story.

Have a look at the *error* $g - Pg$ in our interpolant Pg to g. Since $Pg \in F$, $\|g - Pg\|$ cannot be any smaller than the **distance**

$$\operatorname{dist}(g, F) := \inf_{f\in F} \|g - f\|$$

of g from F. On the other hand, since $\|P\|$ is the supremum over all ratios $\|Pg\|/\|g\|$, we know that

$$\|Pg\| \le \|P\|\|g\|, \quad g \in X.$$

We also know that $Pf = f$ for all $f \in F$, and that P is linear, hence $g - Pg = g - f + Pf - Pg = g - f + P(f - g)$. Therefore,

$$\|g - Pg\| \le \|g - f\| + \|P\|\|f - g\| = (1 + \|P\|)\|g - f\|,$$

and this holds for every $f \in F$. By choosing f to make $\|g - f\|$ as small as possible, we obtain

Lebesgue's inequality. *For all* $g \in X$,

$$\operatorname{dist}(g, F) \le \|g - Pg\| \le (1 + \|P\|) \operatorname{dist}(g, F).$$

In particular, if our interpolation process P has a modest norm, then the interpolation error $\|g - Pg\|$ is within a modest factor of the best possible error.

This shows that there are two aspects to good recovery (by interpolation or otherwise):

(1) The recovery process should have a modest norm, and that is often easily achievable with splines (and not so easy with polynomials).
(2) The distance $\operatorname{dist}(g, F)$ of g from F should be small. This, too, is often easily achievable with splines, simply by making the knot spacing sufficiently small or otherwise placing the knots appropriately, while, for polynomials, the only possibility is to make the degree large, and that leads to many difficulties.

lecture 05may–: spline interpolation (cont)

With $\tau = (\tau_1 < \cdots < \tau_n)$ in the interval $I = [a\,..\,b]$ arbitrary, the data map

$$\Lambda'_\tau : f \mapsto (f(\tau_i) : i = 1{:}n)$$

is invertible on $F = \Pi_{<n}$ for any such τ. This cannot be true for $F = \$_{k,\mathbf{t}}$. For example, if some knot interval contains more than k of the τ_i, then surely we cannot match arbitrary values there since, on such an interval, each element of F is only a polynomial of degree $< k$.

The coefficient vector a of the interpolant $Pg = \sum_j a_j B_{jk}$, if any, must satisfy the linear system

$$A_\tau a = \Lambda'_\tau g,$$

with

$$A_\tau := \Lambda'_\tau[B_{j,k,\mathbf{t}} : j = 1{:}n] = (B_{j<k,\mathbf{t}}(\tau_i) : i, j = 1{:}n).$$

Hence it all depends on whether or not this Gramian matrix is invertible. Remarkably, there is a simple characterization for its invertibility.

Schoenberg-Whitney Theorem. *Let $\tau = (\tau_1 < \cdots < \tau_n)$ and $\mathbf{t} = (t_1 \le \cdots \le t_{n+k})$. Then the Gramian matrix A_τ is invertible if and only if all its diagonal entries are nonzero, i.e.,*

$$B_{i,k,\mathbf{t}}(\tau_i) \neq 0, \quad i = 1{:}n.$$

Moreover, when A_τ is invertible, then it is a banded matrix, with at most $2k+1$ nontrivial bands (consecutive), hence its numerical solution is relatively cheap. More than that, the matrix is *totally positive* (whatever that may mean; look it up in the book), and this has as a consequence that the linear system can be solved by Gauss elimination *without any pivoting*, another savings in work and storage requirements. I hope you begin to appreciate just why B-splines are so wonderful.

There may be some doubt in the Schoenberg-Whitney Theorem in case τ_i equals some knot of multiplicity k, in which case $f \in \$_{k,\mathbf{t}}$ may be discontinuous at τ_i, hence has, in effect, *two* values at τ_i, namely the limit from the left and the limit from the right. In particular, by our way of choosing the knot sequence to work for a particular interval, the first and the last knot has that multiplicity. In such a case, you need to specify, more precisely, whether you want to match the limit from the left or the limit from the right. In particular, you would take the limit from the right at the first knot and the limit from the left at the last knot, and this is done automatically by the command `spapi`, which always takes the limit from the right except at the last knot where it takes the limit from the left.

So, how does the command `spapi(k,tau,g(tau))` choose the knot sequence for the interpolating spline of order k it outputs? It obtains the knot sequence as `aptknt(tau,k)`, and this, for $k > 1$, is simply the output from

`augknt([tau(1),aveknt(tau,k),tau(end)] ,k)`

meaning that it constructs the vector

$$\tau^* := (\tau^*_{jk} : j = 1{:}n{-}k),$$

and appends to it both the first and the last data site with multiplicity k. *Do verify that the resulting knot sequence does satisfy the above Schoenberg-Whitney conditions wrto τ.*

For example, if $k = 2$, then $\tau^* = (\tau_i : i = 2{:}n{-}1)$, hence the knot sequence used is

$$\mathbf{t} = (\tau_1, \tau_1, \tau_2, \tau_3, \ldots, \tau_{n-1}, \tau_n, \tau_n),$$

and the corresponding B-spline sequence is quite simply the sequence in $\$_{k,\mathbf{t}}$ dual to Λ'_τ, i.e., $B_{j,2,\mathbf{t}}(\tau_i) = \delta_{ij}$.

What happens when $k = 3$? Then

$$\tau^*_{j3} = (\tau_{j+1} + \tau_{j+2})/2, \quad j = 1{:}, n-2,$$

hence the knots will lie between the data sites. In particular, the data sites will be at the midpoints between the knots if τ is uniformly spaced.

As another example, if $k = 4$ and τ is uniformly spaced, then $\tau^* = (\tau_3, \ldots, \tau_{n-2})$, hence the knot sequence used is

$$\mathbf{t} = (\tau_1, \tau_1, \tau_2, \tau_3, \ldots, \tau_{n-1}, \tau_n, \tau_n).$$

In particular, the resulting cubic spline interpolant has a break or knot at every data site except the second and the second-last and so satisfies what is known as the **not-a-knot end condition**.

Finally, if $k = n$, then τ^* is empty and, correspondingly, the knot sequence has both τ_1 and τ_n in it k times and nothing else. Correspondingly, $\$_{k,\mathbf{t}} = \Pi_{<n}$ (on the interval $[\tau_1 \,..\, \tau_n]$).

Next, we discuss **Hermite** or **osculatory** interpolation, as follows. You know that, for any continuously differentiable function,

$$\lim_{y \to x} \frac{g(y) - g(x)}{y - x} \;=\; Dg(x).$$

Hence, if all else is ok, we would expect our interpolant Pg to satisfy not only $(Pg)(\tau_i) = g(\tau_i)$ but also, in the limit as $\tau_{i-1} \to \tau_i$,

$$DPg(\tau_i) = Dg(\tau_i).$$

If also $\tau_{i-2} \to \tau_i$, then we would also expect, in the limit, to have

$$D^2Pg(\tau_i) = D^2g(\tau_i).$$

For this reason, it is reasonable to define the data map Λ'_τ even in the limiting case that τ is merely nondecreasing, i.e.,

$$\tau = (\tau_1 \le \cdots \le \tau_n),$$

in the following way:

$$\Lambda'_\tau : g \mapsto (D^r g(\tau_i) : \tau_{i-r-1} < \tau_{i-r} = \cdots = \tau_i; i = 1{:}n).$$

The command `spapi(k,tau,y)` interprets multiplicities in the data site sequence $\tau :=$ `tau` in exactly this way, i.e., interprets `y` as the information $\Lambda'_\tau g$ to be matched, with the following convenient feature. If the input `tau` is not already ordered, it will be re-ordered to make it nondecreasing and the input `y` will be re-ordered in the same way. Hence, if you would like to match the values of g at all points of a some sequence `x` without any multiplicities and also would like to match the first derivative at some subsequence `x1` of `x`, then the command

```
spapi(k, [x x1], [g(x) dg(x1)]);
```

would accomplish this (provided the desired order `k` is at least 3 (to make certain that the splines are continuously differentiable), and provided that `dg` provides values of the first derivative of g).

lecture 08-10may: spline approximation

When the data

$$((\tau_i, y_i) : i = 1{:}N) \tag{0.1}$$

are not exact, e.g., y_i may only be an approximation to $g(\tau_i)$, or there are more data than needed to recover $g \in X$ to the desired accuracy, one does not try to match the data exactly, but simply finds an 'approximate interpolant' from some subspace F that has $\dim F < N$ but can be expected to recover the 'essential part' of the information about g in the data.

There are at least two standard ways to proceed, *least-squares approximation* and *smoothing*.

Look at **discrete least-squares approximation from** $F \subset X = \mathbb{R}^I$ for some interval I containing the sequence τ. With V a basis for F, look again at the linear system

$$\sum_j v_j(\tau_i) a_j = y_j, \quad j = 1{:}N,$$

for the coordinate vector $\mathbf{a}$ for the 'interpolant' $V\mathbf{a}$ from F to the data. If the coefficient matrix

$$A := A_\tau := \Lambda'_\tau V$$

of this linear system is invertible, then there is a unique interpolant, and we get its coefficients just by solving the linear system. In `Matlab`, this amounts to the command

```
a = A;
```

which is very user-friendly.

Remarkably, the very same command also suffices for computing the discrete least-squares approximant from F to these data. This is, by definition, the unique element $P_F\mathbf{y} = V\mathbf{a} \in F$ that minimizes

$$\|\mathbf{y} - f\|_2^2 := \sum_i (y_i - f(\tau_i))^2$$

over all $f \in F$.

To be sure, the asserted *uniqueness* requires the assumption that the Gramian matrix $A = \Lambda_\tau' V$ is 1-1; in particular, this requires that it have at least as many rows as it has columns and be *of full rank*, i.e., its column sequence be linearly independent. But, with that assumption, the matrix $A'A$ is also 1-1 and, since it is square, therefore it is invertible, hence the so-called **normal equations**

$$A'A\mathbf{a} = A'\mathbf{y}$$

have a unique solution. These equations are so called because they identify $\mathbf{c} := A\mathbf{a}$ as the unique element of $\operatorname{ran} A$ for which $\mathbf{y} - A\mathbf{a}$ is perpendicular to $\operatorname{ran} A$, hence for which $\|\mathbf{y} - \mathbf{c}\|_2$ is minimized. In formula,

$$A\mathbf{a} = A(A'A)^{-1}A'\mathbf{y}$$

is the element in $\operatorname{ran} A$ closest to $\mathbf{y}$. Do appreciate the by now familiar structure here: $A\mathbf{a}$ appears as the element of the linear space $\operatorname{ran} A$ recovered from the data $A'\mathbf{y} = (\langle \mathbf{y}, A(:,i)\rangle : i = 1{:}n)$ about $\mathbf{y}$ as written in terms of the basis A for $\operatorname{ran} A$. From this,

$$P_F\mathbf{y} = V(A'A)^{-1}A'\mathbf{y}$$

provides the discrete least-squares approximant to the given data from F. However, for stability reasons, `Matlab` does not actually solve the normal equations when it constructs the vector —A‖**y**.

Here is an example that further exercises your growing understanding of the use of bases. Suppose that we would like to compute the discrete least-squares approximant to the data $\mathbf{y} \sim \Lambda_\tau' g$ by C^2-cubic splines with break-sequence $\boldsymbol{\xi} = (\xi_1 < \cdots < \xi_{\ell+1})$ and $I = [\xi_1 \,..\, \xi_{\ell+1}]$. Then our F is $S_{4,\mathbf{t}}$, with

$$\mathbf{t} = (t_1 \le \cdots \le t_{n+4}) = (\xi_1, \xi_1, \xi_1, \boldsymbol{\xi}, \xi_{\ell+1}, \xi_{\ell+1}, \xi_{\ell+1})$$

(why?). In particular,

$$n = \ell + 3.$$

Hence, the command —spap2(augknt(—$\boldsymbol{\xi}$—,4),4,—τ—,y)— would provide the desired least-squares approximant, using the B-spline basis. But —spap2— is not part of plain —Matlab—, so, what to do?

Recall that plain —Matlab— does provide a cubic spline interpolant. Precisely, with —xi— = $\boldsymbol{\xi}$ and $\mathbf{z}$ an arbitrary *row* vector with n entries, the `Matlab` command

`cs = spline(xi,z);`

provides (the ppform of) a C^2-cubic spline $f = P_{\boldsymbol{\xi}}\mathbf{z}$ with break-sequence $\boldsymbol{\xi}$, i.e., $f \in F$, that matches the data

$$\Lambda' f := (Df(x_1), f(x_1), f(x_2), \ldots, f(x_{n-1}), f(x_n), Df(x_n)) = \mathbf{z}.$$

($P_{\boldsymbol{\xi}}\mathbf{z}$ is known as a **complete cubic spline interpolant** to the data $\mathbf{z} = \Lambda' f$.) In particular,

$$\Lambda' P_{\mathbf{x}} = \mathrm{id}\,.$$

This implies that $\Lambda'|_F$ maps F *onto* $\mathbb{R}^n$, hence, since $\dim F = n$, $\Lambda'|_F$ must also be 1-1, hence invertible. But since $P_{\boldsymbol{\xi}}$ is a right inverse for $\Lambda'|_F$, it must be the inverse for it and, in particular, $P_{\boldsymbol{\xi}}$ must be *linear*, 1-1, and onto F, hence a basis for F. It is this basis for F we now will work with.

Set $[v_1, \ldots, v_n] = V := P_{\boldsymbol{\xi}}$. Then, for $j = 1{:}n$, —vj= spline(x,—$\mathbf{i}_j$—)— is the jth term in this basis, with

$$\mathbf{i}_j = [\underbrace{0, \ldots, 0}_{j-1 \text{ terms}}, 1, 0, 0, \ldots] \in \mathbb{R}^n$$

the jth unit vector of length n written as a row-vector, i.e., the jth row of the identity matrix of order n. Since —spline(xi,z,xx)— is set up to return the values at —xx— of the function —spline(xi,z)—, we therefore have

$(v_j(\tau_1), \ldots, v_j(\tau_N)) =$ —spline(xi,—$\mathbf{i}_j$—,tau)—

provided —tau— is the row-vector containing τ. More than that, —spline— is set up to work on *vector-valued* data. Specifically, if —z— is a $d \times n$-matrix, then —spline(xi,z)— is the d-vector valued function whose jth component is the complete cubic spline interpolant to the data —z(j,:)—. In particular, the command —spline(xi,eye(n), tau)— returns the $n \times N$-matrix whose jth row is the vector $(v_j(\tau_1), \ldots, v_j(\tau_N))$. In other words, —spline(xi,eye(n), tau)— is the *transpose* A' of the matrix $A = \Lambda'_\tau V$ we seek.

Since —(A)'— is the same as —y'/A'—, it follows that —y'/spline(xi,eye(n), tau)— is the row-vector containing the coefficients $\mathbf{a} = (a_i)$ we seek, in order to put together the spline $V\mathbf{a} = P_{\boldsymbol{\xi}}\mathbf{a}$, hence that spline is obtained by the *single* command

`l2 = spline(xi,y/spline(xi,eye(n),tau));`

The idea behind the above construction can be used to construct a least-squares approximant from the range of any linear recovery map P for which one has a computer implementation.

Note again that, strictly speaking, —f = spline(xi,z)— is not the spline recovered from the data —z—, but only a representation for it from which, presumably, we can obtain other information about it; e.g., —ppval(f,tau)— derives from that representation the values of that spline at the entries of —tau—. Hence recovery is, strictly speaking, just a *change* in representation.

quick review of spline part Spring'00

 ©2000 Carl de Boor

Focus was on recovery (exact or partial) of a function g in some linear space X of functions from some data $\Lambda' g$ about it, with $\Lambda' : g \mapsto (\lambda_i g : i \in I)$ a linear map into the coordinate space $\mathbb{F}^I$. (The discussion involved the terms: linear, 1-1, onto, invertible, range, kernel, linear (sub)space, basis, coordinate space, data map, analysis operator, recovery map, synthesis operator, and the totally nonstandard term 'column map'.)

If the recovered function is to be in the linear subspace F with basis W, then $f = Pg = W(\Lambda' W)^{-1}\Lambda' g$ is the unique element of F that matches the given information (i.e., $\Lambda' f = \Lambda' g$), provided the Gramian matrix $\Lambda' W = (\lambda_i w_j)$ is invertible.

In that case, also $V := W(\Lambda' W)^{-1}$ is a basis for F, and $\Lambda' f$ are the coordinates of f wrto that basis. Hence our recovery amounts to nothing more than a change of basis. But such a change of basis is useful if the information about f (or g) we are after is more readily obtainable from $(\Lambda' W)^{-1}\Lambda' f$ than from $\Lambda' f$.

This raises the basic question of which information about $f \in F$ is readily available from its coordinates wrto a given basis for F.

We explored this question for $F = \Pi_{<n}$, comparing Lagrange basis and power basis.

We explored this question in very great detail for F a space of more or less smooth piecewise polynomial functions, with special focus on its B-spline basis (but also some mention of the ppform, which employs a basis for the space $\Pi_{<k,\boldsymbol{\xi}}$ of all piecewise polynomials of order k with break sequence $\boldsymbol{\xi}$).

Specifically, we discussed altogether 12 (of course!) B-spline properties, including the recurrence relation used in their definition, their support, number of nontrivial polynomial pieces, conventions and formula concerning their differentiation, Marsden's Identity, the recipe for constructing the B-spline basis for $\Pi_{k,\boldsymbol{\xi},\boldsymbol{\nu}}$, the Schoenberg-Whitney theorem, the convex hull property, the dual functionals, local and nonnegative partition of unity, knot insertion (refinability), control points and polygons, and variation diminution.

While there was some discussion concerning the construction of a spline, especially a spline curve, by explicit specification of its control points, we mainly discussed construction of a spline by interpolation to some data, including the proper choice of knot sequence for that, and finished (much too early :-) with a discussion of least-squares spline approximation, stressing again the underlying maps.

All details are available in the online lecture notes.

 ©2000 Carl de Boor