Understanding Approximating Polynomials

Maria Leszczyńska
Numerical Analysis

1 Introduction: What are polynomials, and why do we use them to approximate functions?

In many areas of applied mathematics and scientific computing, we deal with functions that are difficult (or sometimes impossible) to evaluate directly. This might be because the function involves something like e^x, $\ln(x)$, or $\sin(x)$, or because we're working with data collected from the real world that only gives us values at certain points. Either way, we often want a simpler function that can "stand in" for the original: one that's easier to work with but still captures the essential behavior of the original function.

That's where polynomial approximation comes in.

1.1 What is a polynomial?

A **polynomial** is a mathematical expression that consists of variables raised to non-negative integer powers, each multiplied by a constant coefficient, and summed together.

The general form of a univariate polynomial of degree n is:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

where the a_i are real or complex coefficients, and $a_n \neq 0$.

Each individual term $a_k x^k$ is called a monomial. The degree of the polynomial is the highest exponent of the variable x that appears with a non-zero coefficient. For example a polynomial of degree 1 is linear $[P(x) = a_1 x + a_0]$, a polynomial of degree 2 is quadratic $[P(x) = a_2 x^2 + a_1 x + a_0)]$, higher degree polynomials include cubics, quartics, and so on.

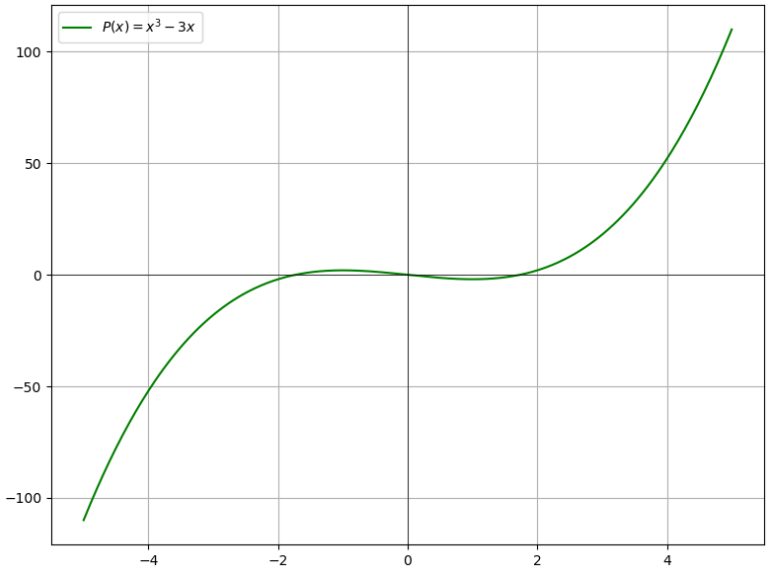

Figure 1: Example of a polynomial $P(x) = x^3 - 3x$.

Polynomials are not limited to a single variable. **Multivariate polynomials** include terms like $a_{ij}x^i y^j$, as in:

$$P(x, y) = a_{00} + a_{10}x + a_{01}y + a_{11}xy + a_{20}x^2 + \dots$$

Polynomials are among the simplest and most computationally friendly functions. They are smooth, differentiable everywhere, and easy to evaluate, integrate, and differentiate. But even more importantly, polynomials can approximate a wide class of functions on a closed interval to any degree of accuracy, which is a foundational result known as the **Weierstrass approximation theorem** (we will get to this later).

2 Why use polynomials?: Motivation and advantages

Polynomials serve as foundational tools in both local and global methods - from Taylor series to interpolation, and appear in a wide range of scientific and engineering applications.

2.1 The Weierstrass Approximation Theorem

One of the fundamental theorems justifying the use of polynomials is the **Weierstrass approximation theorem**.

Weierstrass Approximation Theorem

If f is a continuous, real-valued function on $[a, b]$ and if any $\varepsilon > 0$ is given, there exists a polynomial $P(x)$ such that:

$$|f(x) - P(x)| < \varepsilon \quad \text{for all } x \in [a, b].$$

This means that any continuous function on a closed and bounded interval can be uniformly approximated on that interval to any degree of accuracy using a polynomial.

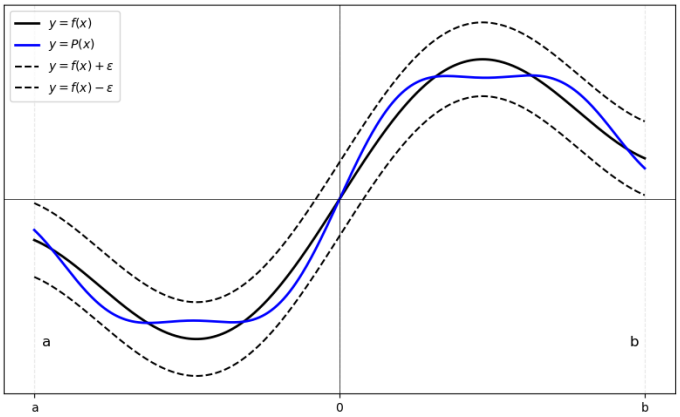

Figure 2: Example of the Weierstrass Approximation Theorem.

2.2 Computational advantages

In addition to their approximation power, polynomials are computationally attractive:

- **Easy to evaluate:** Using Horner's rule, a degree-n polynomial can be evaluated using only n multiplications and n additions.
- **Simple to differentiate and integrate:** The derivative and antiderivative of a polynomial are also polynomials.
- **Structured algebra:** Operations like addition, multiplication, and even division (via long or synthetic division) are easy to implement and analyze.

These properties make polynomials a natural fit for computer based numerical methods.

3 Local polynomial approximation: Taylor and Maclaurin series

A local polynomial approximation constructs a polynomial that closely matches a function near a specific point. The most widely used method for this is the **Taylor series**, which uses the function's derivatives at a single point to build a polynomial approximation.

Using this process, we can approximate functions like $\sin(x)$, e^x, and $\ln(1+x)$ with polynomials that closely match their behavior near a chosen point. For many functions, the Taylor series converges to the function within some interval around that point.

3.1 Taylor and Maclaurin series

Taylor Polynomial

Let f be a function that is sufficiently smooth (it has n continuous derivatives near a point $x = a$). The nth-order Taylor polynomial centered at $x = a$ is:

$$P_n(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$
$$= \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k.$$

This polynomial is an approximation to f near $x = a$. If $a = 0$, the expression is called the **Maclaurin series**.

The infinite Taylor series is given by:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k,$$

and represents the function exactly — but only if the series converges to $f(x)$.

In practice, the nth-order Taylor polynomial is simply the nth partial sum of the Taylor series. So long as the series converges to $f(x)$, the polynomial becomes more accurate as n increases.

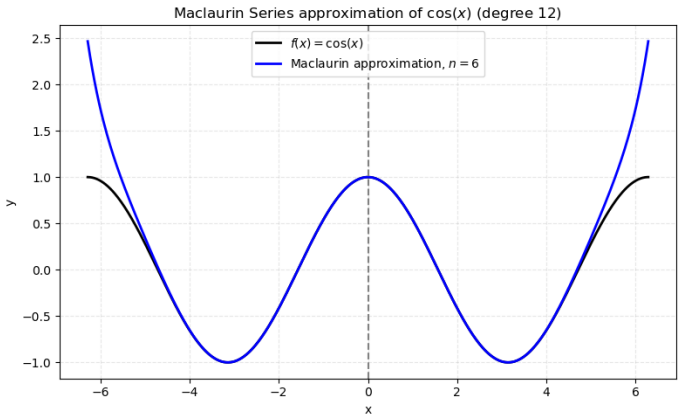

Figure 3: Example of Maclaurin series

3.2 Error and convergence

The difference between the function and its polynomial approximation is called the *remainder* or *error term*. One common form is the integral remainder:

$$f(x) - P_n(x) = \frac{(x-a)^{n+1}}{n!} \int_0^1 (1-s)^n f^{(n+1)}(a + s(x-a))\, ds.$$

A more practical estimate comes from the **Lagrange error bound**, which says:

Lagrange Error Bound

If M is an upper bound on $|f^{(n+1)}(x)|$ over the interval between a and x, then:

$$|f(x) - P_n(x)| \leq \frac{M|x-a|^{n+1}}{(n+1)!}$$

This tells us how good our approximation is and confirms that the error decreases rapidly for smooth functions when x is close to a.

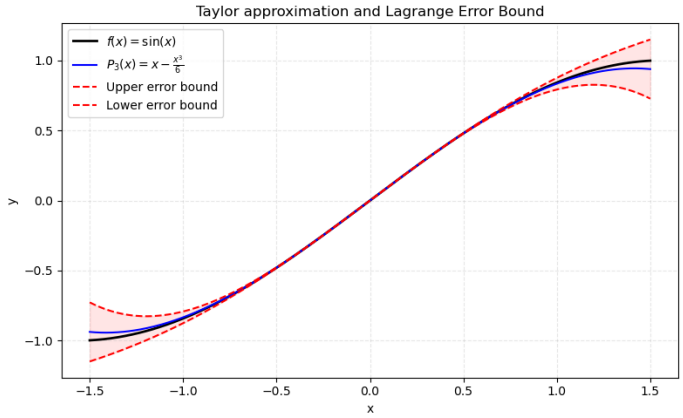

Figure 4: Taylor approximation and Lagrange Error Bound

3.3 Limitations

While Taylor series are powerful, they do have limitations:

- They require knowledge of high order derivatives (which may be hard to compute).
- The approximation is most accurate near the expansion point and often becomes less reliable farther away, this is true especially if the function's Taylor series does not converge at those points.
- Some functions have Taylor series that converge only within a limited interval, or not at all.

So, Taylor and Maclaurin polynomials are best used for local approximations near a known point. In the next sections, we'll shift from local to global polynomial interpolation, which is a method that builds approximations from values at multiple points rather than derivatives at just one.

4 Global Interpolation: different approaches

While Taylor polynomials provide local approximations based on derivatives at a single point, **global interpolation** seeks a polynomial that passes exactly through a given set of data points. This is useful when we know the values of a function at several distinct points and want a single polynomial that captures the overall trend.

Polynomial Interpolation Problem. Given $n+1$ distinct points $(x_0, f_0), \ldots, (x_n, f_n)$, find a polynomial $P_n(x)$ of degree at most n such that:

$$P_n(x_i) = f_i \quad \text{for } i = 0, \ldots, n.$$

4.1 Lagrange Interpolation

One classical method is the **Lagrange interpolation formula**, which expresses the interpolating polynomial as a linear combination of specially constructed basis functions:

$$P_n(x) = \sum_{i=0}^{n} f(x_i)\, \ell_i(x),$$

where each **Lagrange basis polynomial** $\ell_i(x)$ is defined by:

$$\ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

Each $\ell_i(x)$ is a degree-n polynomial that satisfies $\ell_i(x_j) = \delta_{ij}$, meaning it is 1 at x_i and 0 at all other data points. This guarantees that the polynomial passes exactly through the given data.

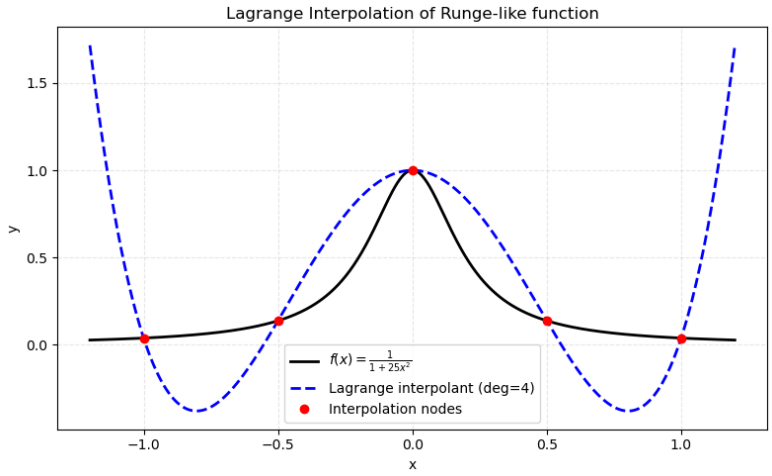

Figure 5: Lagrange interpolation of $f(x) = \frac{1}{1+25x^2}$.The polynomial passes exactly through the nodes but shows oscillation near the interval boundaries.

For example, for two points (x_0, y_0) and (x_1, y_1), the linear Lagrange interpolating polynomial is:

$$P(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1).$$

Given $n+1$ distinct points $x_0, x_1, \ldots, x_n$, there is a unique polynomial $P_n(x)$ of degree at most n that interpolates the function at those points:

$$P_n(x_i) = f(x_i), \quad \text{for all } i = 0, \ldots, n.$$

Error Formula

If the function f is sufficiently smooth, the interpolation error at a point x is given by:

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

for some $\xi(x) \in [x_0, x_n]$. This shows that the error depends both on how smooth the function is and how the interpolation nodes are spaced.

Unlike Taylor polynomials, which use information at a single point, Lagrange interpolation uses values at multiple nodes. Its error term involves a product of $(x - x_i)$ factors, reflecting sensitivity to node placement and spacing.

4.2 Newton's Divided Differences

An alternative approach to polynomial interpolation is the **Newton interpolation formula**, which is especially useful when adding new data points

incrementally.

The Newton interpolating polynomial is written as:

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots$$
$$+ f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}),$$

where $f[x_0, x_1, \ldots, x_k]$ denotes the kth order **divided difference**, computed recursively:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i},$$

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

Each coefficient $f[x_0, x_1, \ldots, x_k]$ depends only on the values of f at the nodes, and the polynomial naturally adapts if a new point is added.

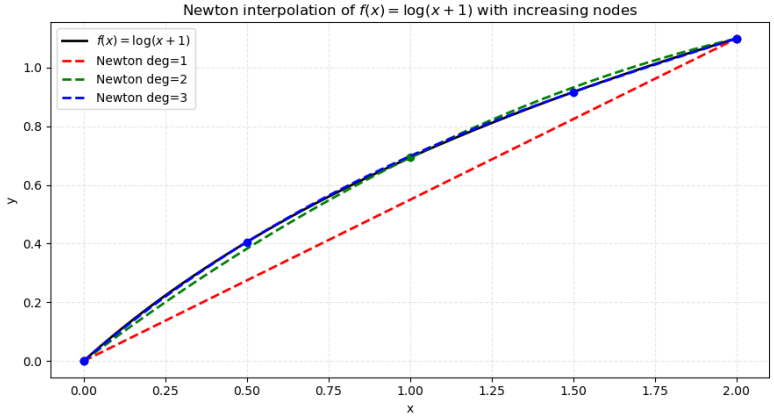

Figure 6: Newton interpolation of $f(x) = \log(x + 1)$.

Alternatively, the Newton interpolating polynomial can be expressed more compactly as:

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k] \prod_{j=0}^{k-1} (x - x_j).$$

Error Term

If $f \in C^{n+1}[a, b]$, and $x_0, x_1, \ldots, x_n$ are distinct points in $[a, b]$, then there exists some $\xi \in (a, b)$ such that:

$$f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

This result is a key part of the Newton interpolation theory: it connects the highest order divided difference to the nth derivative of the function, which is the basis for the error formula.

Using this, the error in the Newton interpolating polynomial can be written as:

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

for some $\xi \in (a, b)$.

Just like the Lagrange polynomial, the accuracy of Newton interpolation depends on the function's smoothness and the distribution of interpolation nodes.

4.3 Hermite Interpolation

Sometimes, we not only know the value of a function at certain points but also the values of one or more of its derivatives. In such cases, we use **Hermite interpolation**, which constructs a polynomial that matches both function values and derivative values at specified points.
These polynomials preserve both the value and the slope of the function at the interpolation points, ensuring a smoother and more natural fit.

For example, if $f(x_i) = y_i$ and $f'(x_i) = y_i'$ for all i, Hermite interpolation constructs a polynomial $H(x)$ such that:

$$H(x_i) = f(x_i), \quad H'(x_i) = f'(x_i).$$

This leads to a polynomial of higher degree than in the Lagrange case, typically of degree $2n + 1$ if we are using $n + 1$ points. The way it's built often involves either repeating the nodes symbolically or using a special basis that comes from the Lagrange polynomials.

Hermite Polynomial Definition. If f and f' are known at distinct nodes $x_0, x_1, \ldots, x_n$, then the Hermite polynomial of degree at most $2n+1$ that interpolates both function values and derivatives is:

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_j(x) + \sum_{j=0}^{n} f'(x_j) \widehat{H}_j(x),$$

where

$$H_j(x) = \left[1 - 2(x - x_j) L_j'(x_j)\right] (L_j(x))^2, \quad \widehat{H}_j(x) = (x - x_j)(L_j(x))^2,$$

and $L_j(x)$ is the Lagrange basis polynomial for node x_j.

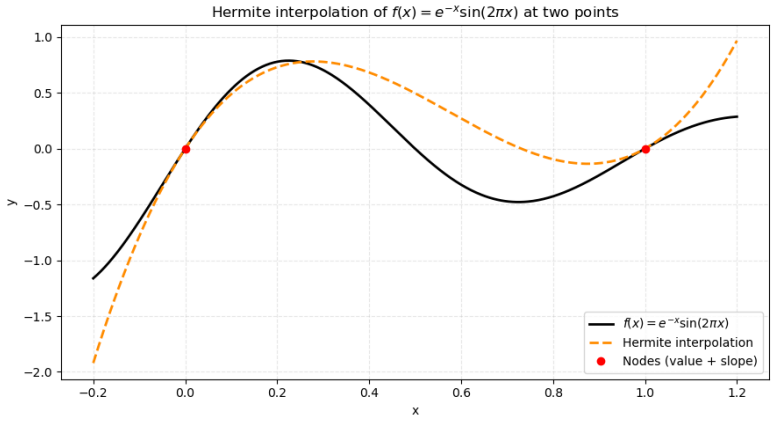

Figure 7: Hermite interpolation of $f(x) = e^{-x}\sin(2\pi x)$ using both function values and derivatives at two nodes. The resulting interpolant captures both shape and slope, even with minimal data.

Error Term

If $f \in C^{2n+2}[a, b]$, then for some unknown $\xi(x) \in (a, b)$, the error between $f(x)$ and its Hermite interpolant is given by:

$$f(x) = H_{2n+1}(x) + \frac{(x - x_0)^2(x - x_1)^2 \cdots (x - x_n)^2}{(2n + 2)!} f^{(2n+2)}(\xi(x)).$$

This shows that Hermite interpolation, just like Lagrange and Newton interpolation, depends on the smoothness of f and the distribution of interpolation nodes.

Alternative construction: divided differences

Hermite interpolation can also be constructed using an adaptation of Newton's divided-difference method. In this approach, each node x_i is repeated, and special rules are applied to compute divided differences when function values and derivatives are both known. This method is especially useful for recursive table based implementation and builds on the same recursive principles seen in Newton interpolation.

4.4 Limitations of global interpolation: Runge's phenomenon

While global polynomial interpolation works well in theory, it can run into problems in practice, especially when we use equally spaced points with high-degree polynomials. A famous example of this issue is called **Runge's phenomenon**.

This phenomenon is usually illustrated using the **Runge function**:

$$r_\gamma(x) = \frac{1}{1 + (\gamma x)^2},$$

where $\gamma \geq 1$. Even though this function is smooth and infinitely differentiable on the interval $[-1, 1]$, something unexpected happens when we try to interpolate it using high-degree polynomials at equally spaced points.

As the degree of the polynomial increases, the interpolant starts to oscillate heavily near the edges of the interval, even though the original function behaves nicely. The more nodes we add, the worse this effect becomes. For example, if we choose $\gamma = 3$ and use 11 equally spaced nodes, the interpolation error becomes very large near $x = \pm 1$.

This surprising behavior is known as **Runge's phenomenon**, and it shows that:

- High-degree interpolation with equispaced nodes can diverge or become unstable.
- Even smooth functions can be hard to approximate well, depending on their behavior in the complex plane.
- The largest errors often occur near the boundaries of the interval.

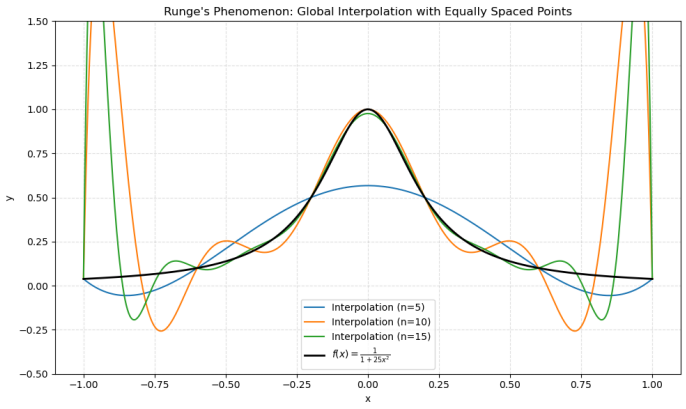

Figure 8: Example of Runge's Phenomenon

Consequences and Remedies

Because of these limitations, global polynomial interpolation should be applied with care. Some common strategies to mitigate Runge's phenomenon include:

- Using **Chebyshev nodes** instead of equally spaced points, which cluster more densely near the endpoints,

- Applying **piecewise polynomial methods**, such as splines, to reduce oscillations and improve stability,
- Reducing the degree of the polynomial by interpolating fewer points or using lower order polynomials locally.

In practice, these approaches lead to much more stable approximations, especially for smooth functions or data-driven interpolation tasks.

4.5 Chebyshev interpolation: a remedy for Runge's phenomenon

One of the most effective ways to deal with the issues caused by using equally spaced points in global polynomial interpolation is to switch to **Chebyshev interpolation**. Instead of spreading nodes evenly, Chebyshev interpolation places them closer to the ends of the interval. This change alone can greatly reduce the oscillations seen in high-degree polynomials and make the whole process much more stable.

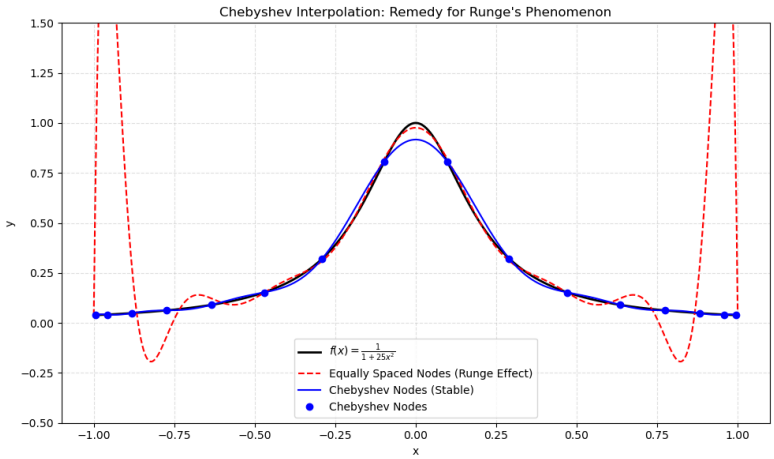

Figure 9: Example of Chebyshev Interpolation

Chebyshev Nodes

On the interval $[-1, 1]$, the Chebyshev nodes of the first kind are defined by:

$$x_j = \cos\left(\frac{(2j+1)\pi}{2(n+1)}\right), \quad j = 0, 1, \ldots, n.$$

These come from projecting evenly spaced angles on a semicircle down onto the x-axis. The result is that the nodes naturally bunch up near $x = -1$ and $x = 1$, which turns out to be exactly what's needed to prevent the wild oscillations caused by Runge's phenomenon.

Error Behavior

Using Chebyshev nodes leads to significantly smaller interpolation errors compared to equally spaced nodes. If $f \in C^{n+1}[-1,1]$, the interpolation error satisfies:

$$\|f - P_n\|_\infty \leq \frac{1}{2^n} \cdot \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \cdot \pi.$$

This exponential decay in the error makes Chebyshev interpolation highly efficient for approximating smooth functions.

Moreover, the error polynomial $\omega_{n+1}(x)$ for Chebyshev nodes has a compact expression involving Chebyshev polynomials, and its maximum absolute value on $[-1,1]$ is minimized over all choices of interpolation points.
Chebyshev nodes also lead to better numerical stability, as they limit how much the interpolation process amplifies small errors in the data

5 Piecewise polynomial approximation: Cubic splines

When global polynomial interpolation becomes unstable (especially with many points), splines offer a great alternative. A **cubic spline** is a piecewise defined function made up of third-degree polynomials, stitched together so that the whole curve is smooth. This makes them ideal for practical applications like computer graphics, data fitting, and numerical simulations.

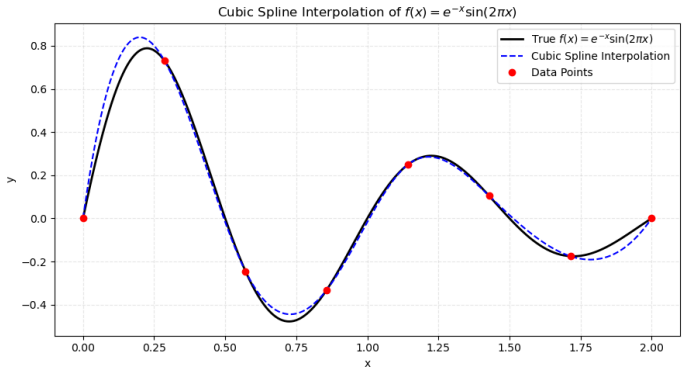

Figure 10: Cubic spline interpolation of $f(x) = e^{-x}\sin(2\pi x)$

The spline is built over a set of nodes $x_0 < x_1 < \cdots < x_n$, with one cubic polynomial on each interval $[x_j, x_{j+1}]$. These polynomials are chosen so that:

- The spline passes through all the data points,
- The whole curve is continuous and has continuous first and second derivatives,

- It satisfies boundary conditions at the endpoints (we'll come back to this).

How it works

Each segment $S_j(x)$ is a cubic polynomial defined on $[x_j, x_{j+1}]$, and the spline is stitched together so that:

(a) $S(x_j) = f(x_j)$ for all j,

(b) The function is continuous: $S_j(x_{j+1}) = S_{j+1}(x_{j+1})$,

(c) So is its first and second derivative: $S'_j(x_{j+1}) = S'_{j+1}(x_{j+1})$, and $S''_j(x_{j+1}) = S''_{j+1}(x_{j+1})$,

(d) Boundary conditions are applied: either

- **Natural spline:** second derivatives at the endpoints are zero,
- **Clamped spline:** first derivatives at the endpoints are specified.

The system that results from these conditions is tridiagonal, which makes it efficient to solve even for a large number of points.

6 Best approximation: Bernstein polynomials

Interpolation forces a polynomial to pass through known data points. But what if we only want the best possible overall fit on an interval, without insisting that the polynomial must pass through any specific values? This leads to the concept of **best approximation**.

6.1 Best approximation in the $\|\cdot\|_\infty$ norm

Given a continuous function $f \in C^0([a,b])$, we define the best approximation error from polynomials of degree at most n as:

$$d_n(f) = \inf_{P \in \mathcal{P}_n} \|f - P\|_\infty.$$

A polynomial $P^* \in \mathcal{P}_n$ is a best approximation to f if:

$$\|f - P^*\|_\infty = d_n(f).$$

The best approximation is unique, and it satisfies the **alternation theorem**: there exist $n+2$ points in $[a,b]$ where the error alternates between $\pm d_n(f)$ (its maximum and minimum value). This "equioscillation" property helps us verify that an approximation is truly optimal.

6.2 Bernstein polynomials and Weierstrass approximation

A constructive proof of the Weierstrass approximation theorem uses **Bernstein polynomials**. For $f \in C[0,1]$, the Bernstein approximation is:

$$B_n f(x) = \sum_{i=0}^{n} f\left(\frac{i}{n}\right) \binom{n}{i} x^i (1-x)^{n-i}.$$

Some important properties:

- $B_n f \to f$ uniformly on $[0,1]$ for any continuous f,
- B_n is a positive linear operator and preserves the shape of the function (for example monotonicity),
- $\|B_n\|_\infty = 1$, so it doesn't amplify errors.

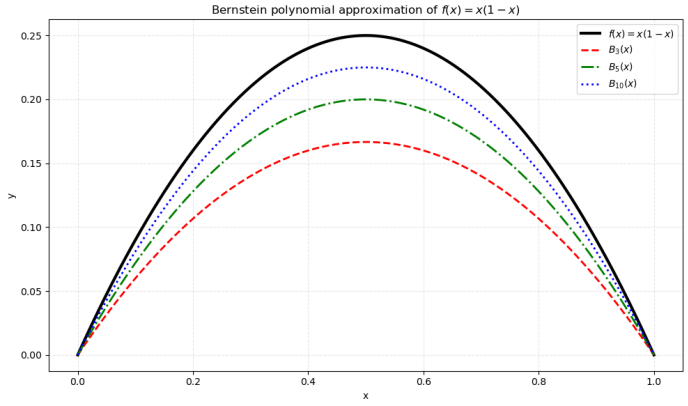

Figure 11: Bernstein polynomial approximation of $f(x) = x(1-x)$ using degrees $n = 3, 5, 10$

6.3 Modulus of continuity and convergence

The rate of convergence of $B_n f$ depends on the smoothness of f, which can be quantified using the **modulus of continuity**:

$$\omega(f;\delta) = \sup_{|x-y|\le\delta} |f(x) - f(y)|.$$

A classical result gives the approximation rate:

$$\|f - B_n f\|_\infty \le C \cdot \omega\left(f; \frac{1}{\sqrt{n}}\right),$$

showing that smoother functions are approximated more rapidly.

7 Least squares approximation: discrete and continuous

In many practical situations, we are not trying to interpolate a function exactly, but rather to *fit* a polynomial to a set of data points that may contain noise, measurement error, or be overdetermined. In such cases, interpolation is inappropriate or unstable, and we instead turn to **least squares approximation**.

7.1 Discrete Least Squares

Given a set of data points $(x_0, y_0), \ldots, (x_{m-1}, y_{m-1})$, we seek a polynomial $P_n(x)$ of degree at most $n < m$ that does not necessarily pass through all the points, but instead minimizes the total squared error:

$$\sum_{i=0}^{m-1} (y_i - P_n(x_i))^2$$

Writing $P_n(x)$ in the monomial basis, the problem becomes a standard linear least squares system. If we define $\mathbf{A}$ as the matrix of monomial terms, $\mathbf{a}$ as the vector of unknown coefficients, and $\mathbf{y}$ as the data values, the optimal solution satisfies the normal equations:

$$\mathbf{A}^T\mathbf{A}\mathbf{a} = \mathbf{A}^T\mathbf{y}$$

This gives a polynomial that smooths out the data rather than trying to interpolate every point. It is especially useful when the data contains noise or irregular variation.

7.2 Choice of degree

The choice of degree n is a balance: too small, and the polynomial underfits the data; too large, and it overfits by modeling noise rather than structure. This is similar to model selection in statistics and machine learning.

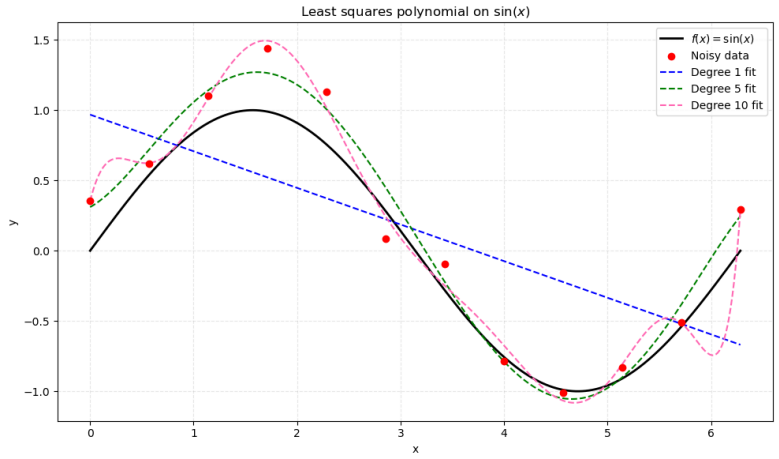

Figure 12: Discrete least squares approximation of $f(x) = \sin(x)$.

7.3 Least squares on a continuous interval

Least squares approximation can also be defined over a continuous interval. Given $f \in C^0([a, b])$, we look for a polynomial $P_n \in \mathcal{P}_n$ that minimizes:

$$\|f - P_n\|_2 = \left(\int_a^b (f(x) - P_n(x))^2 w(x)\, dx \right)^{1/2}$$

where $w(x) > 0$ is a weight function. This defines an inner product:

$$\langle f, g \rangle = \int_a^b f(x) g(x) w(x)\, dx$$

The best approximation satisfies:

$$\langle f - P_n, q \rangle = 0 \quad \text{for all } q \in \mathcal{P}_n$$

7.4 Orthogonal polynomial basis

Rather than solving the normal equations directly, we can express P_n in terms of orthogonal polynomials $\{P_0, P_1, \ldots, P_n\}$ with respect to the inner product above. These satisfy:

$$\langle P_i, P_j \rangle = 0 \quad \text{for } i \neq j$$

and give a convenient formula for the approximation:

$$P_n(x) = \sum_{i=0}^{n} \frac{\langle f, P_i \rangle}{\langle P_i, P_i \rangle} P_i(x)$$

Well-known examples include:

- **Legendre polynomials**, orthogonal on $[-1, 1]$ with $w(x) = 1$,
- **Chebyshev polynomials**, orthogonal on $[-1, 1]$ with $w(x) = \frac{1}{\sqrt{1-x^2}}$.

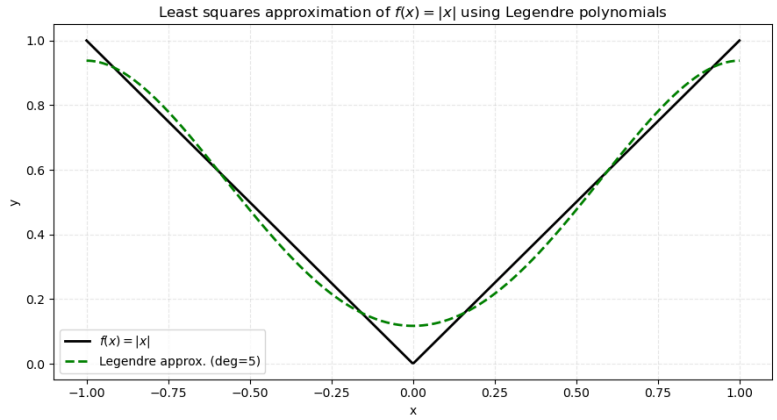

Figure 13: Least squares approximation of $f(x) = |x|$ on $[-1, 1]$ using Legendre polynomials up to degree 5.

Orthogonal polynomials are useful because:

- Their roots are real and lie inside the interval,
- Their roots make good interpolation points,
- They form a complete and stable basis for approximation.

8 Conclusion

Polynomial approximation plays a central role in numerical analysis. Whether we are dealing with smooth functions or noisy data, it provides a flexible way to simplify complex behavior using something much easier to compute and analyze.

In this essay, we explored a variety of polynomial methods:

- **Taylor and Maclaurin series** offer powerful local approximations based on derivatives.
- **Lagrange and Newton interpolation** build global polynomials that pass through specific points.
- **Hermite interpolation** extends this idea to include both function values and derivative information.
- **Chebyshev interpolation** helps overcome the problems of high-degree polynomials with equally spaced nodes.

- **Cubic splines** provide a smooth, piecewise alternative that avoids instability.
- **Bernstein polynomials** introduce the idea of best approximation in a uniform sense.
- **Least squares methods** are especially helpful for uncertain or noisy data, where interpolation may not be appropriate.

Each method comes with its own strengths and limitations. Some are designed for precision near a single point, others aim for a good fit across an interval. Some require derivative information, while others work directly with known values. The choice depends on the context: the structure of the problem, the available data, and what kind of approximation we are looking for.
Understanding these techniques is not just a theoretical exercise. They are used in practice: in computer graphics, machine learning, differential equations, and data fitting. In the end, polynomials are more than algebraic expressions. They are essential tools for making complex functions more manageable, both conceptually and computationally.

References

- Burden R. L, Faires J. D. (2010) *Numerical Analysis. Ninth Edition.*
- Ridgway S. L. (2011). *Numerical Analysis*
- Polynomial Interpolation: Newton Interpolating Polynomials - Engineering at Alberta
- Weierstrass Approximation Theorem - Wolfram MathWorld
- Taylor Polynomials and Taylor Series - LibreTexts Mathematics