MAT360 Section Summary: 3.1
Interpolation and the Lagrange Polynomial

1. Summary

Obviously, polynomials are useful functions for fitting data. When you were a kid you drew "connect the dots" pictures, but you didn't realize that you were fitting a data set with "linear splines". Now you might, or at least you can be so informed, and not terribly disturbed....

Polynomials are useful partly because of the

Weierstrass Approximation Theorem: Suppose that $f \in C[a,b]$. For each $\epsilon > 0$, there exists a polynomial $P(x)$ such that

$$|f(x) - P(x)| < \epsilon \quad \forall x \in [a,b]$$

When using the secant method, we fitted a line to two points and used that to approximate a root. Müller's method involved fitting a quadratic polynomial to three points, and we profitably used our knowledge of linear algebra to do so. That's the sort of "interpolation" that we want to engage in throughout the present chapter.

While Taylor series polynomials are very valuable throughout numerical analysis, this is one time when we want to chuck them into the garbage. They're not particularly useful for interpolating functions, because they focus their information at a point, and we're interested in fitting points over space: we want to **distribute error** over an interval, not **eliminate it at a single point** (and let it grow larger and larger as we move away from that point).

Generally, we eschew the use of high degree polynomials for the interpolation of data, for several reasons:

(a) they tend to wiggle more than we like;

(b) their derivatives (while easy to calculate) also wiggle alot;

(c) small errors in coefficients can result in dramatic changes in the polynomial.

(d) They're terrible for extrapolation: they blow up quickly once they're done with their job of fitting the data between x_0 and x_n.

In spite of this, we begin with a method of interpolating any data with a polynomial, called a **Lagrange polynomial** – because we can! Furthermore, doing so is easy.

2. Definitions

- n^{th} **Lagrange interpolating polynomial**: the unique polynomial of degree n that interpolates (that is, passes through) the $n+1$ data points $\{(x_i, f(x_i))\}_{i=0}^{n}$.

3. Theorems/Formulas

There is a very interesting way of deriving the Lagrange interpolating polynomial (linear algebra aside!). Write the n^{th} degree polynomial as a sum of n^{th} degree polynomials, as follows:

Theorem 3.2: Let $\{x_i\}_{i=0}^n$ be $n+1$ distinct numbers, and f is a function defined at those numbers, having values $\{f(x_i)\}_{i=0}^n$. Then there is a unique polynomial of degree n passing through those points, and it is given by

$$P(x) = f(x_0)L_{n,0}(x) + \ldots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)$$

where

$$L_{n,k}(x) = \frac{(x-x_0)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)} = \Pi_{i=0,i\neq k}^{n}\frac{(x-x_i)}{(x_k-x_i)}$$

Let's look at the error we're making in estimating our function (which looks familiar, actually), and examine a method for calculating the value of the interpolating polynomial at a single point (Neville's method).

Theorem 3.3: Let $\{x_i\}_{i=0}^n$ be $n+1$ distinct numbers in $[a,b]$, and $f \in C^{n+1}[a,b]$. Then $\forall x \in [a,b]\ \exists\ \xi(x) \in (a,b)$ /

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)\cdots(x-x_n)$$

The error term looks a lot like the error term of the n^{th} Taylor polynomial, except that it replaces $(x-x_0)^{n+1}$ with $(x-x_0)\cdots(x-x_n)$: i.e., it distributes the pain across the interpolation points (sometimes called the nodes, or knots) x_i.

4. Properties/Tricks/Hints/Etc.

Neville's method: is useful for calculating a value of the interpolating polynomial of degree n to $n+1$ data points at a single value of x: it is based on successive linear approximation to higher powered interpolating functions to more and more points.