last change: 10oct00

piecewise cubic interpolation

Cubic Hermite interpolation provides a nice occasion to admire the power of divided differences. As the script `ShowHermite` so nicely shows, coalescence of interpolation points leads to osculation, i.e., to matching of derivative values.

In particular, the Newton form

$$P_4(x) = f[x_1] + (x - x_1)(f[x_1, x_2] + (x - x_2)(f[x_1, x_2, x_3] + (x - x_3)f[x_1, \ldots, x_4]))$$

of the cubic interpolant to f at the four points x_1, x_2, x_3, x_4 makes good sense even when $x_1 = x_2$ and $x_3 = x_4$ provided only that we explain properly what, e.g., $f[x_1, x_1]$ might be. But that is easy since we know from Calculus that $\lim_{x_2 \to x_1} f[x_1, x_2] = f'(x_1)$. To keep the notation simple, I'll write out everything for the special case that $x_1 = 0 = x_2$, $x_3 = h = x_4$. The divided difference table for f at this four-point sequence is

$$\begin{array}{lllll}
x & f[] & f[,] & f[,,] & f[,,,] \\
0 & f(0) & & & \\
 & & f'(0) & & \\
0 & f(0) & & f[0,0,h] & \\
 & & f[0,h] & & f[0,0,h,h] \\
h & f(h) & & f[0,h,h] & \\
 & & f'(h) & & \\
h & f(h) & & &
\end{array}$$

from which we derive the cubic Hermite polynomial matching f in value and slope at both 0 and h to be

$$(1) \qquad q(x) = f(0) + x(f'(0) + x(f[0,0,h] + (x - h)f[0,0,h,h])).$$

Since $hf[0,0,h,h] = f[0,h,h] - f[0,0,h]$, this takes the power form

$$q(x) = f(0) + x(f'(0) + x(2f[0,0,h] - f[0,h,h] + xf[0,0,h,h])).$$

Translation of this formula, from the interval $[0\,..\,h]$ to the interval $[x_i\,..\,x_{i+1}]$ of length $h_i := \Delta x_i$, provides a formula for the cubic polynomial q_i that matches value and slope of f at both x_i and x_{i+1}, as follows. Let `y`$= f($`x`$)$, `s`$= f'($`x`$)$ for some increasing sequence `x`, and set further `h=diff(x)`, `dy=diff(y)./h`$= (f[x_i, x_{i+1}] : i = 1{:}n{-}1)$. Then

$$f[x_i, x_i, x_{i+1}] = (dy_i - s_i)/h_i =: e_{i0}, \qquad f[x_i, x_{i+1}, x_{i+1}] = (s_{i+1} - dy_i)/h_i =: e_{i1},$$

hence

$$f[x_i, x_i, x_{i+1}, x_{i+1}] = (e_{i1} - e_{i0})/h_i =: d_i.$$

With this, we conclude that

$$(2) \qquad q(z) := q_i(z) := y_i + (z - x_i)(s_i + (z - x_i)(c_i + (z - x_i)d_i)), \quad x_i \le z \le x_{i+1},$$

with

$$c_i := 2e_{i0} - e_{i1}$$

is piecewise cubic with breaks at the x_i and matches f in value and slope at each x_i, $i = 1{:}n$. Our textbook prefers to write the cubic pieces in the form (1), but that prevents the use of `ppval` for the evaluation of this piecewise cubic.

```
function pc = pwch(x,y,s)
%
% Pre: x = strictly increasing sequence of length n
%      y, s = sequence of the same orientation and length as x
%
% Post: pc = a description, ready for use with ppval, of the piecewise
%      cubic function  q  that satisfies
%          q(x_i) = y_i,  q'(x_i) = s_i,  i=1:n .
h = diff(x); dy = diff(y)./h;
dzzh = (dy-s(1:end-1))./h; dzhh = (s(2:end)-dy)./h;
pc = mkpp(x,[(dzhh-dzzh)./h 2*dzzh-dzhh s(1:end-1) y(1:end-1)]);
```

Since q_i is the cubic interpolant to f at the four-point sequence $x_i, x_i, x_{i+1}, x_{i+1}$, the error formula for polynomial interpolation tells us that, for $x_i \le z \le x_{i+1}$,

$$f(z) - q(z) = f[x_i, x_i, x_{i+1}, x_{i+1}, z](z - x_i)^2(z - x_{i+1})^2.$$

On that interval, $|(z-x_i)^2(z-x_{i+1})^2| \le (\Delta x_i/2)^4$, while $f[x_i, x_i, x_{i+1}, x_{i+1}, z] = f^{(4)}(\eta)/4!$ for some η between x_i and x_{i+1}. Hence, altogether,

$$(3) \qquad |f(z) - q(z)| \le \max_{x_1 \le \eta \le x_n} |f^{(4)}(\eta)|/384 \ \ \max_i (\Delta x_i)^4.$$

the cubic spline

In piecewise cubic Hermite interpolation, we match values and slopes at the given data sites x_i, $i = 1{:}n$, and obtain a continuous function with continuous first derivative. However, it is not always easy to supply those slopes, while, at the same time, it turns out to be possible to so choose the numbers s_i for given $y = f(x)$ that the resulting interpolant, S, has even the second derivative continuous. Remarkably, the error bound available for this interpolant, called the **cubic spline interpolant**, is

$$(4) \qquad |f(z) - S(z)| \le \max_{x_1 \le \eta \le x_n} |f^{(4)}(\eta)|(5/384) \ \max_i (\Delta x_i)^4.$$

i.e., only five times as big as the one for the piecewise cubic Hermite interpolant which uses twice as much information about f.

From (2) and with $h_i := \Delta x_i$, $y_i' := \Delta y_i / h_i$, we compute that

$$(5) \ \ q_i''(x_i) = 2c_i = 2(2e_{i0} - e_{i1}) = 2(2(y_i' - s_i) - (s_{i+1} - y_i'))/h_i = 2(3y_i' - 2s_i - s_{i+1})/h_i.$$

Therefore, replacing here x_{i+1} by x_{i-1}, hence h_i by $-h_{i-1}$, s_{i+1} by s_{i-1} etc.,

$$q''_{i-1}(x_i) = 2(3y'_{i-1} - 2s_i - s_{i-1})/(-h_{i-1}). \tag{6}$$

Thus the requirement that $q''_{i-1}(x_i) = q''_i(x_i)$ is equivalent to the equation

$$h_i s_{i-1} + 2(h_{i-1} + h_i)s_i + h_{i-1}s_{i+1} = 3(h_{i-1}y'_i + h_i y'_{i-1}). \tag{7}$$

We get such an equation for each interior breakpoint, i.e., for $i = 2{:}n{-}1$. This gives us $n-2$ equations in the n unknowns $s_1, \ldots, s_n$. Two additional equations are needed to determine the n slopes uniquely. Here are some standard choices.

complete spline: Simply supply $s_i = f'(x_i)$ for $i = 1$ and $i = n$. That leaves us with $n-2$ unknowns to be determined by the $n-2$ equations (7), $i = 2{:}n{-}1$. Such a spline is also known as a **clamped** spline since we are prescribing its slope at the ends.

natural spline: Insist that $S''(x_1) = 0 = S''(x_n)$, i.e., that

$$3y'_1 - 2s_1 - s_2 = 0 = 3y'_{n-1} + 2s_n + s_{n-1}.$$

Such a spline is also known as a **free** spline, and both terms refer to the fact that this spline models (roughly) the behavior of a draftman's spline that is forced to go through the given data points but is not clamped at the ends. Don't be fooled by the term 'natural', though. If the function being interpolated doesn't have a vanishing second derivative at the ends, then near the ends, the error bound (4) is not valid. Under those circumstances, one does better by using one of the other end conditions discussed here, or by explicitly prescribing second derivatives. From (5), (6), this means that the two additional equations take the form

$$3y'_1 - 2s_1 - s_2 = h_1 f''(x_1)/2 \qquad 3y'_{n-1} + 2s_n + s_{n-1} = h_{n-1}f''(x_n)/2.$$

not-a-knot spline: Insist that also the third derivative be continuous across x_2 and across x_{n-1}. In effect, this makes q_1 and q_2 be the same polynomial (hence the break or 'knot' x_2 might as well not be there), and also makes q_{n-2} the same polynomial as q_{n-1}. This is the interpolating spline returned by `matlab`'s `spline(x,y)`. It has about the same error bound as complete spline interpolation and doesn't require any slopes.
`matlab`'s current version of `spline` can provide, optionally, the complete spline interpolant. Specifically, if, in the statement `spline(x,y)`, `y` has two more entries than `x`, then the first and last entry of `y` are taken to be the endslopes to be matched. E.g.,

```
n = 20; x = linspace(0,2*pi,n);
pc = spline(x,[cos(x(1)) sin(x) cos(x(n))]); pcc = spline(x,sin(x));
z = linspace(0,2*pi,100); plot(z, ppval(pc,z)-ppval(pcc,z));
```

provides both the complete spline interpolant to sin at twenty equally-spaced points as well as the not-a-knot spline interpolant, and plots their difference (which is very small, but is noticeable near the ends). Any other end conditions can be enforced with the aid of complete spline interpolation, as I'll make clear when we get into solving systems of linear equations.

On CS machines, `matlab` provides the command `splinetool` which makes it easy to experiment with these (and other choices for the) **endpoint conditions**.
`matlab`'s entire suite of commands for work with piecewise polynomials, `spline`, `ppval`, `mkpp`, `unmkpp`, also handle *vector-valued* piecewise polynomials, for working with planar and spatial curves. We'll return to that topic later in the course, if there is time.