The Remez Algorithm

Deval Deliwala

July 18, 2026

At the lowest level, microprocessors are made of logic gates that only perform basic arithmetic. Computers only natively add, subtract, multiply, and shift bits.

This is why polynomials are everywhere in numerical analysis. They are the simplest, smoothest, and most computationally friendly functions available. Like Lego bricks, they let us continuously build towards and approximate any complex function to high accuracy, with only native/pure arithmetic.

Design Problem

Mathematically, the Weierstrass Approximation Theorem says *every* continuous function defined on a closed interval can be *uniformly* approximated to *any* degree of accuracy with a polynomial function.

However, a normal `float32` has only 24 bits of significand precision. So once a polyomial approximation error is below roughly $2^{-24} \sim 10^{-8}$, making the polynomial more accurate with more degrees is nonessential and using it would cost more time.[1]

> Find the lowest-degree polynomial whose approximation error, together with floating-point evaluation error, remains within the desired `float32` accuracy over an input interval.

Taylor Polynomial

Consider calculating e^x. Take its degree-5 Maclaurin polynomial:

$$T_5(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120},$$

At $x = 0.1$,

$$|e^{0.1} - T_5(0.1)| = 1.41 \times 10^{-9},$$

which is comfortably below the `float32` rounding scale near 1. However, at $x = 0.2$, the absolute error rises to 9.15×10^{-8}. And at $x = 0.3$, it's already $\sim 10^{-6}$, roughly 100 times too large.

Of course, this Maclaurin polynomial is most accurate near 0. Similarly, Taylor polynomials are only accurate near a single point. But numerical libraries need to handle a *range* of possible inputs, not just one.

[1] More generally, error depends on the output magnitude and is measured in ULPs. Evaluating the polynomial also introduces rounding error, so polynomial approximation error is only a part of the issue, but is the most important to resolve.

Minimax Polynomial

Polynomials always create a game of balance. Taylor polynomials concentrate their accuracy near a single point. But if you want a polynomial that is highly accurate for many points, the level of accuracy gets *spread* down across the interval.

So the natural question is:

> Is there a polynomial q of degree n that approximates a given function in an interval such that *the absolute maximum error in that interval* is minimized?

Such a polynomial does not minimize error at each individual point. Instead, it makes the largest error *anywhere in the interval* as small as possible. This worst-case guarantee is precisely what numerical libraries need.

Existence: Let f be a continuous function. Let P_n contain all polynomials of degree at most n. The goal is to show that some $p^* \in P_n$ minimizes

$$E(p) = \|f - p\|_\infty = \max_{x \in [a,b]} |f(x) - p(x)|,$$

the maximum absolute error over all inputs in $[a, b]$.

Suppose $p = 0$ is the zero-polynomial. Its error is

$$M = \|f\|_\infty = \max_{x \in [a,b]} |f(x)|.$$

We now have a crude upper bound for the maximum absolute error. Any best polynomial p^*, if it exists, would have $E(p^*) \leq M$. So we restrict our attention to a smaller set of polynomials,

$$S = \{p \in P_n \mid E(p) \leq M\}.$$

For every $p \in S$, via the triangle inequality,

$$\|p\|_\infty \leq \|p - f\|_\infty + \|f\|_\infty \leq 2M.$$

So every polynomial $p \in S$ is uniformly bounded by $2M$.

The error function E is Lipschitz continuous, since

$$|E(p) - E(q)| \leq \|p - q\|_\infty.$$

Hence, $S = \{p \in P_n \mid E(p) \leq M\}$ is closed. P_n is also finite-$n + 1$-dimensional, so by the Heine–Borel theorem, its subset S is compact.

And by the Extreme Value Theorem, a continuous function on a compact set attains a minimum.

Therefore, there exists some $p^* \in P_n$ such that

$$\|f - p^*\|_\infty = \min_{p \in P_n} \|f - p\|_\infty.$$

So this best polynomial exists. It's known as the **minimax polynomial**. Chebyshev also proved that it is unique and gave a criterion that characterizes it.

Chebyshev's criterion states

> A polynomial $p^* \in P_n$ is the minimax polynomial if and only if its signed error $f - p^*$ attains $\pm E(p^*)$ at least $n + 2$ ordered points, with signs alternating from one point to the next.

This is known as the Chebyshev Equioscillation Theorem. Note that the $n + 2$ points do not have to be evenly spaced. They just have to exist inside the interval $[a, b]$.

Now the natural question is

> Given a continuous function f and an interval $[a, b]$, how do you calculate its minimax polynomial p^*?

The Remez Algorithm

We want to minimize

$$E(p) = \max_{x \in [a,b]} |f(x) - p(x)|.$$

The minimax polynomial p^* is the unique polynomial for which this maximum error, $E(p^*)$, is as small as possible.

The signed error $f(x) - p^*(x)$ attains the alternating values $\pm E(p^*)$ at at least $n + 2$ points in $[a, b]$, by the Equioscillation theorem.

The Remez algorithm finds the minimax polynomial by repeatedly forcing the error to become more evenly balanced across the interval. Unless the largest positive and negative error peaks are equally balanced in an alternating pattern, the polynomial can still be "tilted" to reduce its worst error.

Algorithm

The unknowns are

> 1. The coefficients $c_0, \ldots, c_n$ that define the polynomial.
> 2. The alternating error height ε at the current ordered points.

Hence there are $n + 2$ unknowns. During an intermediate iteration, $|\varepsilon|$ is not yet necessarily the maximum error over the interval. It becomes the minimax error only at convergence. The algorithm is given below.

Ansatz

Choose any $n + 2$ ordered points

$$a \leq x_0 < x_1 < \cdots < x_{n+1} \leq b.$$

The Remez Algorithm is *an iterative algorithm*. We already start by trying to make a minimax-like polynomial by requiring these points to have alternating errors of magnitude $|\varepsilon|$ like in the Equioscillation theorem.

$$f(x_i) - p_n(x_i) = (-1)^i \varepsilon, \qquad i = 0, ..., n+1.$$

Writing out $p_{n(x_i)}$,

$$c_0 + c_1 x_i + \cdots + c_n x_i^n + (-1)^i \varepsilon = f(x_i).$$

This gives $n+2$ linear equations for $n+2$ unknowns $c_0, ..., c_n, \varepsilon$. After solving this system, the result is the unique polynomial whose errors at all x_i are equal in magnitude $|\varepsilon|$ and alternate in sign.

But this likely isn't the minimax yet because the x_i were arbitrarily chosen. There may be some unchosen point where the error is larger than $|\varepsilon|$.

We test this by scanning across the interval $[a, b]$ for extrema of the signed error, including the point with the true largest error magnitude,

$$E_{\max} = \max_{x \in [a,b]} |f(x) - p_n(x)|.$$

If this is larger than $|\varepsilon|$, then the current polynomial is not the minimax. There is a larger peak somewhere that was not included in the previous system of equations; it is possible to spread the error more evenly.

Iteration

Suppose the largest error magnitude $E_{\max}$ occurs at x_{new}. We insert x_{new} into the ordered list of previous points x_i.

This now gives $n+3$ points, so one point must be removed. If x_{new} lies between two old points, the Remez exchange rule removes the neighboring point whose error has the same sign as the error at x_{new}. Endpoint cases are handled similarly. This preserves an ordered set of $n+2$ points whose error signs alternate.

The idea is simple:

1. The previous polynomial balanced the error only at the old points.
2. The newly discovered point reveals where that balance failed.
3. Include that point in the next balancing problem.
4. Discard the old point selected by the exchange rule.

Then solve the linear system again using the updated $n+2$ points. This process is repeated. As it continues, $|\varepsilon|$ gets closer and closer to newly found $E_{\max}$'s when scanning after an iteration. That is, the magnitudes of the error peaks get more and more balanced.

The algorithm stops when, $E_{\max} \approx |\varepsilon|$. No more significant progress is being made; $|\varepsilon|$ is already very close to the true minimax error.

The final optimized coefficients $c_0, ..., c_n$ define the near-minimax polynomial p^*. The final $|\varepsilon|$ gives its near-minimax error $E(p^*)$.

Now that the minimax polynomial is understood, we can begin implementing `vex`, "vectorized elementary functions" in C. It's just a personal project, like `lak`, to help me learn C, ARM NEON intrinsics, and computational math together.

We will try to write routines to calculate nontrivial functions, like e^x, $\sin x$, $\cos x$, $\log x$, and others, that are as fast as possible on Apple Silicon, while keeping the code elegant and readable.