Lecture 10: Introduction to Splines

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

1 Review of Chebyshev Points

Last time we talked briefly about using Chebyshev points for polynomial interpolation. The idea is that our choice of interpolation points can have a large impact on the error of our interpolant. Recall the expression for error in polynomial interpolation (Given f the function we are approximating, interval $[a, b]$, interpolation points $\vec{x} = (x_0, x_1, \ldots, x_n)$, and interpolant p_n):

$$E(t) = f(t) - p_n(t) = \frac{f^{(n+1)}(c)}{(n+1)!} \cdot \underbrace{\left(\prod_{i=0}^{n} (t - x_i) \right)}_{\text{(a)}} \qquad \text{(for some } c \in [a, b]) \tag{1}$$

If we know a great deal about the function f, then we may be able to choose points so as to reduce the error. If we don't have such information about the function, however, the best we can do is to reduce the product (a). The Chebyshev points effectively minimize the maximum value of the product (a).

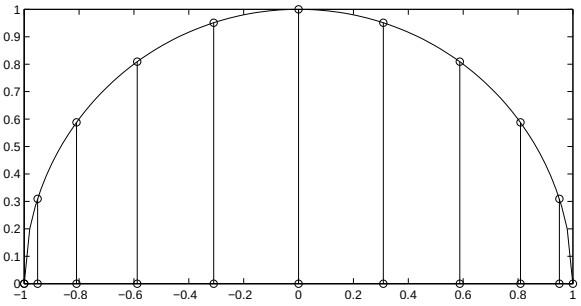


Figure 1: Choosing Chebyshev Points

Recall the process for selecting Chebyshev points over an interval $[a, b]$, as shown in Figure 1:

1. Draw the semicircle on $[a, b]$ centered at the midpoint $((a + b)/2)$.
2. To select $N + 1$ points, split the semicircle into N arcs of equal length.
3. Project the arcs onto the x-axis, giving the following formula for each Chebyshev point x_j

$$x_j = \frac{a+b}{2} + \frac{b-a}{2} \cdot \cos\left(\frac{j \cdot \pi}{N}\right) \qquad \text{(for } j = 0, 1, \ldots, N)$$

Note that this is not exactly the process for choosing Chebyshev points, but it is a close approximation.

2 Interpolation Using Piecewise Polynomials

Recall that last time we discussed interpolation over an interval $[a, b]$ by splitting the interval into N equal partitions and using a fixed degree polynomial to interpolate each partition separately. We discussed two specific ideas for approximating the function over each partition, using linear polynomials and using cubic polynomials.

2.1 Piecewise Linear Functions

Given an interval $[a, b]$, and a function f, we split the interval into N equal subintervals of size $h = (b - a)/N$ with endpoints: $(x_0, x_1, x_2, \ldots, x_N)$. We can then interpolate each subinterval $[x_i, x_{i+1}]$ using a line. Last time we proved an upper bound on the error of the resulting interpolant for any $t \in [a, b]$:

$$E(t) = |f(t) - p(t)| \leq \frac{\|f''\|_{\infty,[a,b]}}{8} \cdot h^2 \tag{2}$$

The error is reduced in proportion to the square of the size of the subintervals - if the size is halved, the error will be reduced by a factor of four.

The cost of doing interpolation this way is the cost of performing $N + 1$ function evaluations and N interpolations. Note that the error is inversely proportional to the square of the number of intervals ($E(t) = O(1/N^2)$), this is generally not considered good enough to make this a viable interpolation method, the methods in general use have $E(t) = O(h^4) = O(1/N^4)$.

2.2 Piecewise Cubic Functions

As we saw briefly last time, a simple idea for improving the error bound is to use higher degree polynomials to interpolate each subinterval. Given a function f and an interval $[a, b]$, we partition the interval into N subintervals with endpoints $\vec{x} = (x_0, x_1, \ldots, x_N)$. If we take four consecutive points from $\vec{x}$, $(x_{i-1}, x_i, x_{i+1}, x_{i+2})$, we can use these points to find a cubic polynomial $p_i \in \Pi_3$ that interpolates f. We will use p_i only on the middle interval $[x_i, x_{i+1}]$, as shown in Figure 2.

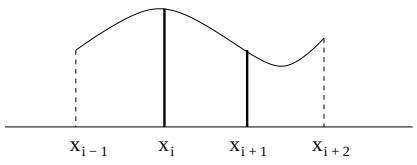


Figure 2: The Cubic Interpolant for Interval $[x_i, x_{i+1}]$

By simple substitution into equation 1, the error over the subinterval $[x_i, x_{i+1}]$ is:

$$|f(t) - p_i(t)| = |\frac{f^{(4)}(c)}{4!} \cdot (t - x_{i-1})(t - x_i)(t - x_{i+1})(t - x_{i+2})| \qquad \text{(for some } c \in [x_i, x_{i+1}]) \tag{3}$$

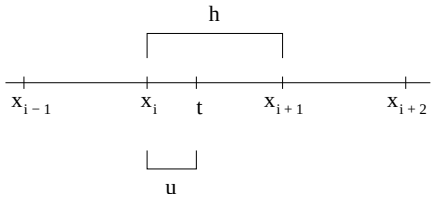


Figure 3: A Point t in the Interval $[x_i, x_{i+1}]$

Here p_i refers to the polynomial interpolant for the ith subinterval, and not a polynomial of degree $\leq i$. Recall that each interval has length h, and let $u = t - x_i$. As shown in Figure 3 we can transform equation 3 into

$$f(t) - p_i(t) = \frac{f^{(4)}(c)}{24} \cdot \underbrace{(h+u)(u)(h-u)(2h-u)}_{(*)} \tag{4}$$

Simple calculus shows that the maximum of the expression (*) over $u \in [0, h]$ occurs at the midpoint, $u = h/2$. Thus, we can bound the error on $[x_i, x_{i+1}]$:

$$\begin{aligned}
f(t) - p_i(t) &= \frac{f^{(4)}(c)}{24} \cdot (h+u)(u)(h-u)(2h-u) \\
&\leq \frac{\left\|f^{(4)}\right\|_{\infty,[x_i,x_{i+1}]}}{24} \cdot (h + h/2)(h/2)(h - h/2)(2h - h/2) \\
&= \frac{\left\|f^{(4)}\right\|_{\infty,[x_i,x_{i+1}]}}{24} \cdot \left(\frac{3h}{2}\right)^2 \left(\frac{h}{2}\right)^2 \\
&= \frac{\left\|f^{(4)}\right\|_{\infty,[x_i,x_{i+1}]}}{24} \cdot \frac{9h^4}{16} \\
&= \frac{9 \cdot \left\|f^{(4)}\right\|_{\infty,[x_i,x_{i+1}]}}{384} \cdot h^4
\end{aligned}$$

Since the error over the entire interval $[a, b]$ is bounded by the maximum of any of the errors over a particular subinterval, this yields the following error bound for $t \in [x_1, x_{N-1}]$

$$E(t) = |f(t) - s(t)| \leq \underbrace{\frac{9}{384}}_{(a)} \cdot \underbrace{\left\|f^{(4)}\right\|_{\infty,[a,b]}}_{(b)} \cdot \underbrace{h^4}_{(c)} \tag{5}$$

Where $s(t)$ is the piecewise cubic function we have created. We do not include the first and last subintervals, since we have not defined the interpolant there similarly. (On the first subinterval, we use x_0, x_1, x_2, x_3 to find a curve between x_0 and x_1, while on the last subinterval, we use $x_{N-3}, x_{N-2}, x_{N-1}, x_N$ to find a curve between x_{N-1} and x_N.) Error equation 5 has three important components

(a) $\frac{9}{384}$

This is a small constant, so that, unless the fourth derivative is very large over the interval, we will have a small error without having to shrink h too much.

(b) $\left\|f^{(4)}\right\|_{\infty,[a,b]}$

The error bound depends on the fourth derivative, so f must have the fourth derivative defined everywhere on the interval $[a, b]$ for this analysis to apply. Even if the fourth derivative is large over the interval, this value is only a constant, so that it will be dominated by a small enough h.

(c) h^4

The error is $O(h^4)$, meaning that the error decreases very rapidly as h gets smaller. For example, if we halve h, the error will be divided by sixteen. It is possible to invent methods that decrease the error by higher powers of h (for example, by using quartic or quintic polynomials), but the improvement in convergence is marginal, and generally considered not worth the extra cost of a single iteration.

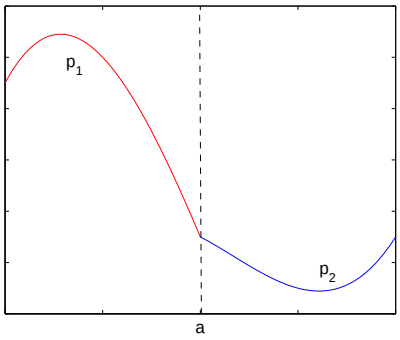


Figure 4: A Piecewise Cubic Function s

Unfortunately, while this method will efficiently produce functions s with a very small error, there are some problems with the results. It is easy to see that the function s produced by this method will be continuous, since the polynomials p_j over $[x_j, x_{j+1}]$ and p_{j+1} over $[x_{j+1}, x_{j+2}]$ both match f exactly at x_{j+1}. However, as shown in Figure 4, it is quite likely that the derivative at the endpoints of each subinterval will not exist, since $p'_j(x_{j+1})$ may not match $p'_{j+1}(x_{j+1})$. This creates problems with the graph of s, which will not appear smooth. Additionally, we cannot use s to estimate the derivative of f, which may cause problems depending on our intended use for s. As we shall see, splines seek to solve these problems.

Note that the error bound given in Equation 5 is a uniform bound of the error over the whole interval $[a, b]$. Of course, the error in any subinterval may be even less than this. For example consider the function $f(t) = e^t$ on $[0, 10]$. Here $f^{(4)} = f = e^t$, and e^t is increasing everywhere, so the maximum of f over any interval is always attained at the right endpoint of the interval. In particular,

$$\|f^{(4)}\|_{\infty,[x_i,x_{i+1}]} = e^{x_{i+1}} \quad i = 0, \ldots, N-1$$

so on the interval $[x_i, x_{i+1}]$ the error is bounded by

$$\frac{9}{384} e^{x_{i+1}} h^4.$$

3 Cubic Splines

The idea behind cubic splines is to piece together cubic polynomials so as to make the result differentiable as many times as possible.

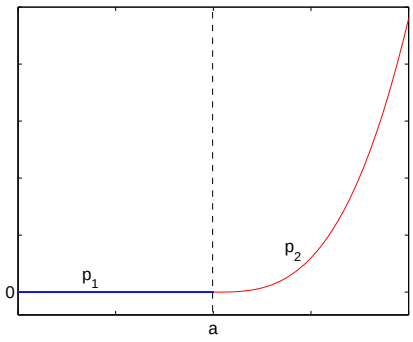


Figure 5: Cubic Functions at a Point a

This begs the question - for how many derivatives can two different cubic functions agree at a point a? Consider two functions p_1 and p_2 at a point a as shown in Figure 5. Assume, for simplicity, that p_1 is identically 0.

Clearly, we can create a different cubic function p_2 with $p_2(a) = 0$, for example, $p_2(t) = c_1 \cdot (t-a)^3 + c_2 \cdot (t-a)^2 + c_3 \cdot (t-a)$ works. Similarly, we can create a p_2 with $p_2'(a) = 0$, for example $p_2(t) = c_1 \cdot (t-a)^3 + c_2 \cdot (t-a)^2$. We can also create a p_2 with $p''(a) = 0$, but it must be of the form $p_2(t) = c \cdot (t-a)^3$. However, if we try to match the third derivative, the only cubic function with a third derivative 0 is $p_2(t) = 0$, which is the same function as p_1.

As this simple example illustrates, we can only hope to match the first two derivatives of two different cubic functions at any particular point, and thus we can only hope to make our function s twice differentiable over $[a, b]$. This requires us to restrict the piecewise cubic functions. Once we find a cubic polynomial p_1, and we wish to find a cubic polynomial p_2 with

$$\begin{aligned} p_1(a) &= p_2(a) \\ p_1'(a) &= p_2'(a) \\ p_1''(a) &= p_2''(a) \end{aligned}$$

then p_2 must have the form:

$$p_2(t) = p_1(t) + c \cdot (t-a)^3$$

This representation has only one degree of freedom (the value of c). In general, each time we restrict a function to match one of the derivatives of another function at a particular point, we remove one degree of freedom. This discussion of matching derivatives leads to the following definition of a cubic spline.

Definition 3.1 (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 $[a, b]$*

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

Note that the points $(x_0, x_1, \ldots x_N)$ that partition the interval $[a, b]$ are known as knots.

It is important to note that, while discontinuities in the first and second derivative of a function are visually detectable, this is not true of the third and higher derivatives. This is why cubic splines are particularly appropriate for visual approximations of a function.

4 Interpolation Using Cubic Splines

How do we use cubic splines to approximate a function? The formulation of the problem we have used before would go something like this.

PROBLEM 4.1 (Interpolation Using Cubic Splines). *Given interval $[a, b]$, $\vec{x} = (x_0, x_1, \ldots, x_N)$ partitioning the interval, and function $f : [a, b] \to \mathbb{R}$, find a cubic spline s with knots $\vec{x}$ such that*

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

Unfortunately, this statement of the problem is incorrect. We would like, when we state a problem, to have the number of constraints in the problem match the number of degrees of freedom in the representation for the solution. For example, we know that a polynomial of degree k has $k+1$ degrees of freedom. If we consider the monomial form, there are $k + 1$ coefficients that uniquely determine the polynomial (in Π_k). Thus, when we stated the polynomial interpolation problem, we constrained the degree k polynomial we were looking for with $k + 1$ points.

If the number of degrees of freedom match the number of independent constraints, we will have a unique solution. If we have more constraints than degrees of freedom, then the problem will have no solution, unless some of the constraints are not independent. If we have more degrees of freedom than constraints, there will be many solutions.

The problem statement above has $N + 1$ constraints, one for each knot. How many degrees of freedom does a cubic spline over $N + 1$ knots have?

Consider that, in choosing an initial polynomial (for the first partition $[x_0, x_1]$), we have four degrees of freedom. As we've seen, we only have one degree of freedom for the cubic polynomial in the next partition, and the same for each partition thereafter. Since we have N total partitions, we have $N + 3$ degrees of freedom. Thus, the problem as stated is under-constrained.

We can address this in two fundamental ways, by increasing the number of constraints or decreasing the number of degrees of freedom. These translate into the following two general methods for forming a spline interpolation problem.

Not-a-Knot

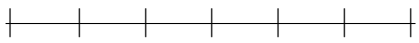
Reduce the number of degrees of freedom by removing two knots from the list of knots. We remove the second knot x_1 and the next to last knot x_{N-1}

$$(x_0, x_1, x_2, \ldots, x_{N-2}, x_{N-1}, x_N) \to (x_0, x_2, \ldots, x_{N-2}, x_N)$$

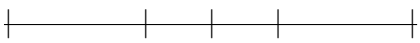
Note that the points (x_1, x_{N-1}) are still used in determining the polynomials near the endpoints, but instead of finding two cubic polynomials over the two intervals $[x_0, x_1]$, $[x_1, x_2]$, we find a single polynomial over the interval $[x_0, x_2]$. We treat x_{n-1} similarly.

For example if we have the 7 interpolation points $x_0, \ldots, x_6$ depicted below:

Instead of fitting polynomial pieces in every interval, i.e.,



we will combine the first two and the last two pieces:



Note that we still interpolated x_1 and x_{N-1} - we just *added* constraints.

As another example, suppose we have just four interpolation points:

This gives the intervals

When we combine the first and second, and next-to-last and last intervals, we end up with a simple polynomial interpolation problem over the whole interval:

Finally, if we have 5 interpolation points:

and we remove the first partition point and the next-to-last partition point, we get

- two pieces. So we interpolate this with two cubic polynomials.

Complete Spline Interpolation

Increase the number of constraints, by requiring that the cubic polynomial over $[x_0, x_1]$ match the derivative of f at x_0 and placing a similar constraint at x_N.

Of course, this can be difficult if you do not know the derivative of the function. (You can estimate the derivative numerically in this case, but this has some subtleties and is outside our scope.)

We will discuss these methods in more detail at the next lecture.