Chapter 10: Polynomial Interpolation

Polynomial interpolants are rarely the end product of a numerical process. Their importance is more as building blocks for other, more complex algorithms in differentiation, integration, solutions of differential equations, approximation theory in the large, ... Interpolation is often used both to design a algorithm and to analyze its convergence properties.

10.1: General approximation and interpolation

Interpolation is a special case of approximation.

- Data fitting (discrete): Given data (x_i, y_i) find a reasonable function $v(x)$ that fits the data. If the data are acurate it may make sense to interpolate the data with $v(x)$.
- Approximating functions (continuous): For a complicated function $f(x)$ find a simpler function $v(x)$ that approximates f.

10.1: General approximation and interpolation

Why do we need interpolating functions $v(x)$?

- For prediction: if x is inside the domain of data points, we call $v(x)$ the interpolation value at x. Otherwise, we call it extrapolation.
- For manipulation: like finding approximate derivatives or integrals.
- For storage: It is usually easier to store $v(x)$, rather than the data points themselves.

10.1: General approximation and interpolation

We generally assume a *linear form* for all interpolating functions $v(x)$. We write

$$v(x) = \sum_{j=0}^{n} c_j \phi_j(x) = c_0\phi_0(x) + \ldots + c_n\phi_n(x).$$

where $\{c_j\}$ are the unknown coefficients, or parameters, determined by the data, and $\{\phi_j(x)\}$ are predetermined basis functions. We are using the language and concepts from linear algebra to describe interpolation. Linear algebra helps unify much of numerical analysis.

10.1: General approximation and interpolation

- Our first goal in interpolation is to find the scalars $\{c_j\}$.
- Use the interpolating linear combination and data points to rewrite as a matrix equation.
- Note that we assume that the number of basis functions $n + 1$ is the same as the number of data points

10.1: General approximation and interpolation

There are many types of interpolation depending on the data.

- When $\{\phi_j(x)\}$ are polynomials, we say polynomial interpolation.
- When $\{\phi_j(x)\}$ are trigonometric, we say trigonometric interpolation.
- There is also piecewise interpolation.
- We will only consider polynomial interpolation in this course.

10.1: General approximation and interpolation

We will only consider polynomial interpolation in this course of the following three types of basis polynomials $\{\phi_j(x)\}$.

- Monomial Interpolation: $p_n(x) = c_0 + c_1 x + c_2 x^2 + \ldots c_n x^n$.
- Lagrange Interpolation: $p_n(x) = y_0 L_0(x) + y_1 L_1(x) + \ldots + y_n L_n(x)$.
- Newton's: $p_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) \ldots + c_n(x - x_0)(x - x_1) \ldots (x - x_{n-1})$.
- It is important to understand that each method leads to the same answer. Yet, each method has its advantages. Some are easier to compute while others are more useful theoretically.

10.2: Monomial interpolation

- Use the polynomials $\phi_0(x) = 1, \phi_1(x) = x$ to approximate the data $(1,1),(2,3)$.
- Use the polynomials $\phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2$ to approximate the data $(1,1),(2,3),(4,3)$.

10.2: Monomial interpolation

```
# ALGORITHM: Matrix interpolation p. 300
import numpy as np

A = np.array([[1,1,1],
              [1,2,4],
              [1,4,16]], float)

y = np.array([[1],
              [3],
              [3]],float) # initial condition

c = np.linalg.solve(A,y)
print(c)
```

10.2: Monomial interpolation

This method suggests a general way for obtaining the interpolating polynomial $p(x)$.

- Form the Vandermonde matrix and solve the linear system.
- The advantage of this approach is its simplicity.
- The disadvantage of this approach is that the Vandermonde matrix X is often ill-conditioned.
- Moreover, this approach requires $\frac{2}{3}n^3$ operations. We may be able to do better?

10.3 Lagrange Interpolation

Use the lagrange polynomials

$$L_j(x) = \frac{(x - x_0)\dots(x - x_{j-1})(x - x_{j+1})\dots(x - x_n)}{(x_j - x_0)\dots(x_j - x_{j-1})(x_j - x_{j+1})\dots(x_j - x_n)}$$

to interpolate in the previous examples.

10.4 Newton's divided differences

Use Newton's divided differences to interpolate in the previous examples (by hand). You will not be required to code the Newton's divided differences algorithm in this course. Though if you comfortable coding, it is a great exercise to try to cook up code to compute the interpolating polynomial using Newton's method.

10.5 Polynomial Interpolation Error

The error in polynomial interpolation is given by

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n).$$

As it stands this formula is more or less useless in computing exact error since we can rarely find ξ and evaluate $f^{(n+1)}(\zeta)$. This error formula can however often be used to bound the error in the interpolation. You should understand this error formula and understand its proof.

10.5 Polynomial Interpolation Error

The error in polynomial interpolation is given by

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n).$$

You may expect that the higher the degree, the more accurate the interpolating polynomial. This is not always true. Low order approximations are often reasonable. High degree interpolants, on the other hand, are expensive to compute and can be poorly behaved, especially near the endpoints. Chapter 11 "Piecewise Polynomial Interpolation" partially resolves these issues by interpolating in pieces with low degree polynomials.

10.5 Polynomial Interpolation Error

Assume that the polynomial $p_9(x)$ interpolates the function $f(x) = e^{-2x}$ at the 10 evenly spaced points $x = 0, \frac{1}{9}, \frac{2}{9}, \ldots \frac{8}{9}, 1$. Use the error formula to find an upper bound for the error $e_9(0.5) = |f(0.5) - p_9(0.5)|$.