Smooth Cubic Splines

This project is due at 11 PM on Tuesday, October 25. Turn it in the usual way.

Part 1.

The first part of this project is to produce a function, smoothCubic_DEC() that gives a C^2 (that is twice continuously differentiable) function, g, represented as an Hermite cubic. The input consists of

n	the number of points in the mesh, n must be at least 2.
p	a strictly increasing array of length n
v	an array of values of the function at the points of p
dl	a value that gives $g'(p[0])$
dr	a value that gives $g'(p[n-1])$

The output is an array d of length n that gives the derivatives of g at the points of p, i.e., $g'(p[i]) = d[i]$ for $i = 0$ to $n-1$. The DEC in the name of the function indicates "derivative end conditions".

I would suggest that you make an hermiteCubic data type so that you can deal easily with objects of this type. This would allow you to encapsulate the data and the output of this function in a single object, just include dl and dr as the first and last values in d before calling the function smoothCubic_DEC().

A tool that you will need in dealing with functions represented as Hermite cubics is an evaluation function. I would make this a member function of my hermiteCubic class. On each interval $I_i = [p[i-1], p[i]]$ the cubic is defined by the four pieces of data that give the value and derivative at each end of I_i for $i = 1$ to $n-1$. I would suggest that you define the function for all x by extending the function outside the interval using the end value and derivative. For the purposes of this project you can use a simple search to find where int he partition p the value of x falls. (Since p is increasing, there is likely a tool provided in your language to find this efficiently.)

In general a function represented as an Hermite cubic will be C^1 but not C^2. Your function smoothCubic_DEC() will need to formulate and solve a tridiagonal system to find the values of d that make g C^2.

Demonstrate the correctness of your smoothCubic_DEC() and evaluation process by testing your code on a nonuniform partition for 3 cases in which the data are taken from a function defined by a global polynomial of degree

less than 4.

Using the function $sin(x)$ on $[0, \pi/2]$ do a convergence study using uniform intervals to see how the maximum error goes to zero as the number of intervals increases.

Part 2.

For this part of the problem you will need n to be at least 4. You should produce a function not_a_knot() that takes the same inputs as smoothCubic_DEC() except for dl and dr. Instead of giving the end slopes of g, require that g be C^3 at $p[1]$ and $p[n-2]$. Solving for the n values of $d[i]$ requires the solution of a system of equations that is almost tridiagonal, but not quite. There are several ways to find the values, modifying your tridiagonal solver, doing the solution for the first and last values of d outside the solver, or using a low-rank update formula. In part 1 the natural $n \times n$ tridiagonal has a first and last row that has only one entry nonzero; these correspond to setting $d[0]$ and $d[n-1]$ to data. In part 2 the corresponding matrix has three nonzero values in every equation.

Repeat the tests that you did for part 1 using not_a_knot().

Some Formulas

The Hermite basis for cubics on $[0, 1]$ is used in setting up the equations you need to solve, so I will record some facts here that I believe to be correct and useful. You can check the correctness.

Define four functions V_0, V_1, S_0, and S_1 as follows:

$$\begin{aligned} V_0(x) &= 1 - 3x^2 + 2x^3 \\ V_1(x) &= 3x^2 - 2x^3 \\ S_0(x) &= x(x-1)^2 \\ S_1(x) &= x^2(x-1). \end{aligned}$$

With these any polynomial $q(x)$ of degree less than 4 can be written as

$$q(x) = q(0)V_0(x) + q(1)V_1(x) + q'(0)S_0(x) + q'(1)S_1(x).$$

These functions with shifting, dilation, and scaling, give the formulas you use in evaluating Hermite cubics.

Some facts that you will want to use are

$$\begin{array}{llll} V_0''(0) = & -6, & V_0''(1) = & 6 \\ V_1''(0) = & 6, & V_1''(1) = & -6 \\ S_0''(0) = & -4, & S_0''(1) = & 2 \\ S_1''(0) = & -2, & S_1''(1) = & 4. \end{array}$$

$$\begin{array}{l} V_0''' = 12, \\ V_1''' = -12, \\ S_0''' = 6, \\ S_1''' = 6. \end{array}$$

With I_i defined as above let $dp_i = p[i] - p[i-1]$ denote the length of this interval. For simplicity in the formulas that follow take I_0 and I_n to be empty. An Hermite cubic, g, over the partition p can be written as

$$g(x) = \sum_{i=0}^{n-1} (v[i]\mathcal{V}_i(x) + d[i]\mathcal{S}_i(x)),$$

where for $i = 0$ to $n-1$

$$\begin{array}{rcl} \mathcal{V}_i(x) & = & \begin{cases} V_1((x - p[i-1])/dp_i) & x \in I_i, \\ V_0((x - p[i])/dp_{i+1}) & x \in I_{i+1}, \\ 0 & otherwise, \end{cases} \\ \mathcal{S}_i(x) & = & \begin{cases} dp_i S_1((x - p[i-1])/dp_i) & x \in I_i, \\ dp_{i+1} S_0((x - p[i])/dp_{i+1}) & x \in I_{i+1}, \\ 0 & otherwise. \end{cases} \end{array}$$

Let $[[g_i'']]$ denote the jump in g'' at $p[i]$ for $0 < i < n-1$, i.e.,

$$[[g_i'']] = g''(p[i] + 0) - g''(p[i] - 0).$$

Here the "+0" and "−0" signify the limits from above and below, respectively. When we require that g be C^2 at each interior knot, we are just requiring that the jump in g'' be zero at each of the $n-2$ such knots. We can express this jump as follows:

$$
\begin{aligned}
[[g_i'']] &= v[i-1][0 - V_0''(1)/dp_i^2] \\
&\quad + v[i][V_0''(0)/dp_{i+1}^2 - V_1''(1)/dp_i^2] \\
&\quad + v[i+1][V_1''(0)/dp_{i+1}^2 - 0] \\
&\quad + d[i-1][0 - S_0''(1)/dp_i] \\
&\quad + d[i][S_0''(0)/dp_{i+1} - S_1''(1)/dp_i] \\
&\quad + d[i+1][S_1''(0)/dp_{i+1} - 0] \\
&= v[i-1][0 - 6/dp_i^2] \\
&\quad + v[i][-6/dp_{i+1}^2 + 6/dp_i^2] \\
&\quad + v[i+1][6/dp_{i+1}^2 - 0] \\
&\quad + d[i-1][0 - 2/dp_i] \\
&\quad + d[i][-4/dp_{i+1} - 4/dp_i] \\
&\quad + d[i+1][-2/dp_{i+1} - 0].
\end{aligned}
$$

Extending the notation for jumps in the natural way we get the formula that we need in the not-a-knot case:

$$
\begin{aligned}
[[g_i''']] &= v[i-1][0 - V_0'''/dp_i^3] \\
&\quad + v[i][V_0'''/dp_{i+1}^3 - V_1'''/dp_i^3] \\
&\quad + v[i+1][V_1'''/dp_{i+1}^3 - 0] \\
&\quad + d[i-1][0 - S_0'''/dp_i^2] \\
&\quad + d[i][S_0'''/dp_{i+1}^2 - S_1'''/dp_i^2] \\
&\quad + d[i+1][S_1'''/dp_{i+1}^2 - 0] \\
&= v[i-1][0 - 12/dp_i^3] \\
&\quad + v[i][12/dp_{i+1}^3 + 12/dp_i^3] \\
&\quad + v[i+1][-12/dp_{i+1}^3 - 0] \\
&\quad + d[i-1][0 - 6/dp_i^2] \\
&\quad + d[i][6/dp_{i+1}^2 - 6/dp_i^2] \\
&\quad + d[i+1][6/dp_{i+1}^2 - 0].
\end{aligned}
$$

In doing part 2 of this problem you need to write $[[g_1''']] = 0$ and $[[g_1'']] = 0$. If you do this in the straight forward way you write these as the first and second

equations in the system you want to solve. Similarly you want $[[g''_{n-2}]] = 0$ and $[[g'''_{n-2}]] = 0$. The set of equations is almost a tridiagonal system, but not quite. If you choose m carefully you can instead write the first equation as $[[g'''_1]] + m[[g''_1]] = 0$ and get an equation that does not involve $d[2]$. This and the analogous thing at $p[n-2]$ give an equivalent system that is tridiagonal.