Lecture 12: Cubic Hermite Spline Interpolation

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

1 Review of Interpolation using Cubic Splines

Recall from last time the problem of approximating a function over an interval using cubic splines.

PROBLEM 1.1. *Given an interval $[a,b]$, a function $f : [a,b] \to \mathbb{R}$, and a set of nodes $\vec{x} = (x_0, x_1, \ldots, x_N)$. Assume for simplicity that $a = x_0 < x_1 < \cdots < x_N = b$. Find a cubic spline $s : [a,b]$ such that*

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

We showed last time that, using the not-a-knot formulation, the error for the spline s could be approximated as:

$$\|s - f\|_{\infty,[a,b]} \sim \frac{5}{384} \left\| f^{(4)} \right\|_{\infty,[a,b]} \cdot h^4 \tag{1}$$

We did not provide an algorithm for finding a spline, but we can use builtin Matlab functions to create and evaluate splines. For example, to create and evaluate a spline s approximating function f using nodes x, we could use the following Matlab commands.

```
s    = spline(x, f(x))
sval = ppval(s, z)
```

where `s` is the internal representation of a spline in Matlab, `x` is a set of nodes, `z` is a dense mesh of points to evaluate the spline at and `sval` is the evaluation of the spline `s` at the points `z`.

Recall the example we used to end the last lecture.

EXAMPLE 1.1. *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}?

Last time we determined that it would take approximately 65 function evaluations to achieve an error of around 10^{-6}. It is important to note that this accuracy is entirely dependent on the

accuracy of our function evaluations - if the function evaluations do not have an accuracy of at least 10^{-6}, then the extra digits of accuracy in the spline interpolation are meaningless.

This may seem like an unimportant point but later on, we will find that some methods for evaluating functions defined similarly to f over an interval $[a, b]$ will require doing incremental evaluations over a dense mesh of points on $[a, b]$. Since the error of the spline is proportional to the 4th power of the size of the intervals, the error of the spline we could derive from these incremental evaluations over a large number of points could be dwarfed by the errors of the individual evaluations themselves. Finally, we will end our discussion of interpolation with interpolation by *cubic Hermite splines*.

2 A Different Polynomial Interpolation Problem

We will motivate interpolation by cubic Hermite splines by way of an example problem in polynomial interpolation.

PROBLEM 2.1. *Given an interval $[L, R]$ and a function $f : [L, R] \to \mathbb{R}$, find a polynomial $p \in \Pi_3$ approximating f such that:*

$$\begin{aligned} p(L) &= f(L) \\ p(R) &= f(R) \\ p'(L) &= f'(L) \\ p'(R) &= f'(R) \end{aligned}$$

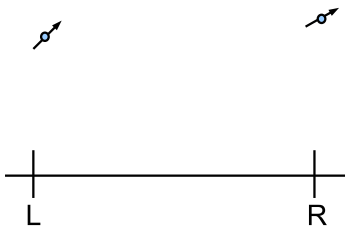

Figure 1: Situation described in Problem 2.1

The idea is that the interval $[L, R]$ will be a subinterval in a larger interval $[a, b]$, so that we can piece together several such solutions to form an approximation of f over a larger interval.

This problem is somewhat different than the interpolation problems we have seen before, which only involved constraints using the function values. For example, even though we want to interpolate just two points, we have four constraints, so we will need to take the interpolating polynomial p from the class Π_3 of cubic polynomials. The question is, can we adapt the tools we already have for polynomial interpolation to solve this problem?

2.1 Solution Using Divided Differences

The answer is yes, we can solve this problem using divided differences. Recall the formula for filling out the divided difference table

$$D[i,j] = f[x_i, x_{i+1}, \ldots, x_{i+j}] = \begin{cases} x_i & \text{if } j = 0 \\ \frac{D[i,j-1] - D[i+1,j-1]]}{x_i - x_{i+j}} & \text{Otherwise} \end{cases} \tag{2}$$

where $D[i,j]$ is the entry in the ith row and jth column of the divided difference table, with indexes starting from 0. If we look at the entries in column 1 of the table, we see that:

$$\begin{aligned} D[i,1] = f[x_i, x_{i+1}] &= \frac{D[i,0] - D[i+1,0]]}{x_i - x_{i+1}} &(3)\\ &= \frac{f(x_i) - f(x_{i+1})}{x_i - x_{i+1}} &(4) \end{aligned}$$

So $f[x_i, x_{i+1}]$ is the slope of the secant line from $f(x_i)$ to $f(x_{i+1})$.

To solve a problem like Problem 2.1, we set up the divided difference table by doubling the points and putting them in the order L, L, R, R:

L	$f[L]$	$f[L,L]$	$f[L,L,R]$	$f[L,L,R,R]$
L	$f[L]$	$f[L,R]$	$f[L,R,R]$	
R	$f[R]$	$f[R,R]$		
R	$f[R]$			

So, how can we use our derivative values, and how do we define $f[L,L]$ and $f[R,R]$, since, according to equation 3, each evaluates to $0/0$? Consider $f[L,L]$. We can try perturbing the second point near L, yielding

$$\begin{aligned} f[L,L] &= \lim_{\epsilon \to 0} f[L, L+\epsilon] \\ &= \lim_{\epsilon \to 0} \frac{f(L+\epsilon) - f(L)}{\epsilon} \\ &= f'(L) \end{aligned}$$

Thus, we can set $f[L,L] = f'(L)$ and $f[R,R] = f'(R)$, allowing us to fill in the divided difference table as usual. We obtain the Newton polynomials for the first row of the table by considering the points in order (L, L, R, R)

$$\begin{aligned} N_0(t) &= 1 \\ N_1(t) &= t - L \\ N_2(t) &= (t-L)(t-L) = (t-L)^2 \\ N_3(t) &= (t-L)^2(t-R) \end{aligned}$$

So that we obtain the polynomial p:

$$p(t) = f(L) \cdot 1 + f'(L) \cdot (t-l) + f[L,L,R] \cdot (t-l)^2 + f[L,L,R,R] \cdot (t-L)^2(t-R)$$

To illustrate this process, consider the following example.

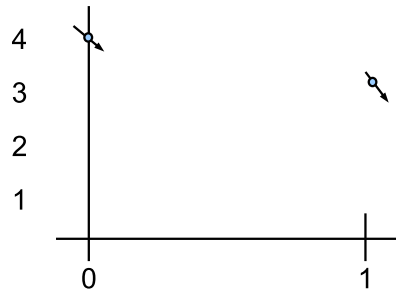

Figure 2: Situation described in Example 2.1

EXAMPLE 2.1. *Use divided differences to find a polynomial $p \in \Pi_3$ that approximates a function f over $[0,1]$ with*

$$\begin{aligned} f(0) &= 4 \\ f(1) &= 3 \\ f'(0) &= -1 \\ f'(1) &= -2 \end{aligned}$$

We set up our divided difference table by doubling the points 0 and 1, filling in the values for f in column 0 and the values for f' in the appropriate rows of column 1.

0	4	-1		
0	4			
1	3	-2		
1	3			

We can then fill out the rest of the table using equation 2.

0	4	-1	$\frac{-1--1}{0-1} = 0$	$\frac{0--1}{0-1} = -1$
0	4	$\frac{4-3}{0-1} = -1$	$\frac{-1--2}{0-1} = -1$	
1	3	-2		
1	3			

The Newton polynomials for the first row are given by:

$$\begin{aligned} N_0(t) &= 1 \\ N_1(t) &= t - 0 = t \\ N_2(t) &= t(t-0) = t^2 \\ N_3(t) &= t^2(t-1) \end{aligned}$$

And the polynomial p is:

$$p(t) \quad = \quad 4 \cdot 1 + -1 \cdot t + 0 \cdot t^2 + -2 \cdot t^2(t-1)$$

2.2 Error Analysis

Given that we are using a cubic polynomial to approximate f, the error for $t \in [L, R]$ is given by:

$$f(t) - p(t) = \frac{f^{(4)}(c)}{4!} \underbrace{(t-L)(t-L)(t-R)(t-R)}_{(*)} \qquad \text{for some } c \in [L, R] \tag{5}$$

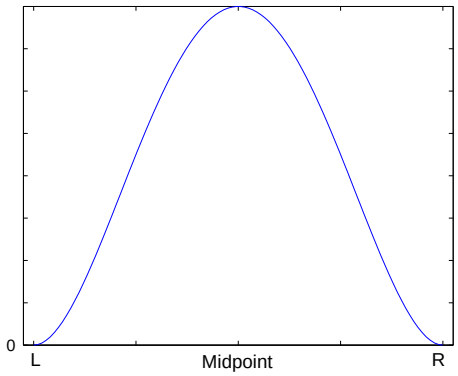

Figure 3: Graph of $(t-L)^2(t-R)^2$ over $[L, R]$

Since (*) is a quartic polynomial with a double-root at each endpoint, as shown in Figure 3, its maximum will be at the midpoint $(L+R)/2$, so that we can bound the error for $t \in [L, R]$ with the expression:

$$|f(t) - p(t)| \quad \leq \quad \frac{\left\|f^{(4)}\right\|_{\infty,[L,R]}}{4!} \frac{(R-L)^4}{16} \tag{6}$$

For example 2.1, this yields the bound

$$\begin{aligned} |E| \quad &\leq \quad \frac{\left\|f^{(4)}\right\|_{\infty,[L,R]}}{4!} \frac{(1-0)^4}{16} \\ &= \quad \frac{\left\|f^{(4)}\right\|_{\infty,[L,R]}}{384} \end{aligned}$$

This error formula depends only on the fidelity of our function - there's no parameter we can change to control the error, if L and R are fixed - so the procedure we have described in this section is not practical by itself. However, we will use it as a building block in the next section.

3 Cubic Hermite Spline Interpolation

The idea behind *interpolation by cubic Hermite splines* is to piece together the polynomials we constructed in the previous section. Given an interval $[a, b]$, a function $f : [a, b] \to \mathbb{R}$, with derivative $f' : [a, b] \to \mathbb{R}$ we would like to find a cubic Hermite spline s that approximates f over $[a, b]$. Our method will be to split the interval $[a, b]$ into N subintervals using nodes $\vec{x} = (x_0, x_1, \ldots, x_N)$ with $a = x_0 < x_1 < \cdots < x_N = b$. For each subinterval $[x_i, x_{i+1}]$, we find the cubic polynomial $p \in \Pi_3$ with

$$\begin{aligned} p(x_i) &= f(x_i) \\ p(x_{i+1}) &= f(x_{i+1}) \\ p'(x_i) &= f'(x_i) \\ p'(x_{i+1}) &= f'(x_{i+1}) \end{aligned}$$

If we simply piece these individual polynomials together, and we assume that each subinterval has width $h = (b - a)/N$, logically we can bound the error for $t \in [a, b]$ using:

$$|E(t)| \leq \frac{\left\| f^{(4)} \right\|_{\infty,[a,b]}}{384} \cdot h^4$$

For our error analysis, the nodes $\vec{x}$ are assumed to be equidistant, but this need not be the case. However, there is no reason to choose a different point distribution unless we have particular knowledge of f. For example, if we know that f is highly oscillatory on some part of $[a, b]$, we might choose more points from this part and fewer points elsewhere to improve the overall error.

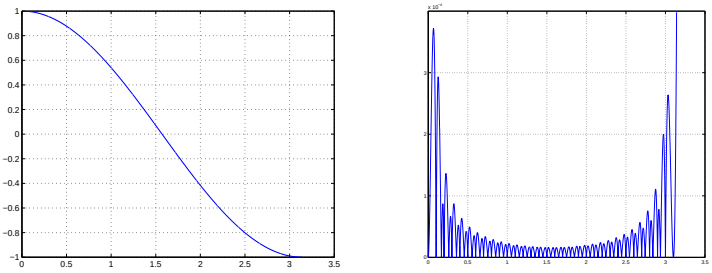

Figure 4: Interpolation of $\cos(t)$ over $[0, 3.1]$ using Cubic Hermite Splines and Associated Error

Note that the process of cubic Hermite spline interpolation requires f to be differentiable everywhere on $[a, b]$, and further, requires that we know how to differentiate f, so that we may not always be able to use this method.

In Matlab, the `pchip` function does cubic Hermite spline interpolation. Figure 4 shows an example using `pchip` to interpolate $\cos(t)$ over the interval $[0, 3.1]$.

Next time we will compare cubic spline interpolation with cubic Hermite spline interpolation.