Lecture 13: Cubic Hermite Spline Interpolation II

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

1 Cubic Hermite Spline Interpolation

Recall in the last lecture we presented a special polynomial interpolation problem.

PROBLEM 1.1. *Given an interval* $[L, R]$ *and a function* $f : [L, R] \to \mathbb{R}$*, find the cubic polynomial* $p \in \Pi_3$ *with*

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

Recall that we found a solution to problem 1.1 of the form:

$$p(t) = \tilde{a_1}(t-L)^2(t-R) + \tilde{a_2}(t-L)^2 + \tilde{a_3}(t-L) + \tilde{a_4} \tag{1}$$

Last time we saw that we can bound the error on p for $t \in [L, R]$ using:

$$|E(t)| \leq \frac{\left\|f^{(4)}\right\|_{\infty,[L,R]}}{384} \cdot (R-L)^4$$

To approximate a function f over an interval $[a, b]$, we can split the interval $[a, b]$ into N subintervals using partition points $\vec{x} = (x_0, x_1, \ldots, x_N)$, and solve problem 1.1 for every subinterval $[x_i, x_{i+1}]$. More formally, we can define the following cubic Hermite spline interpolation problem.

PROBLEM 1.2 (Cubic Hermite Spline Interpolation). *Given an interval* $[a, b]$*, a function* $f : [a, b] \to \mathbb{R}$*, with derivative* $f' : [a, b] \to \mathbb{R}$*, and a set of partition points* $\vec{x} = (x_0, x_1, \ldots, x_N)$ *with* $a = x_0 < x_1 < \cdots < x_N = b$*, find a set of polynomials* $p_0, p_1, \ldots, p_{N-1}$ *(a* cubic Hermite spline*) with*

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

for $i = 0, 1, \ldots, N-1$.

If we assume that each subinterval $[x_i, x_{i+1}]$ has equal width $h = (b-a)/N$, then we can bound the error for $t \in [x_i, x_{i+1}]$ as follows.

$$|E(t)| \leq \underbrace{\left\|f^{(4)}\right\|_{\infty,[x_i,x_{i+1}]}}_{(*)} \cdot \frac{1}{384} \cdot h^4 \tag{2}$$

Note that we bound the error of the cubic Hermite spline over each subinterval $[x_i, x_{i+1}]$ individually, yielding a very fine-grained error bound. Since the error depends on the fourth derivative of f only over a small interval (*), large fluctuations of the function in one subinterval will not affect the error in a different subinterval. We can also use a more coarse error bound, using the maximum of the $4th$ derivative of f over the entire interval, but this may be a slight overestimate of the error.

To illustrate the effect of the 4th power of h in equation 2, consider the following example.

EXAMPLE 1.1. *Given a function $f : [0, 4]$, if we partition $[0, 4]$ into N equal size partitions and perform cubic Hermite spline interpolation, we obtain an (experimental) error of $\approx 10^{-3}$. Approximately how many subintervals M (as a function of N) must we use to obtain an error of $\approx 10^{-5}$?*

Examining the error function for $t \in [a, b]$

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

we see that (*) is simply a constant , so that, in the first case, we have:

$$|E_1(t)| \approx 10^{-3} = c_f \cdot h^4 = c_f \cdot \left(\frac{4-0}{N}\right)^4$$

where c_f is a constant that depends on f. Similarly, in the second case:

$$|E_2(t)| \approx 10^{-5} = c_f \cdot h^4 = c_f \cdot \left(\frac{4-0}{M}\right)^4$$

To determine M, we take the quotient of the two errors and solve for M:

$$\begin{aligned}
\frac{|E_1(t)|}{|E_2(t)|} &= \frac{10^{-3}}{10^{-5}} = 100 \\
100 &= \frac{c_f \cdot \left(\frac{4}{N}\right)^4}{c_f \cdot \left(\frac{4}{M}\right)^4} \\
100 &= \left(\frac{M}{N}\right)^4 \\
\frac{M}{N} &= \sqrt[4]{100} \approxeq 3.1
\end{aligned}$$

Thus, it will take approximately $M = 3.1 \cdot N$ intervals to reduce the error by a factor of 100.

2 Comparison of Interpolation Methods

Now that we have seen two methods for interpolation using two different kinds of splines (cubic splines and cubic Hermite splines), how can we choose which method to use for a particular problem? Table 1 compares some properties of each method.

	Cubic Hermite Splines	Cubic Splines
Error Bound:	$\frac{1}{384}\|\|f^{(4)}\|\|_{\infty[a,b]}h^4$	$\frac{5}{384}\|\|f^{(4)}\|\|_{\infty[a,b]}h^4$
Smoothness:	$\mathcal{C}^1$	$\mathcal{C}^2$
Locality:	very local	local

Table 1: Comparison of Interpolation by Cubic Hermite Splines and Cubic Splines

As we can see from the table, both methods have an error bound that is $O(h^4)$. But it would seem that cubic Hermite spline interpolation uses a smaller constant:

$$\frac{1}{384}\left\|f^{(4)}\right\|_{\infty,[a,b]} < \frac{5}{384}\left\|f^{(4)}\right\|_{\infty,[a,b]}$$

so that it would provide more accuracy for the same interval size h. However, to truly gauge the *speed* of each algorithm (the amount of cost required to achieve a desired accuracy), we need to look more closely at the cost of each algorithm. As before, we'll use the number of function evaluations to gauge the cost.

It is easy to see that cubic spline interpolation over N intervals requires $N+1$ function evaluations (one for each partition point). On the other hand, cubic Hermite spline interpolation over N subintervals requires $N+1$ function evaluations and an additional $N+1$ evaluations of the derivative. Assuming that evaluating the derivative is at least as costly as evaluating the function, this means that finding cubic Hermite splines over N subintervals is *twice* as expensive as finding cubic splines. So, for the same cost, we can use twice as many subintervals (subintervals of half the size) to find cubic splines as we use for cubic Hermite splines. So, the error for cubic splines normalized for cost is:

$$\begin{aligned} |E(t)| &\sim \frac{5}{384}||f^{(4)}||_{\infty[a,b]}\left(\frac{h}{2}\right)^4 \\ &= \frac{5}{16\cdot 384}||f^{(4)}||_{\infty[a,b]}h^4 \end{aligned}$$

and thus, cubic splines can produce slightly more accuracy for the same cost as cubic Hermite splines. Note that this analysis applies to situations in which the cost of evaluating the function dominates the cost of running the algorithm given the function values. This covers most interesting cases, however, if function and derivative values are provided in advance, the algorithm for finding cubic Hermite splines over N subintervals is actually faster than the algorithm for finding cubic splines over the same subintervals.

From the table, we can see that both kinds of splines are differentiable, but cubic splines are doubly differentiable, so they may provide a more pleasing visual representation of the function.

The last row in the table compares the locality of the methods. Locality measures the degree to which the function approximation is affected by small, local changes in the interpolation data.

A method is *local* if small, local changes in the interpolation data have limited affects outside the area near the change. A method is *global* if small, local changes in interpolation data may affect the entire approximation. An example of locality is shown in Figure 1.

We can tell from the algorithm for cubic Hermite spline interpolation that the method is extremely local. If we change a single interpolation point x_i, the approximation can only be affected in the two intervals $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$ that share that partition point. It turns out that cubic spline interpolation is also local, but not quite to the degree of cubic Hermite spline interpolation. In cubic spline interpolation, local changes to the interpolation data may have small effects outside the area of change, but these effects diminish rapidly as the distance from the area of change increases. In contrast, polynomial interpolation is a *global* method - local changes in interpolation data can affect the entire approximation.

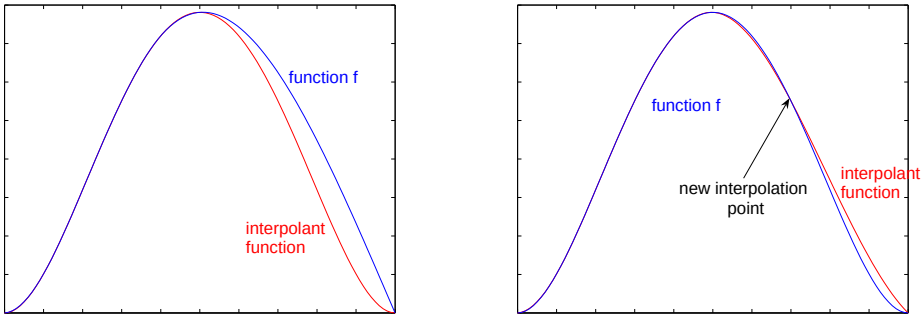

Figure 1: An Example of a *Local* Interpolation Method

3 Cubic Hermite Spline Interpolation in MATLAB

There are two methods of doing interpolation using cubic Hermite splines in Matlab. The first is the function `pchip`.

```
pp = pchip(x, f(x))
```

`pchip` takes a vector of nodes x and the corresponding function values $f(x)$, and produces a cubic Hermite spline in Matlab's internal format. One can then use `ppval` to evaluate the cubic Hermite spline over a dense mesh of points.

The careful reader will notice that `pchip` takes function values as input, but no derivative values. This is because `pchip` uses the function values $f(x)$ to estimate the derivative values. Recall that we can approximate the derivative at a point x_1 using the slope of a secant line between adjacent points x_0 and x_2:

$$f'(x_1) \approxeq \frac{f(x_2) - f(x_0)}{x_2 - x_0}$$

However, this would produce an error $\sim O(h^2)$, which is much worse than the interpolation error. To do a good derivative approximation, the function has to use an approximation using 4

or more points - the exact algorithm is beyond the scope of this class. It is important to realize, however, that approximating the derivative introduces an additional error into the interpolation.

Luckily, using Matlab we can write our own functions to do interpolation using real cubic Hermite splines.

3.1 Creating a Function for Cubic Hermite Spline Interpolation

Recall that, to do spline interpolation, we use the command `spline`, for example:

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

We can evaluate the spline s using the function `ppval`.

```
fv=ppval(s,z)
```

We can access the individual polynomial pieces of a spline using the `unmkpp` command.

```
[X, C, ...] = unmkpp(s)
```

`unmkpp` returns an array of 5 elements, but we only need the first two.

(1) `X`

The vector of input nodes that were used in creating the spline.

(2) `C`

A matrix containing a representation of the polynomials for each subinterval. Each row i of the matrix contains the coefficients of the polynomial for the ith interval. Remembering that Matlab indexes vectors and matrices starting with 1, the first row $C(1)$ represents the interval $[X(1), X(2)]$ (the first two elements of the vector X). In our previous notation, this is the interval $[x_0, x_1]$. Thus, C has the form:

$$C = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \vdots & \vdots & \vdots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & a_{i4} \\ \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & a_{N4} \end{bmatrix}$$

The polynomial for row i, p_i for the interval $[x_{i-1}, x_i]$ is represented as:

$$p_i(t) = a_{i1}(t - x_{i-1})^3 + a_{i2}(t - x_{i-1})^2 + a_{i3}(t - x_{i-1}) + a_{i4}$$

If we replace x_{i-1} with L, this becomes:

$$p(t) = a_1(t - L)^3 + a_2(t - L)^2 + a_3(t - L) + a_4 \quad (3)$$

If we can reconstruct this matrix C, then we can create a cubic Hermite spline s using the Matlab function `mkpp`:

```
s = mkpp(X, C)
```

Recall that the result of cubic Hermite spline interpolation was a polynomial for each interval $[L, R]$ of the form:

$$p(t) = \tilde{a_1}(t-L)^2(t-R) + \tilde{a_2}(t-L)^2 + \tilde{a_3}(t-L) + \tilde{a_4} \tag{4}$$

Substituting $[t - L - (R - L)]$ for $(t - R)$ in equation 4 yields:

$$\begin{aligned} p(t) &= \tilde{a_1}(t-L)^2[t-L-(R-L)] + \tilde{a_2}(t-L)^2 + \tilde{a_3}(t-L) + \tilde{a_4} \\ &= \tilde{a_1}(t-L)^3 - \tilde{a_1}(t-L)^2(R-L) + \tilde{a_2}(t-L)^2 + \tilde{a_3}(t-L) + \tilde{a_4} \\ &= \tilde{a_1}(t-L)^3 + [\tilde{a_2} - \tilde{a_1}(R-L)](t-L)^2 + \tilde{a_3}(t-L) + \tilde{a_4} \end{aligned}$$

Thus we can represent each polynomial p we obtain from cubic Hermite spline interpolation as a row of matrix C using:

$$\begin{aligned} a_4 &= \tilde{a_4} \\ a_3 &= \tilde{a_3} \\ a_2 &= \tilde{a_2} - \tilde{a_1}(R-L) \\ a_1 &= \tilde{a_1} \end{aligned}$$

Using this transformation, we can create the matrix C, and use it to create true cubic Hermite splines in Matlab.