

Lecture 9: Polynomial Interpolation

2. Polynomial Interpolation.

As an application of solving linear systems, we turn to the problem of modeling a continuous real function $f : [a, b] \rightarrow \mathbb{R}$ by a polynomial. Of the several ways we might design such polynomials, we begin with *interpolation*: we will construct polynomials that exactly match f at certain fixed points in the interval $[a, b] \subset \mathbb{R}$.

2.1. Basic definitions and notation.

Definition. The set of continuous functions that map $[a, b] \subset \mathbb{R}$ to \mathbb{R} is denoted by $C[a, b]$. The set of continuous functions whose first r derivatives are also continuous on $[a, b]$ is denoted by $C^r[a, b]$. (Note that $C^0[a, b] \equiv C[a, b]$.)

Definition. The set of polynomials of degree n or less is denoted by \mathcal{P}_n .

Note that $C[a, b]$, $C^r[a, b]$ (for any $a < b$, $r \geq 0$) and \mathcal{P}_n are *linear spaces* of functions (since linear combinations of such functions maintain continuity and polynomial degree). Furthermore, note that \mathcal{P}_n is an $n + 1$ dimensional subspace of $C[a, b]$.

The polynomial interpolation problem can be stated as:

Given $f \in C[a, b]$ and $n + 1$ points $\{x_j\}_{j=0}^n$ satisfying $a \leq x_0 < x_1 < \cdots < x_n \leq b$, determine some $p_n \in \mathcal{P}_n$ such that

$$p_n(x_j) = f(x_j) \quad \text{for } j = 0, \dots, n.$$

It shall become clear why we require $n + 1$ points x_0, \dots, x_n , and no more, to determine a degree- n polynomial p_n . (You know the $n = 1$ case well: two points determine a unique line.) If the number of data points were smaller, we could construct infinitely many degree- n interpolating polynomials. Were it larger, there would in general be no degree- n interpolant.

As numerical analysts, we seek answers to the following questions:

- Does such a polynomial $p_n \in \mathcal{P}_n$ exist?
- If so, is it *unique*?
- Does $p_n \in \mathcal{P}_n$ behave like $f \in C[a, b]$ at points $x \in [a, b]$ when $x \neq x_j$ for $j = 0, \dots, n$?
- How can we compute $p_n \in \mathcal{P}_n$ *efficiently* on a computer?
- How can we compute $p_n \in \mathcal{P}_n$ *accurately* on a computer (with floating point arithmetic)?
- How should the interpolation points $\{x_j\}$ be chosen?

Regarding this last question, we should note that, in practice, we are not always able to choose the interpolation points as freely as we might like. For example, our ‘continuous function $f \in C[a, b]$ ’ could actually be a discrete list of previously collected experimental data.

2.2. Constructing interpolants in the monomial basis.

Of course, any polynomial $p_n \in \mathcal{P}_n$ can be written in the form

$$p_n(x) = c_0 + c_1x + c_2x^2 + \cdots + c_nx^n$$

for coefficients c_0, c_1, \dots, c_n . We can view this formula as an expression for p_n as a linear combination of the *basis functions* $1, x, x^2, \dots, x^n$; such basis functions are called *monomials*.

To construct the polynomial interpolant to f , we merely need to determine the proper values for the coefficients c_0, c_1, \dots, c_n in the above expansion. The interpolation conditions $p_n(x_j) = f(x_j)$ for $j = 0, \dots, n$ reduce to the equations

$$\begin{aligned} c_0 + c_1x_0 + c_2x_0^2 + \cdots + c_nx_0^n &= f(x_0) \\ c_0 + c_1x_1 + c_2x_1^2 + \cdots + c_nx_1^n &= f(x_1) \\ &\vdots \\ c_0 + c_1x_n + c_2x_n^2 + \cdots + c_nx_n^n &= f(x_n). \end{aligned}$$

Note that these $n + 1$ equations are linear in the $n + 1$ unknown parameters c_0, \dots, c_n . Thus, our problem of finding the coefficients c_0, \dots, c_n reduces to solving the linear system

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}.$$

Matrices of this form, called *Vandermonde* matrices, arise in a wide range of applications.[†] Provided all the interpolation points $\{x_j\}$ are distinct, one can show that this matrix is invertible. (In fact, the determinant takes the simple form

$$\det(\mathbf{A}) = \prod_{j=0}^n \prod_{k=j+1}^n (x_k - x_j);$$

[see MathWorld; add citation]. This is evident for $n = 1$; we will not prove it for general n , as we will have more elegant ways to establish existence and uniqueness of polynomial interpolants.) Hence, fundamental properties of linear algebra allow us to confirm that there is exactly one degree- n polynomial that interpolates f at the given $n + 1$ distinct interpolation points.

Theorem. Given $f \in C[a, b]$ and distinct points $\{x_j\}_{j=0}^n$, $a \leq x_0 < x_1 < \cdots < x_n \leq b$, there exists a unique $p_n \in \mathcal{P}_n$ such that $p_n(x_j) = f(x_j)$ for $j = 0, 1, \dots, n$.

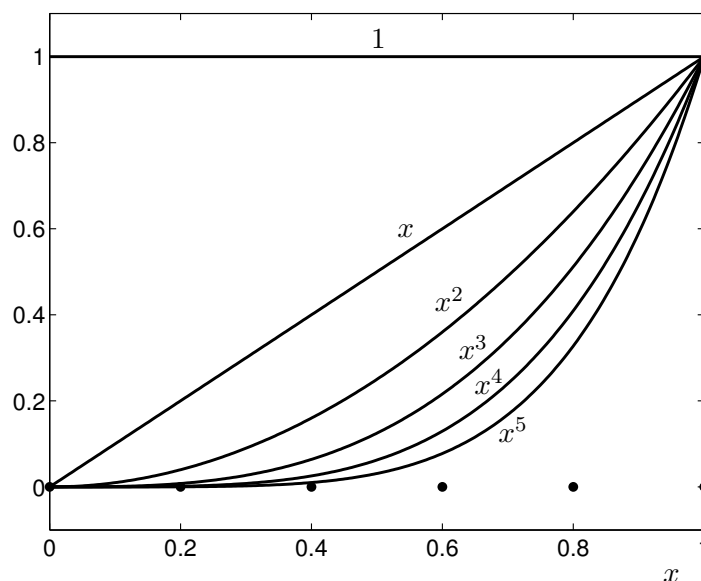
To determine the coefficients $\{c_j\}$, we could solve the above linear system using the QR decomposition of the Vandermonde matrix, as described in previous lectures. Alternatively, we could use Gaussian elimination, which we will discuss in future lectures. (This is what MATLAB's `\` command uses.) There exists a third alternative, specialized algorithms that exploit the Vandermonde

[†]Higham presents many interesting properties of Vandermonde matrices and related computations in Chapter 21 of *Accuracy and Stability of Numerical Algorithms* (SIAM, Philadelphia, 1996).

structure to determine the coefficients $\{c_j\}$ in $O(n^2)$ operations, a vast improvement over the $O(n^3)$ operations required by Gaussian elimination or QR decomposition.[‡]

2.2.1. Potential pitfalls of the monomial basis.

Though it is straightforward to see how to construct interpolating polynomials in the monomial basis, this procedure can give rise to some unpleasant numerical problems when we actually attempt to determine the coefficients $\{c_j\}$ on a computer. The primary difficulty is that the monomial basis functions $1, x, x^2, \dots, x^n$ look increasingly alike as we take higher and higher powers. The following plot illustrates this on the interval $[a, b] = [0, 1]$ with $n = 5$ and $x_j = j/5$.



Because these basis vectors become increasingly alike, one finds that the expansion coefficients $\{c_j\}$ in the monomial basis can become very large in magnitude even if the function $f(x)$ remains of modest size on $[a, b]$.

Consider the following analogy from linear algebra. The vectors

$$\begin{bmatrix} 1 \\ 10^{-10} \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

form a basis for \mathbb{R}^2 . However, both vectors point in *nearly* the same direction, though strictly speaking they are linearly independent. We can write the vector $(1, 1)^T$ as a unique linear combination of these basis vectors,

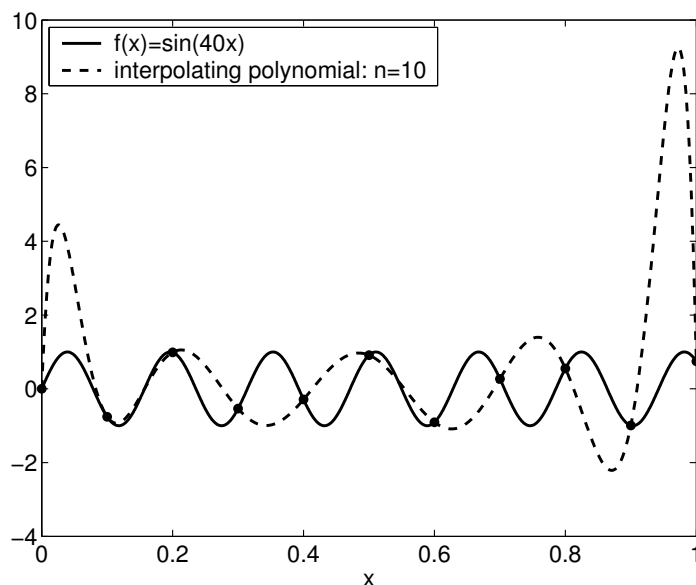
$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 10,000,000,000 \begin{bmatrix} 1 \\ 10^{-10} \end{bmatrix} - 9,999,999,999 \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Although the vector we are expanding and the basis vectors themselves are all small in norm, the expansion coefficients are enormous. Furthermore, small changes to the vector we are expanding

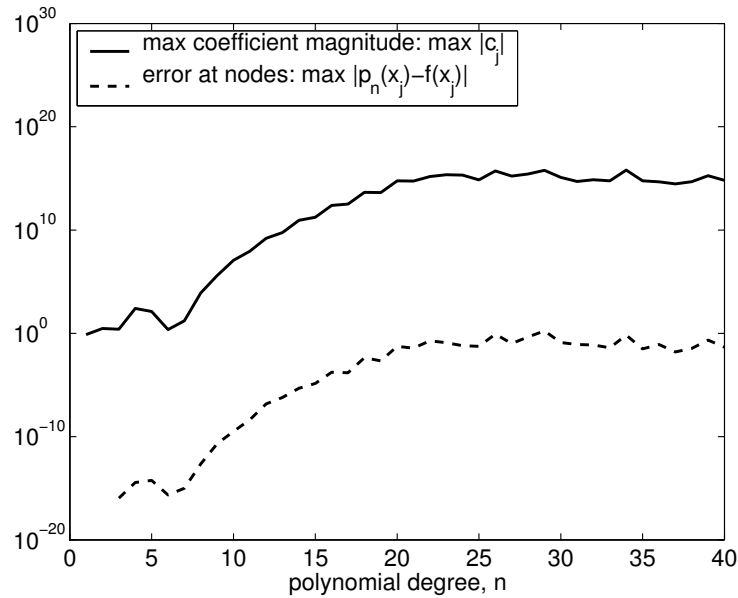
[‡]See Higham's book for details and stability analysis of specialized Vandermonde algorithms.

will lead to huge changes in the expansion coefficients. This is a recipe for disaster when computing with finite-precision arithmetic.

This same phenomenon can occur when we express polynomials in the monomial basis. As a simple example, consider interpolating $f(x) = \sin(40x)$ at uniformly spaced x_j in the interval $[0, 1]$. Note that $f \in C^\infty[0, 1]$, and $|f(x)| \in [0, 1]$. As seen in the following plot, f oscillates modestly on the interval $[0, 1]$, but it certainly does not grow excessively large in magnitude or exhibit any nasty singularities.



It turns out that as $n \rightarrow \infty$, the interpolating polynomial converges to the true function in a manner we shall make precise in coming lectures. However, our MATLAB computation that solves the Vandermonde system (using `\`) and then evaluates the polynomial (using `polyval`) is remarkably inaccurate due to the magnitude of the expansion coefficients. The plot below shows how the computed coefficients grow to roughly 10^{15} in magnitude as n increases to 40. We compare this error to the quantity $\max_{0 \leq j \leq n} |f(x_j) - p_n(x_j)|$. In theory, this quantity should be zero, since p_n interpolates f at the points $\{x_j\}$, but in practice, numerical errors pollute the evaluation of p_n .



We are prepared to accept errors of roughly 10^{-15} in size, due to the finite accuracy of the computer's floating point arithmetic. However, we see errors more like 10^{-1} : we must have higher standards! This is an example where a simple problem formulation quickly yields an algorithm, but that algorithm gives unacceptable numerical results. In the next lecture, we shall obtain improved results by expanding the interpolating polynomial in a basis that is better conditioned.