CS412: INTRODUCTION TO NUMERICAL ANALYSIS 10/19/10

Lecture 11: Interpolation by Cubic Splines

Instructor: Professor Amos Ron Scribes: Yunpeng Li, Mark Cowlishaw, Nathanael Fillmore

1 Review

Recall from last time the definition of a cubic spline.

DEFINITION 1.1 (Cubic Spline). *Given the partition $\vec{x} = (x_0, x_1, x_2, \ldots, x_N)$ of interval $[a, b]$, a cubic spline $s : \mathbb{R} \to \mathbb{R}$ is any function defined on $[a, b]$ such that*

1. *Each component of s is cubic, that is:*

$$s|_{[x_i, x_i+i]} \in \Pi_3 \qquad (\textit{for } i = 0, 1, \ldots, N-1)$$

2. *The second derivative of s is continuous over $[a, b]$*

$$s \in C^2[a, b]$$

(The notation $s \in C^k[a, b]$ means that s is k-times differentiable in the interval $[a, b]$ and the kth derivative is continuous.) We call the interior points of the partition $x_1, x_2, \ldots, x_{N-1}$ *knots*. Using this definition, we gave the following (incorrect) formulation for a cubic spline interpolation problem.

PROBLEM 1.1 (Interpolation Using Cubic Splines). *Given interval $[a, b]$, a set of points $\vec{x} = (x_0, x_1, \ldots, x_N)$ partitioning the interval, and function $f : [a, b] \to \mathbb{R}$, find a cubic spline s with knots $x_1, \ldots, x_{N-1}$ such that*

$$s(x_i) \quad = f(x_i) \quad (\textit{for } i = 0, 1, \ldots, N)$$

For simplicity, we assume that $\vec{x}$ is given in order ($a = x_0 < x_1 < \cdots < x_N = b$), and that each partition $[x_i, x_i + 1]$ has the same size, though these assumptions are not required.

As we showed in the last lecture, this problem is under-constrained. A spline over N intervals has $N + 3$ degrees of freedom, while the problem above gives only $N + 1$ constraints (one for each interpolation point). There are two fundamental ways we can change the problem to make the number of constraints match the number of degrees of freedom. We can either reduce the number of degrees of freedom for the spline s (not-a-knot), or increase the number of constraints (complete spline interpolation).

Not-a-Knot

Reduce the number of degrees of freedom by removing two knots. We remove the first knot x_1 and the last knot x_{N-1}, so the vector of knots becomes

$$(x_1, x_2, \ldots, x_{N-2}, x_{N-1}) \to (x_2, \ldots, x_{N-2})$$

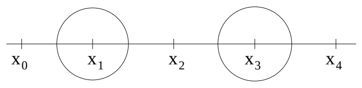

Figure 1: Not-a-Knot Interpolation over 5 Points

Note that the points (x_1, x_{N-1}) are still used in determining the polynomials near the endpoints, but instead of finding two cubic polynomials over the two intervals $[x_0, x_1]$, $[x_1, x_2]$, we find a single polynomial over the interval $[x_0, x_2]$. We treat x_{n-1} similarly.

EXAMPLE 1.1. *As an example, consider the not-a-knot formulation for a problem with* 5 *interpolation points as shown in Figure 1. We discard original knots x_1 and x_3, leaving x_2 as the only knot. We use polynomial interpolation to find a polynomial $p_0 \in \Pi_3$ for interval $[x_0, x_2]$ using the points x_0, x_1, x_2, x_3. Similarly, we find a polynomial interpolant $p_1 \in \Pi_3$ over $[x_2, x_4]$ using points x_1, x_2, x_3, x_4. Since both of these polynomials are cubic, they have* 4 *degrees of freedom, giving* 8 *degrees of freedom total. However, we add the additional constraints at the knot x_2:*

$$\begin{aligned} p_0(x_2) &= p_1(x_2) \\ p_0'(x_2) &= p_1'(x_2) \\ p_0''(x_2) &= p_1''(x_2) \end{aligned}$$

These constraints remove three degrees of freedom, so that we are left with $8 - 3 = 5$ *degrees of freedom, matching the original number of constraints.*

Complete Spline Interpolation

Increase the number of constraints, by requiring that the spline s matches the derivative of f at the endpoints.

$$\begin{aligned} s'(x_0) &= f'(x_0) \\ s'(x_N) &= f'(x_N) \end{aligned}$$

Unfortunately, the derivative of the function we are approximating is not always available. Even though complete spline interpolation can produce slightly better results than the not-a-knot formulation, it is not always possible to use it.

2 Using Splines

We will not discuss explicit algorithms for constructing cubic splines in this course. Instead, we will rely on the Matlab functions `spline` and `ppval`.

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

Input

`x` - The vector of nodes for the interpolation.

`y` - The vector of function evaluations at the nodes x (i.e. $y = f(x)$).

Output

`spline` outputs an abstract entity that represents the spline matching the given inputs. Examining the output value s directly is not useful, but you can use s as input to other Matlab functions, such as `ppval` to evaluate the spline.

You can use `spline` to perform spline interpolation using both forms of the problem.

Not-a-Knot

If `length(y) = length(x)`, then Matlab will use the not-a-knot formulation to create the spline s.

Complete Spline Interpolation

If `length(y) = length(x) + 2`, then Matlab will use complete spline interpolation to create the spline s. Matlab expects that the `y` vector is of the form

$$\texttt{y} = [f'(x_0), f(x_0), f(x_1), \ldots, f(x_{N-1}), f(x_N), f'(x_N)]$$

`w = ppval(s, z)`

`ppval` evaluates a given piecewise polynomial (the output of the `spline` function) over a given vector of points.

Input

`s` - A spline, the value returned by `spline(x, y)`.

`z` - A set of points over which to evaluate the spline.

Output

`ppval` outputs the vector of spline evaluations at the points in z (i.e. $w = s(z)$).

For example, to interpolate the function $\sin(t)$ using 10 interpolation points over the interval $[0, 3.1]$ then evaluate the result over a dense mesh of points z from $[0, 3.1]$, we could use the following Matlab commands:

```
x = linspace(0, 3.1, 10)
y = sin(x)
% z is a dense mesh of points on [0, 3.1]
z = linspace(0, 3.1)
s = spline(x, y)
w = ppval(s, z)
```

Note that there is a shorthand version of the `spline` function using three parameters that will create a spline *and* evaluate it over a given set of points. The above code can be rewritten as follows:

```
x = linspace(0, 3.1, 10)
y = sin(x)
% z is a dense mesh of points on [0, 3.1]
z = linspace(0, 3.1)
w = spline(x, y, z)
```

3 Error Analysis

Recall from previous lectures that we calculate interpolation error as the maximum difference (in absolute value) between the function we are trying to approximate and our approximation. When we calculate the error of piecewise functions over an interval $[x_0, x_N]$, we express the error in terms of the width h of each subinterval $[x_i, x_i + 1]$, with $h = (x_N - x_0)/N$, as shown in Figure 2. In this way, we can correlate the error to the cost of performing the interpolation.

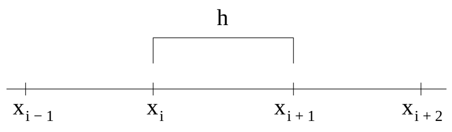

Figure 2: A Subinterval $[x_i, x_{i+1}]$ of the Interval $[x_0, x_N]$

Recall that the error of piecewise cubic functions was $O(h^4)$, so that if we double the cost of interpolation (by halving the size of each subinterval), the error will be reduced by a factor of 16. We would expect the error of cubic splines to be similar. The error for cubic spline interpolation using the not-a-knot form of the interpolation problem is:

$$\|s - f\|_{\infty,[x_0,x_N]} \underbrace{\sim}_{(1)} \underbrace{\frac{5}{384}}_{(2)} \cdot \underbrace{\left\|f^{(4)}\right\|_{\infty,[x_0,x_N]}}_{(3)} \cdot \underbrace{h^4}_{(4)} \tag{1}$$

This error expression has several important features:

(1) $\sim$

The expression is not an upper bound on the maximum error, but an approximation of the maximum error. The error may be somewhat larger than this near the endpoints.

(2) $\frac{5}{384}$

This is a small constant, so that, if the function is well-behaved, we should not have to make the subintervals too small to get a small error.

(3) $\left\|f^{(4)}\right\|_{\infty,[x_0,x_N]}$

The error depends on the fourth derivative of the function over the interval, so the function must be four times differentiable on $[x_0, x_N]$ for this formula to apply. A function with a large fourth derivative on the interval will require a fine partition of the interval. Note, however, that this term is a constant so that, as long as the function is four times differentiable on the interval, there is always some subinterval width h that will produce an acceptable error.

(4) h^4

The error shrinks proportionally to the fourth power of the size of the subintervals. If we double the amount of work, by shrinking the subinterval size by 1/2, the error will be reduced by a factor of 16.

We can make several comparisons and remarks about this error formula:

- First, note that this error formula applies only to spline interpolation using not-a-knot, although complete spline interpolation has a similar (though slightly smaller) error formula.
- Second, note that this error formula does not apply in the first and last subintervals in the partition, since knots are removed here. The error in these subintervals could be bigger by about 50%.
- Recall that for piecewise linear interpolation the error is bounded by $\frac{1}{8}\|f''\|_{\infty,[a,b]}h^2$. Thus, compared to cubic spline interpolation, piecewise linear interpolation requires less "fidelity" from our function (only the second derivative f'' must exist, not the fourth derivative $f^{(4)}$) but has a slower rate (quadratic instead of quartic).
- For piecewise cubic interpolation we saw that the error is bounded by $\frac{9}{384}\|f^{(4)}\|_{\infty,[a,b]}h^4$, which only differs from the error for cubic spline interpolation by a small constant factor. The primary advantage of splines over piecewise cubic polynomials is not that splines have better error, but that splines have derivatives everywhere on the interval.

3.1 Examples of Error Analysis for Spline Interpolation

Example 3.1. *If we use spline interpolation to approximate $f(t) = \cos(t)$ over the interval $[0, 3.1]$ with $h = 0.1$, what is our expected error?*

Since we know that $f^{(4)}(t) = \cos(t)$, $\left\|f^{(4)}\right\|_{\infty,[x_0,x_N]} \leq 1$. Plugging into error equation 1 yields:

$$\frac{5}{384} \cdot 1 \cdot (0.1)^4 = \frac{5}{3,840,000} = \frac{1}{768,000}$$

Notice that this relatively small error required only $31 + 1 = 32$ function evaluations.

We can also use equation 1 to decide how many function evaluations to perform to reach an "acceptable" level of error. This is a task of practical importance; for example we may want to determine in advance how many function evaluations are needed to achieve a certain level of interpolation error before we run an expensive or time-consuming experiment to access the function.

Example 3.2. *Consider a function $f : [0, 5] \to \mathbb{R}$ that is defined, but not easily accessible. We know the following facts, which uniquely define f on the interval $[0, 5]$:*

$$\begin{aligned} f''(t)\big|_{[0,5]} &= e^{-t^2} \\ f(0) &= 0 \\ f'(0) &= 0 \end{aligned}$$

How many function evaluations must we perform to approximate f on $[0,5]$ using cubic spline interpolation, if we would like the error to be approximately 10^{-6}?

To solve this problem, we use equation 1, set the error to 10^{-6}, and solve for h to determine an appropriate number of function evaluations:

$$\frac{5}{384} \cdot \left\| f^{(4)} \right\|_{\infty,[0,5]} \cdot h^4 = 10^{-6} \tag{2}$$

To determine $\left\| f^{(4)} \right\|_{\infty,[0,5]}$, we must evaluate $f^{(4)}$ at the endpoints and at any local extrema on the interval, so to find local extrema, we must differentiate f'' three times.

$$\begin{aligned}
f''(t) &= e^{-t^2} \\
f^{(3)}(t) &= -2te^{-t^2} \\
f^{(4)}(t) &= -2e^{-t^2} + 4t^2e^{-t^2} = (4t^2-2)e^{-t^2} \\
f^{(5)}(t) &= 8te^{-t^2} + -2t(4t^2-2)e^{-t^2} = 4t(3-2t^2)e^{-t^2}
\end{aligned}$$

Setting $f^{(5)}(t) = 0$, we find the local extrema 0 (which we would evaluate anyway, since it's an endpoint), and $\sqrt{3/2}$. Evaluating $f^{(4)}$, we find that

$$\begin{gathered}
f^{(4)}(0) = -2 \\
f^{(4)}(\sqrt{3/2}) \approxeq 0.4402 \\
f^{(4)}(5) = 0
\end{gathered}$$

So, $\left\| f^{(4)} \right\|_{\infty,[0,5]} = 2$. Plugging this value into equation 2 yields

$$\begin{aligned}
\frac{5}{384} \cdot 2 \cdot h^4 &= 10^{-6} \\
h^4 &= \frac{384 \cdot 10^{-6}}{2 \cdot 5} \\
h &= \sqrt[4]{384 \cdot 10^{-7}} \\
h &\approxeq 0.0787 \\
N &= \frac{5-0}{h} \approxeq 64
\end{aligned}$$

Since we require $N+1$ function evaluations to perform the spline interpolation, 65 function evaluations in total are needed to achieve an error of about 10^{-6}.

What if, having completed the above analysis, we change our mind, and we now want error of about $\frac{1}{16}10^{-6}$? We don't need to completely redo the analysis, but instead can do the following.

Let h and N be the subinterval size and the number of subintervals we found above, resulting in error about 10^{-6}. If we set $h' = \frac{1}{2}h$ then observe that

$$\begin{aligned}
\frac{5}{384}\|f^{(4)}\|_{\infty,[0,5]}(h')^4 &= \frac{5}{384}\|f^{(4)}\|_{\infty,[0,5]}(\frac{1}{2}h)^4 \\
&= \frac{1}{16}\frac{5}{384}\|f^{(4)}\|_{\infty,[0,5]}h^4 \\
&\approxeq \frac{1}{16}10^{-6} \quad \text{by above}
\end{aligned}$$

So $N' = 2N$ evaluations are needed to achieve the new error.

We end with two related notes:

- Note that when we are dealing with functions that are difficult to access, achieving an error of 10^{-6} is dependent on the accuracy of our function evaluations. We must have accuracy of at least 10^{-6} in the evaluations to have any hope of similar accuracy in the spline interpolant.
- Recall that to do complete spline interpolation we need derivatives at the endpoints. If these derivatives are not known, people sometimes approximate the derivative at the endpoints using a finite mesh, say using five points. If one does this, one needs to be careful, since the error introduced by approximating the derivatives can be larger than the error of splines on the whole interval. Usually, it is better to just use not-a-knot if the derivatives at the endpoints are not known.