cs 514, 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**, $[\xi_1 \mathinner{..} \xi_{\ell+1}]$, 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 $[\xi_1 \mathinner{..} \xi_{\ell+1}]$, 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
     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 $[\xi_i \mathinner{..} \xi_{i+1}] = [(i-1)\pi/2 \mathinner{..} i\pi_2]$, 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`.

