CS412: INTRODUCTION TO NUMERICAL ANALYSIS 10/17/10

Lecture 9: Polynomial Interpolation: Error Analysis and Introduction to Splines

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

1 Error Analysis

Recall the error formula for polynomial interpolation. Given some function f, a set of interpolation points $\vec{x} = (x_0, x_1, \ldots, x_n)$ over the interval $[a, b]$ ($x_i \in [a, b]$ for $i = 0, 1, \ldots, n$), and the polynomial interpolant $p_n \in \Pi_n$, we define the error at any point $t \in [a, b]$:

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

Note that the product (*) ensures that the error is zero at each of the interpolation points and would be the next Newton polynomial (N_{n+1}), if another interpolation point were added.

Why do we use the $n+1$ order derivative of f to describe the error of a polynomial of degree at most n? The intuition is that the $n+1$ order derivative of a degree n polynomial is identically 0, so that the difference between the actual function and the polynomial interpolant can be encapsulated by this derivative.

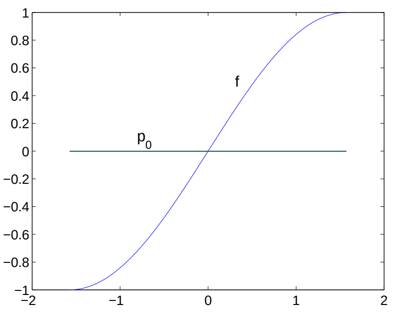


Figure 1: The Error of a Degree Zero Polynomial Interpolant over $[a, b]$

To illustrate, consider a degree 0 polynomial interpolating some function f, as shown in Figure 1. The difference between p_0 and f is determined by how quickly f increases (or decreases), and is thus encapsulated in the first derivative.

Similarly, consider the error of a degree 1 polynomial interpolant, as shown in Figure 2. In this case, the first derivative does not capture the difference between p_1 and f, since a large first derivative over $[a, b]$ can be approximated well by a line with large slope. In this case, the concavity

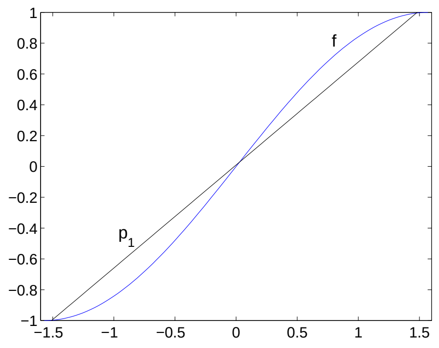


Figure 2: The Error of a Degree One Polynomial Interpolant over $[a, b]$

of the function is the main source of the error, so that the second derivative describes how well the polynomial interpolant fits the function.

As we shall see, there are two main ways that polynomial interpolation error can become unmanageable.

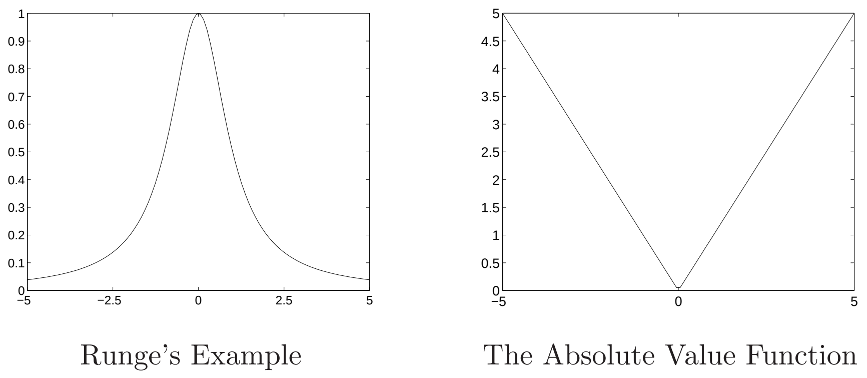


Figure 3: Two Examples of Bad Functions for Polynomial Interpolation

1. The function f that we are interpolating may simply be a bad function - it may not be differentiable over the interval $[a, b]$. Note that, even if a function is differentiable everywhere on an interval except at a single point, the error could be large over the entire interval (not just near the "bad" point). An example of this is the absolute value function, $f(t) = |t|$ over any interval containing 0, as shown in Figure 3

   We may be able to minimize the global effects of a "bad" point like this through different interpolation methods, or through careful selection of the interpolation points, but we will never be able to fully overcome the error near the "bad" point. You may notice the recurring theme - there are "bad" functions that resist numerical analysis no matter how sophisticated the methods we use.

2. The function f may have a derivative that grows very fast at the higher orders. This means, that, the more points we choose for such functions, the worse the error may become. The function f need not be very complex for this to occur, for example Runge's "bell" function (as shown in Figure 3)

$$f(t) = \frac{1}{1+t^2}$$

is quite regular, a simple quotient of polynomials, but the error of the polynomial interpolant of this function over an interval like $[5, -5]$ gets larger as the number of interpolation points goes up. We can control the error somewhat through careful selection of the interpolation points, but for an effective interpolation, we need another method altogether.

1.1 Selection of Interpolation Points

If we examine the error formula for polynomial interpolation over an interval $[a, b]$:

$$E(t) = p_n(t) - f(t) = \frac{f^{(n+1)}(c)}{(n+1)!} \cdot \left(\underbrace{\prod_{i=0}^{n} (t - x_i)}_{*} \right) \qquad \text{(for some } c \in [a, b]) \tag{2}$$

we see that we can reduce the error by selecting interpolation points $x_0, x_1, \dots, x_n$ so as to minimize the product (*).

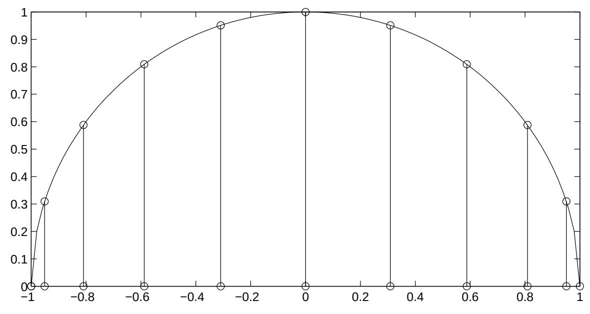


Figure 4: Chebyshev Point Distribution

To do this, conceptually, we would like to take many points near the endpoints of the interval and few near the middle. The point distribution that minimizes the product (*) is called the *Chebyshev distribution*, as shown in Figure 4. In the Chebyshev distribution, we proceed as follows:

1. Draw the semicircle on $[a, b]$.
2. To sample $n + 1$ points, place $n + 1$ equidistant points on the semicircle.
3. Project each point onto the x-axis:

$$x_j = \frac{b-a}{2} \cdot \cos\left(j \cdot \frac{\pi}{n}\right) + \frac{a+b}{2} \qquad \text{(for } j = 0, 1, \dots, n) \tag{3}$$

We can understand the formula 3 in three stages:

- Case 1: interval of interpolation is [-1,1]. Draw a unit circle centered at the origin, as shown in figure 4. Choose n arcs between $0, \frac{\pi}{n}, \frac{2\pi}{n}, \ldots, \frac{(n-1)\pi}{n}, 1$. Then project onto the circle, obtaining:

$$x_0 = \cos 0 = 0, x_1 = \cos\frac{\pi}{n}, x_2 = \cos\frac{2\pi}{n}, \ldots, x_n = \cos\frac{n\pi}{n} = 1$$

- Case 2: interval of interpolation is $[-a, a]$. Scale the unit circle from Case 1 by a. Thus also scale the points x_j by a, obtaining:

$$x_j = a\cos\frac{j\pi}{n}$$

- Case 3: interval of interpolation is $[\alpha, \beta]$. Scale the unit circle from Case 1 by $a = \frac{\beta-\alpha}{2}$, and shift by $\frac{\alpha+\beta}{2}$. Thus also scale and shift the points x_j by the same amount, obtaining:

$$x_j = \frac{\beta-\alpha}{2}\cos\frac{j\pi}{n} + \frac{\alpha+\beta}{2}$$

  This is Formula 3.

Another question we may ask is where to put points if we already have $x_0, \ldots, x_n$ given, and we want to add $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ from the interval $[a, b]$ to get the best improvement in error. A good heuristic: For $j = 1, \ldots, m$,

1. Compute the error formula based on the points so far, i.e., $x_0, \ldots, x_{n+j}$.
2. Put x_{n+j+1} at the point in $[a, b]$ that maximizes the error formula from Step 1.

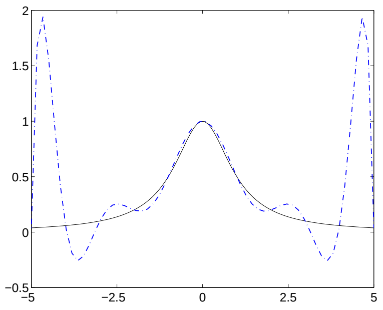
This approach is probably competitive with an optimal placement of additional points for $x_{n+1}, \ldots, x_{n+m}$; finding the optimal placement is probably NP hard.

1.2 Comparison of Interpolation Methods

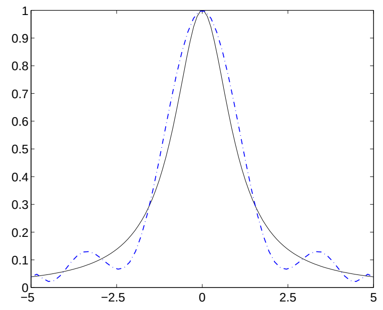
How big an effect can the selection of points have? Figure 5 shows Runge's "bell" function interpolated over $[-5, 5]$ using equidistant points, points selected from the Chebyshev distribution, and a new method called *splines*. The polynomial interpolation using Chebyshev points does a much better job than the interpolation using equidistant points, but neither does as well as the splines method.

2 Splines

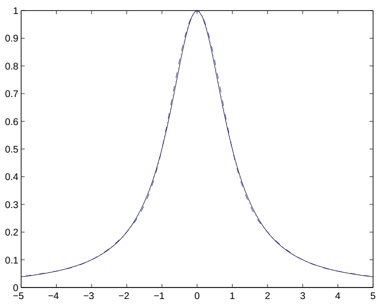
Splines usually mean piecewise polynomials, or smooth, piecewise polynomials. We will begin our discussion of splines with the simplest kind of piecewise polynomials, piecewise lines.



(a) Polynomial Interpolation (Equidistant points)



(b) Polynomial Interpolation (Chebyshev points)



(c) Interpolation Using Splines

Figure 5: Comparison of Interpolation Methods on Runge's Example.

2.1 Interpolation by Piecewise Linear Functions

Instead of attempting to interpolate a function over a large interval $[a, b]$, we may divide the interval into small subintervals of equal size, then interpolate each of the subintervals using a line through the endpoints, as shown in Figure 6.

This approach is very simple! But is it good? The resulting curve is a little ugly, and perhaps it is bad for graphics applications. However, for many applications all we care about is the error, and we will see that we can control this well.

Notice the difference from our previous strategy: as n increases, instead of increasing the degree of the polynomial, we make the mesh finer and keep the degree the same.

Now we compute the error. If we divide $[a, b]$ into N subintervals, each will have length $h = (b - a)/N$. If we consider a single subinterval $[x_i, x_{i+1}]$, by equation 1 the error will be:

$$E(t) = |\frac{f''(c)}{2} \cdot (t - x_i)(t - x_{i+1})| \qquad (t, c \in [x_i, x_{i+1}])$$

To bound the error over the entire interval $[a, b]$, we'll have to consider the maximum possible value for $|f''|$ and the maximum possible value of $|(t - x_i)(t - x_{i+1})|$ in any particular subinterval. We know from elementary calculus that the maximum of the quadratic $(t - x_i)(t - x_{i+1})$ over $[x_i, x_{i+1}]$ occurs at the midpoint, and, since each interval has length h, this makes the maximum $(h/2)^2 = h^2/4$.

To denote the maximum of f'' over $[a, b]$, we define the following notation:

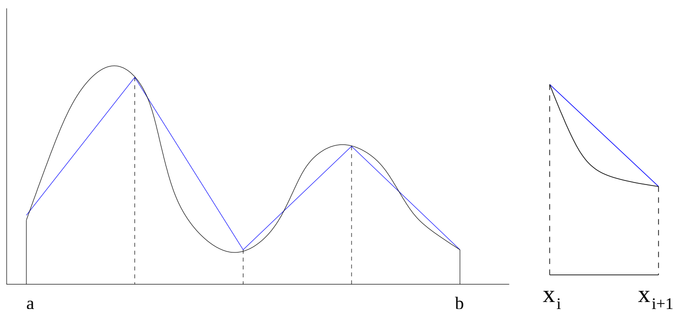


Figure 6: Interpolation using Piecewise Linear Functions

Definition 2.1 (Infinity Norm). *The infinity norm of a function $f : \mathbb{R} \to \mathbb{R}$, is denoted $\|f\|_\infty$ and is defined as*

$$\|f\|_\infty = \max_{i \in \mathbb{R}} |f(i)|$$

The infinity norm of a function $f : \mathbb{R} \to \mathbb{R}$ over an interval $[a, b]$ is denoted $\|f\|_{\infty,[a,b]}$ and is defined as

$$\|f\|_{\infty,[a,b]} = \max_{i \in [a,b]} |f(i)|$$

Using this new notation, we can write the following bound for the error for any $t \in [a, b]$.

$$E(t) \leq \frac{\|f''\|_{\infty,[a,b]}}{8} \cdot h^2 \tag{4}$$

This new expression for the error has many important features

- The error goes down as we increase the number of interpolation points (since h shrinks). In fact, the error decreases as the square of the size of the interval. In computer science, we say that the error is $O(h^2)$ (meaning that it is bounded above by some constant times h^2).
- This analysis requires less regularity of the function than standard polynomial interpolation. We require only that the function is twice differentiable.
- The size of the error does not depend on the higher order derivatives of the function. Functions like Runge's "bell" function can be effectively approximated using this method.

Unfortunately, the function approximations that this method produces are not smooth, which affects both their appearance and their differentiability. Additionally, while the error is $O(h^2)$, there are interpolation methods for which the error is $O(h^4)$, as we'll see.

2.2 Improving the Approximation Using Polynomials of Higher Degree

A natural idea for improving this approximation method is to use higher-order polynomials. Again, we split up the interval $[a, b]$ into smaller subintervals of length $h = (b - a)/2$. To interpolate over a subinterval $[x_i, x_{i+1}]$, we consider the subintervals to its left ($[x_{i-1}, x_i]$) and right ($[x_{i+1}, x_{i+2}]$).

Taking the endpoints of each subinterval, this yields four interpolation points $(x_{i-1}, x_i, x_{i+1}, x_{i+2})$, which we use to find a cubic polynomial $p_3 \in \Pi_3$. We will use this polynomial as an interpolant only over the interval $[x_i, x_{i+1}]$. If we perform this process over each subinterval (other than the first and last subinterval), the resulting piecewise cubic function will have error that is $O(h^4)$. We will discuss this method in more detail next time.