Cubic Splines and Matlab

October 7, 2006

1 Introduction

In this section, we introduce the concept of the cubic spline, and how they are implemented in Matlab. Of particular importance are the new Matlab data structures that we will see.

2 Cubic Splines Defined

Definition: Given n data points, $(x_1, y_1), \ldots, (x_n, y_n)$, a cubic spline is a piecewise-defined function of the form:

$$\begin{aligned}
S_1(x) &= y_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 && \text{for } x \in [x_1, x_2] \\
S_2(x) &= y_2 + b_2(x - x_2) + c_2(x - x_2)^2 + d_2(x - x_2)^3 && \text{for } x \in [x_2, x_3] \\
&\vdots \quad \vdots \\
S_{n-1}(x) &= y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3 && \text{for } x \in [x_{n-1}, x_n]
\end{aligned}$$

The $3n-3$ unknowns (b's, c's, d's) are chosen to satisfy the interpolation constraints and also some smoothness constraints:

- (Interpolation Conditions) We already have $S_i(x_i) = y_i$ for $i = 1$ to $n-1$. In addition, we want the overall function to be continuous. At the interior points,

  $$S_i(x_{i+1}) = S_{i+1}(x_{i+1}) = y_{i+1}$$

  and at the far right value of x, $S_{n-1}(x_n) = y_n$

  Altogether, there are $n-1$ constraints here.

- (Smoothness Condition 1) At the interior points,

  $$S_i{}'(x_{i+1}) = S_{i+1}{}'(x_{i+1})$$

  There are $n-2$ constraints here.

- (Smoothness Condition 2) At the interior points,

$$S_i{}''(x_{i+1}) = S_{i+1}{}''(x_{i+1})$$

  There are $n-2$ constraints here.

So far, we have $(n-1)+(n-2)+(n-2) = 3n-5$ constraints, but we have $3(n-1) = 3n-3$ coefficients. We need two more constraints to define the cubics uniquely.

2.1 Using the Two Extra Constraints

We can have different cubic splines depending on how we want to use our two extra constraints. Here are some common ones:

- (The Natural Spline) $S_1{}''(x_1) = 0 = S_{n-1}{}''(x_n)$
- (The Clamped Spline) The user defines $S_1{}'(x_1)$ and $S_{n-1}{}'(x_n)$
- "Not-A-Knot": The third derivatives match at x_2 and x_{n-1}- That is, $S'''_1(x_2) = S'''_2(x_2)$ and $S'''_{n-2}(x_{n-2}) = S'''_{n-1}(x_{n-2})$

Normally, at this step we would include some data. Notice that with cubic splines, you probably would not use them without at least 5 data points. With 4 data points, we could use a single cubic, for example.

2.2 Derivatives and Integrals

We just note here the derivative and antiderivative of each piece of the cubic spline,

$$S_i(x) = y_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3$$

$$S_i{}'(x) = b_i + 2c_i(x-x_i) + 3d_i(x-x_i)^2$$

and

$$S_i{}''(x) = 2c_i + 6d_i(x-x_i)$$

The antiderivative (note the shift):

$$\int S_i(x)\,dx = y_i(x-x_i) + \frac{1}{2}b_i(x-x_i)^2 + \frac{1}{3}c_i(x-x_i)^3 + \frac{1}{4}d_i(x-x_i)^4$$

In particular, if we integrate over the interval $[x_i, x_{i+1}]$ and the data is evenly spaced with width h, then:

$$\int_{x_i}^{x_{i+1}} S_i(x)\,dx = y_i h + \frac{1}{2}b_i h^2 + \frac{1}{3}c_i h^3 + \frac{1}{4}d_i h^4$$

3 Matlab Data Structures

In Matlab, we can collect different types of data together into a single "structure". For example, suppose that you have a vector `x` and a matrix `A` and several constants, a, b, c that you are going to be using over and over again. To wrap them up into a structure, you could write:

```
Examp.x=x;
Examp.A=A;
Examp.constants=[a,b,c];
```

To access, for example, the $(1, 3)$ element you've stored in the matrix A, you would type: `Examp.A(1,3)`

You can create arrays of structures and structures within structures. For example, suppose we use the data fitting tool in Matlab to create a spline:

```
x=0:10; y=sin(x);
plot(x,y,'r*');
```

Choose `Tools -> Basic Fitting` from the menus at the top of the figure. Click on the Spline Interpolant option, then click on the right arrow at the bottom right corner. We can save the spline structure by selecting `Save to Workspace` (You can uncheck the residuals boxes, just leave `Save fit as a MATLAB struct named` box. You can leave it named `fit`, then press OK. Close down the dialog boxes and plots and return to Matlab.

Type in the word for the structure, `fit` and you'll see a string, `type: 'spline'` and another structure, `coeff`. That is, `fit.coeff` is itself a structure- Type that in, and you'll see the contents:

```
>> fit.coeff
ans =
      form: 'pp'
    breaks: [0 1 2 3 4 5 6 7 8 9 10]
     coefs: [10x4 double]
    pieces: 10
     order: 4
       dim: 1
```

The actual spline coefficients are stored in the 10×4 array `coefs`. To see these, we would type: `fit.coeff.coefs`.

3.1 Some Useful Array Commands

Suppose I want to delete the second column of the matrix A. In Matlab, I would type: `A(:,2)=[];` Note the use of the colon, `:` to denote "all rows". Also, note that "delete" means to remove that column altogether.

Suppose I want to multiply the third column by 5, and put that column back into A: `A(:,3)=5*A(:,3);`

If I want to divide the elements of column 1 by the elements of column 2: `A(:,1)./A(:,2);`

3.2 Piecewise Polynomials

Matlab has built-in commands for dealing with piecewise-defined polynomials, like cubic splines. A piecewise-defined polynomial is defined in Matlab by a vector containing the breaks and a matrix defining the polynomial coefficients.

Caution: A vector of coefficients, like `[3,2,1]`, over an interval like `[2,3]` is interpreted by the piecewise polynomial routines as:

$$3(x-2)^2 + 2(x-2) + 1 \qquad \text{NOT} \qquad 3x^2 + 2x + 1$$

Example: Constructing a piecewise polynomial by hand:

Suppose we want to define the following:

$$P(x) = \begin{cases} x^2 - 3x + 1 & \text{if } x \in [1,2) \\ 3x - 5 & \text{if } x \in [2,3] \end{cases}$$

(NOTE: In the Matlab documentation, it is unclear which function is used to evaluate the interior breaks. This is the way it is done- The intervals are all assumed to be of the form $[a,b)$ except the last interval.)

First we need to rewrite these[1]:

$$x^2 - 3x + 1 = (x-1)^2 - (x-1) - 1$$

and

$$3x - 5 = 3(x-2) + 1$$

In Matlab, make the piecewise polynomial and plot the results:

```
breaks=[1,2,3];
coefs=[1,-1,-1;0,3,1];
pp=mkpp(breaks,coefs);
x=linspace(1,3);
y=ppval(pp,x);
plot(x,y);
```

[1]Is there an easy way to do this? Use a Taylor expansion based at the left endpoint.

4 Implementing Splines In Matlab

The relevent commands are:

`spline` `ppval` `mkpp`

(Also see the help page for any of these commands).

The spline command takes two forms, depending on what you want out:

```
pp = spline(x,y)
```

Builds the cubic spline using the data in x, y, and outputs the result as a piecewise polynomial, `pp`. See the previous section for how Matlab defines and works with piecewise defined functions.

The second command does not give the piecewise defined polynomial, rather it just evaluates the cubic spline at given values of `xin`, defined below:

```
yout = spline(x,y,xin)
```

where the cubic spline uses the data in x and y, evaluates the cubic spline at data in the vector `xin`, and gives the result in `yout`.

5 Matlab Exercises

In these exercises, go ahead and assume that the spline was constructed using Matlab's default algorithm.

1. What flavor of spline does Matlab use as its default? To verify your answer, look at the matrix of spline coefficients. You might use the sine function as a quick example:

   ```
   x=0:10;
   y=sin(x);
   pp=spline(x,y);
   ```

   In particular, pay attention to the order of the coefficients- Which column corresponds to constants?

2. How would we manipulate the coefficients to get the first, and second derivatives of a spline? Write functions `dspline`, `d2spline` whose input is the original spline structure (we called it `pp` in the previous exercise), and whose output will give you the derivatives as a piecewise polynomial structure. When you do this, you have a choice as to whether or not you want to change the degree of the polynomial.

   Below is a sample function that returns the third derivative, and leaves the degree as a cubic.

```
function pp=d3spline(pp)
% function p2 = d3spline(pp)
%  Returns the piecewise polynomial that represents the third
%  derivative of the cubic spline in the data structure pp.
%  Example:  x=0:10; y=sin(x); pp=spline(x,y);
%            p2=d3spline(pp);
%            xx=linspace(0,10);
%            yy=ppval(p2,xx);
%            plot(xx,yy);

A=zeros(size(pp.coefs));  %Initializes a matrix of zeros
A(:,4)=6*pp.coefs(:,1);
pp.coefs=A;
```

3. Do the same thing as before, but implement the antiderivative of the spline. Call the function `antispline`, and assume the constant of integration is zero. Be sure you also change `pp.order`.

   Extra: If you plot the antiderivative, you will get a very choppy curve (because each piece starts at zero). Is it possible to change this so that you get a smooth curve starting at zero? How would this code then be used to numerically estimate a definite integral?

4. We said before that the cubic spline is not well defined at the knots. That is, for the knot x_j, Matlab has to make a choice between using S_j or S_{j+1}. We know that it doesn't make a difference for the spline, its derivative or second derivative. Can you use the third derivative to determine which function is used at an interior point? Do it!

5. Use cubic splines and its antiderivative to estimate:

$$\int_{-3}^{3} \mathrm{e}^{-x^2}\, dx$$

   using 20 evenly spaced points. You might check your answer using Maple:

```
evalf(int(exp(-x^2),x=-3..3));
```

   Another way to see how your estimate is doing is to re-do the estimate using more points.