Polynomial Interpolation and Runge's Phenomenon

Tanner Wagner

March 29, 2024

1 Polynomial Interpolation | Overview

- Say you want to approximate some complex function. How and why would you do this? Let's first think about this intuitively based on the following image.

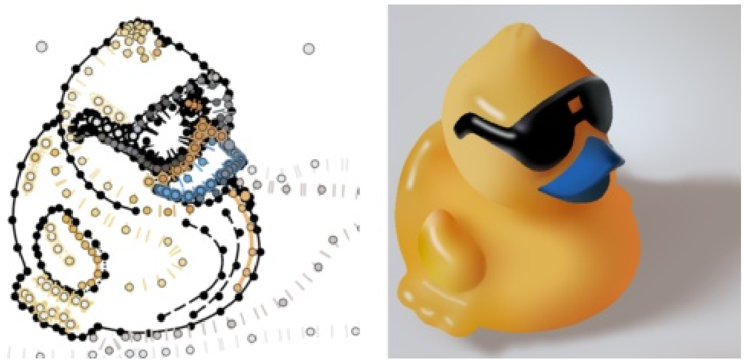

Figure 1: Freeform vector graphics with controlled thin-plate splines.

- Think about the simpler functions that could make up a more complex function.
- This is the basic thought process behind polynomial interpolation.
- Given a set of $n+1$ data points $(x_0, y_0), \ldots, (x_n, y_n)$, with now two x_j the same, a polynomial function $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ is said to <u>interpolate</u> the data if $p(x_j) = y_j \ \forall j \in \{0, 1, \ldots, n\}$.
- This is the general idea behind polynomial interpolation but when it comes to actually finding such a polynomial there are many techniques that each have their own benefits and consequences.

- Before we cover some of the methods of polynomial interpolation, let's first cover what is referred to as the "Interpolation Theorem."

Theorem 1 (Interpolation Theorem)**.** *For any $n+1$ bivariate data points $(x_0, y_0), \dots, (x_n, y_n) \in \mathbb{R}^2$, where no two x_j are the same, $\exists$ $p(x)$ of degree at most n that interpolates these points, i.e., $p(x_0) = y_0, \dots, p(x_n) = y_n$. Equivalently, for a fixed choice of interpolation nodes x_j, polynomial interpolation defines a linear bijection L_n between the $(n+1)$-tuples of real-number values $(y_0, \dots, y_n) \in \mathbb{R}^{n+1}$ and the vector space $P(n)$ of real polynomials of degree at most n, i.e., $L_n : \mathbb{R}^{n+1} \to P(n)$.*

- Now that we're familiar with what polynomial interpolation is and what we are looking for, let's cover some ways in which we can actually calculate this polynomial.

Monomial Basis

- One way in which we can do this is through something called the monomial basis.
- Let the polynomial have the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- We apply $n+1$ conditions to the polynomial

$$p(x_i) = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n = y_i, i = 0, \dots, n$$

- Or

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- That is to say

$$\mathbf{a} = V^{-1}\mathbf{y}$$

- Though this seems fairly simple to set up and solve, this system is terribly ill-conditioned and is generally favored for better techniques.

Lagrange Interpolation

- The whole idea behind this method is to use these things called the Lagrange basis functions and use them as sort of on and off switches. Doing this, we can find an interpolating polynomial $p_n(x)$ given $n+1$ data points.

- This approach is fairly simple and all we have to do is define the general Lagrange form as

$$\ell_j(x) = \prod_{\substack{0\le m\le n \\ m\neq j}} \frac{x - x_m}{x_j - x_m}$$

- The resulting polynomial is then

$$p(x) = \sum_{j=0}^{n} y_j \ell_j(x)$$

Newton Polynomials

- An even more stable approach than the monomial basis and Lagrange interpolation is to use something called a Newton polynomial.
- We can express these polynomial as linear systems and they are fairly easy to solve.
- Additionally, they are generally more stable than monomials, they are almost as computationally efficient, and they are easier when it comes to adding more data points.
- Its general form can be expressed as

$$p_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n(x-x_0)(x-x_1)\dots(x-x_n)$$

- The 2nd order Newton polynomial has the form

$$p_2(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$

- The coefficients can then be represented as

$$\begin{aligned} a_0 = f[x_0] \quad &= y_0 \\ a_1 = f[x_0, x_1] \quad &= \tfrac{y_1 - y_0}{x_1 - x_0} \\ a_2 = f[x_0, x_1, x_2] \quad &= \tfrac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0} \end{aligned}$$

- Here, the brackets indicate divided differences.

Interpolation Methods Conclusion

- As it turns out, these are the very basics of interpolation. While these techniques are each useful in their own right, it is now the job of mathematicians to find better techniques or at least optimize previously existing ones.

2 Spline Interpolation | Overview

- Let's now briefly cover a technique that is better than the previous interpolation techniques.
- First, let's think about what we want to gain from spline interpolation.

  We want our interpolation to cross every data point.

  We want a polynomial.

  We do NOT want a high-degree polynomial (more later).
- With these in mind, let's walk through the basic steps of spline interpolation.
- Given: Set of points (x_i, y_i), degree of smoothness.
- Output: Continuous piecewise polynomial $S(x)$.
- Idea: Degree of the polynomial is fixed, yet we can still satisfy $S(x_i) = y_i$ $\forall$ points.
- I will proceed with caution by mentioning that there are many ways to define splines especially because of the fact that the degree of the spline warrants its own discussion. However, sense this is an overview, I will try to give a general sense of splines and try to be as clear as possible.

Definition 1 (Splines). *As previously mentioned, we want to limit our polynomial to one variable. In this case, a spline is considered a piecewise polynomial function which we call S. It takes values from an interval $[a, b]$ and maps them to $\mathbb{R}$.*

$$S : [a, b] \to \mathbb{R}$$

- Next I will outline a very general proof for this.

Proof. Because we want S to be defined as a piecewise function, let's let the interval $[a, b]$ be covered by k ordered, disjoint subintervals

$$[t_i, t_{i+1}], i = 0, \dots, k-1$$

$$[a, b] = [t_0, t_1) \cup [t_1, t_2) \cup \cdots \cup [t_{k-2}, t_{k-1}) \cup [t_{k-1}, t_k) \cup [t_k]$$

$$a = t_0 \leq t_1 \leq \cdots \leq t_{k-1} \leq t_k = b$$

On each k of $[a, b]$, we want to now define a polynomial, P_i,

$$S(t) = P_0(t) \in t_0 \leq t < t_1$$

$$S(t) = P_1(t) \in t_1 \leq t < t_2$$

$$\vdots$$

$$S(t) = P_{k-1}(t) \in t_{k-1} \leq t \leq t_k$$

□

- It's important to note that the given points t_i are generally referred to as knots.
- While there is a conversation to be had about things like cubic splines, there is simply not enough time and this is meant to be looked at as a simple overview.
- With that, I will end this discussion on splines with the following graphic.

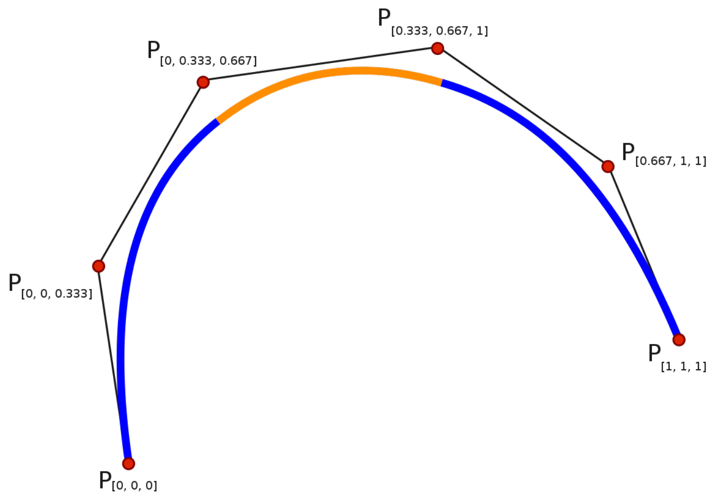

Figure 2: Splines Defining a Curve.

3 Runge's Phenomenon

- In general, Runge's phenomenon describes a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree over a set of equispaced interpolation points.

Figure 3: German Mathematician Carl Runge.

- In numerical analysis we are always concerned with accuracy. Because we are always working in the discrete case, we want to make sure we yield approximations that are numerically stable and accurate. This generally means that we want to take large samples and take as many data points as possible.

- However, it is the case with interpolation that this is not always what we want. This is what Runge's phenomenon describes. In fact, the more "accurate" we try to make our interpolating polynomial, the worse our results are.

- To understand this, we first need to understand a couple error theorems.

Theorem 2 (Interpolation Error I)**.** *If $p_n(x)$ is the at most n degree polynomial interpolating $f(x)$ at $n+1$ distinct points and if $f^{(n+1)}$ is continuous, then*

$$e(x) = f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(c) \prod_{i=0}^{n} (x - x_i) \because R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{(n+1)}$$

Theorem 3 (Bounding Lemma)**.** *Suppose x_i are equispaced in $[a,b]$ for $i = 0, \ldots, n$. Then*

$$\prod_{i=0}^{n} |x - x_i| \leq \frac{h^{n+1}}{4} n!$$

Theorem 4 (Interpolation Error II)**.** *Let $|f^{(n+1)}(x)| \leq M$, then the equispaced interpolation error has the bound*

$$|f(x) - p_n(x)| \leq \frac{M h^{n+1}}{4(n+1)}$$

- Knowing this, Runge's phenomenon is the consequence of two properties of this problem.

  The magnitude of the n-th order derivatives of this particular function grows quickly when n increases.

  The equidistance between points leads to a Lebesgue constant that increases quickly when n increases.

- We see this phenomenon graphically below because both properties combine to increase the magnitude of the oscillations.

- Runge's function is defined as

$$f(x) = \frac{1}{1 + 25x^2}$$

- The resulting interpolation is then seen as the following.

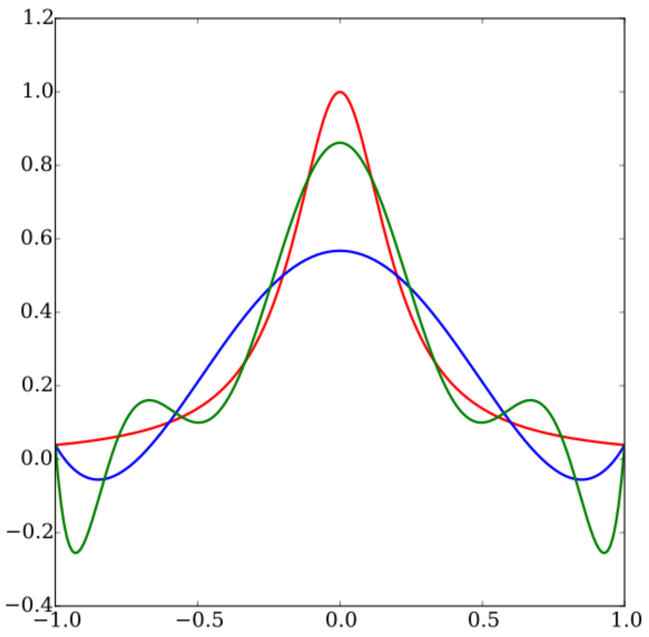

Figure 4: The following displays Runge's phenomenon where the red is Runge's function, the blue is a high order interpolation, and the green is a higher order interpolation.