Cubic Interpolation

Assumptions

We derive the cubic spline polynomials (basis functions), prefiltering system, and boundary condition adjustments.

We assume that the knot vector is uniformly spaced in increments of 1, so that the knots range from 1 to N, where N is the number of data points.

We will also search for symmetric basis functions.

Basis functions

We are looking for equations of the form (notice the p, q, q, p form that imposes the symmetry we said we wanted for):

$y_i(x) = c_{-1}\,p(x) + c_0\,q(x) + c_1\,q(1-x) + c_2\,p(1-x);\ x \in [0,\ 1)$
$y_i(x) = c_{-2}\,p(x+1) + c_{-1}\,q(x+1) + c_0\,q(-x) + c_1\,p(-x);\ x \in [-1,\ 0)$

The general version of these equations is:

$y_i(x) = c_{i-1}\,p(x-i) + c_i\,q(x-i) + c_{i+1}\,q(1-(x-i)) + c_{i+2}\,p(1-(x-i));\ x \in [i,\ i+1)$

Continuity in value, derivative, and second derivative at x=0 imply that

p(1) = 0
p(0) = q(1)
p'(1) = 0
p'(0) = q'(1)
q'(0) = 0
p''(1) = 0
p''(0) = q''(1)

These conditions are satisifed by the following cubic polynomials (TODO: show derivation somehow -- I just took it from Tim's notes):

```
p[x_] := a (1 - x)^3;
q[x_] := b + d x^2 + f x^3;

(* Also want derivatives so we can apply the conditions above *)
dp[x_] := Evaluate[D[p[foo], foo] /. {foo → x}];
ddp[x_] := Evaluate[D[p[foo], foo, foo] /. {foo → x}];
dq[x_] := Evaluate[D[q[foo], foo] /. {foo → x}];
ddq[x_] := Evaluate[D[q[foo], foo, foo] /. {foo → x}];
```

We also want the raw basis functions to be a partition of unity everywhere, which means

p(x) + q(x) + q(1-x) + p(1-x) = 1

```
params = Solve[{p[1] == 0 ,
          p[0] == q[1] ,
          dp[1] == 0 ,
          dp[0] == dq[1] ,
          dq[0] == 0 ,
         ddp[1] == 0 ,
         ddp[0] == ddq[1],
         p[x] + q[x] + q[1 - x] + p[1 - x] == 1}, {a, b, d, f}][[1]]
```

$$\left\{a \to \frac{1}{6}, b \to \frac{2}{3}, d \to -1, f \to \frac{1}{2}\right\}$$

We are now ready to construct our function y(x) for any x:

```
y0[x_] := Collect[cm * p[x] + c * q[x] + cp * q[1 - x] + cpp * p[1 - x], x] /. params
yN[x_, N_] := Collect[
   cm * p[(x - N)] + c * q[(x - N)] + cp * q[1 - (x - N)] + cpp * p[1 - (x - N)], x] /. params
(* To analyzy bounary conditions we also want first
 and second derivatives of y *)
dy0[x_] := Evaluate[D[y0[x], x]];
ddy0[x_] := Evaluate[D[y0[x], {x, 2}]];
dddyn[x_, n_] := Evaluate[D[yN[x, n], {x, 2}]];
coefs = {cm, c, cp, cpp};
```

Prefiltering system

We have derived the classic cubic spline. This will be non-interpolating, meaning the value of the spline at the knots will not equal the value of the input data. To overcome this, we will impose restrictions on cm, c, cp, cpp such that at integral values of x (the knots), the spline and data are equal.

This means, for example, that for x=0 and associated data point v0 want to solve the system

```
y0[0] == v0
```

$$\frac{2\,c}{3} + \frac{cm}{6} + \frac{cp}{6} == v0$$

Notice that this system is the same for all integers (I show only a few here):

```
Table[{i, yN[i, i]}, {i, 0, 5}] // TableForm
```

0	$\frac{2\,c}{3} + \frac{cm}{6} + \frac{cp}{6}$
1	$\frac{2\,c}{3} + \frac{cm}{6} + \frac{cp}{6}$
2	$\frac{2\,c}{3} + \frac{cm}{6} + \frac{cp}{6}$
3	$\frac{2\,c}{3} + \frac{cm}{6} + \frac{cp}{6}$
4	$\frac{2\,c}{3} + \frac{cm}{6} + \frac{cp}{6}$
5	$\frac{2\,c}{3} + \frac{cm}{6} + \frac{cp}{6}$

Because this system is the same for every integer value (knot), we need to solve a tri-diagonal system of the form:

$$\begin{pmatrix} X1 & X2 & X3 & X4 & 0 & 0 \\ 1/6 & 2/3 & 1/6 & 0 & \cdots & 0 \\ 0 & 1/6 & 2/3 & 1/6 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ \vdots & 0 & 0 & 1/2 & 2/3 & 1/6 \\ 0 & \cdots & X4 & X3 & X2 & X1 \end{pmatrix} = \begin{pmatrix} 0 \\ v0 \\ v1 \\ \vdots \\ vN \\ 0 \end{pmatrix}$$

Where X1, X2, X3 are additional constraints imposed on the first and last rows that enforce particular boundary conditions, which we turn to now.

Boundary Conditions

```
(* This function will print a formatted version of the condition in InputForm to
 be pasted into the docstring and build a table of values of x1, x2, x3, x4*)
sep[ex_] := If[ex > 0, " + ", " "];
fmt[ex_, nm_] :=
  If[ex ≠ 0, sep[ex] <> ToString[InputForm[ex]] <> " " <> nm , ""];
getX1234[ex_] := Module[{cx1, cx2, cx3, cx4, str},
  {cx1, cx2, cx3, cx4} = Coefficient[ex, #] & /@ coefs;
  (* Construct formatted string and print it *)
  str = If[cx1 ≠ 0, ToString[InputForm[cx1]] <> " cm", ""];
  str = Fold[#1 <> #2 &, str,
     MapThread[fmt, {{ cx2, cx3, cx4}, {"c", "cp" , "cpp"}}]] <> " = 0";
  Print[str];
  (* return a table of the coefficients on X *)
  TableForm[Transpose[{{X1, X2, X3, X4}, {cx1, cx2, cx3, cx4}}]]]
```

Flat

On Grid

The flat BC OnGrid sets y'=0 at x=0

```
getX1234[dy0[0]]
```

```
-1/2 cm + 1/2 cp = 0
```

X1	$-\frac{1}{2}$
X2	0
X3	$\frac{1}{2}$
X4	0

OnCell

Similarly, OnCell sets y'=0 at x=-1/2

```
getX1234[dy0[-1 / 2]]
```

```
-9/8 cm + 11/8 c -3/8 cp + 1/8 cpp = 0
```

$$\begin{aligned} X1 &\quad -\frac{9}{8} \\ X2 &\quad \frac{11}{8} \\ X3 &\quad -\frac{3}{8} \\ X4 &\quad \frac{1}{8} \end{aligned}$$

Natural (Linear) BC

OnGrid

The natural BC OnGrid sets y''0 at x=0

```
getX1234[ddy0[0]]
```

```
1 cm -2 c + 1 cp = 0
```

```
X1    1
X2   -2
X3    1
X4    0
```

OnCell

The natural BC OnCell sets y''0 at x=-1/2

```
getX1234[ddy0[-1 / 2]]
```

```
3/2 cm -7/2 c + 5/2 cp -1/2 cpp = 0
```

$$\begin{aligned} X1 &\quad \frac{3}{2} \\ X2 &\quad -\frac{7}{2} \\ X3 &\quad \frac{5}{2} \\ X4 &\quad -\frac{1}{2} \end{aligned}$$

Free BC

OnGrid or OnCell

The free boundary condition makes sure the interpoland has a continuous third derivative at the second-to-outermost cell boundary: $y_0'''(1) = y_1'''(1)$ and $y_{n-1}'''(n) = y_n'''(n)$

```
getX1234[Simplify[dddyn[0, 1] - dddyn[1, 1]]]
```

```
1 cm -3 c + 3 cp -1 cpp = 0
```

```
X1    1
X2   -3
X3    3
X4   -1
```

Note that this is the same as the quadratic case, so we use implementation from there.

Scratchpad