1D Function Approximation

Polynomials, Rational Functions, and the Bag of Tricks to Maximize Accuracy Per Computation

How do we get the numerical value of functions at *every* value of x?

- For polynomials, we can multiply and add
  - $P_N(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N$
- For rational functions, we multiply and add, then divide
  - $R_{[N/M]}(x) = \frac{P_N(x)}{Q_M(x)} = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N}{1 + b_1 x + b_2 x^2 + \cdots + b_M x^M}$
- That's all arithmetic (PEMDAS) on numbers that computers do in hardware, what about everything else?
  - Trigonometric functions at not nice angles?
  - Exponential functions with irrational base or exponent?
  - Logs?
  - Roots of real numbers?
  - "Special" functions?
- Answer: we approximate using the top two and characterize the approximation error! If it's the same order as finite precision rounding errors (or we can otherwise tolerate the error at the system level), then we're good to go.
  - Why? Our CPUs/GPUs/TPUs/DSPs/FPGAs can do multiply-add-divide in hardware, other functions are far less common
- Absolute approximation error: $e(x) = |P_N(x) - f(x)|$ or $e(x) = \left|R_{[N/M]}(x) - f(x)\right|$

Motivation

- Its cool to know how you really calculate something like sine or erfc as accurately as you like
- But this also will demonstrate the path to efficiently calculate those things on real hardware for speed
- Specifically, there’s one NN activation function that’s everywhere these days that’s based on special functions derived from the Gaussian function: GELU, more on this near the second half of the talk

Minimizing Local Error – Polynomials

- Let's say I really care about $f(x) = \log(x+1)$ around $x = 0$
  - Near zero -> local error
  - I'll use $x = 0$ like a reference point because you can always do $g(x) = f(x - x_0)$ to shift around that function and do the same analysis on g
- What's the best constant to use to approximate f? $f(0) = 0$, so zero is a good first approximation, $\mathrm{P}_0(x) = 0$
- What's the best linear function around $f(0)$?

$$\mathrm{P}_1(x) = f(0) + f'(0)x$$

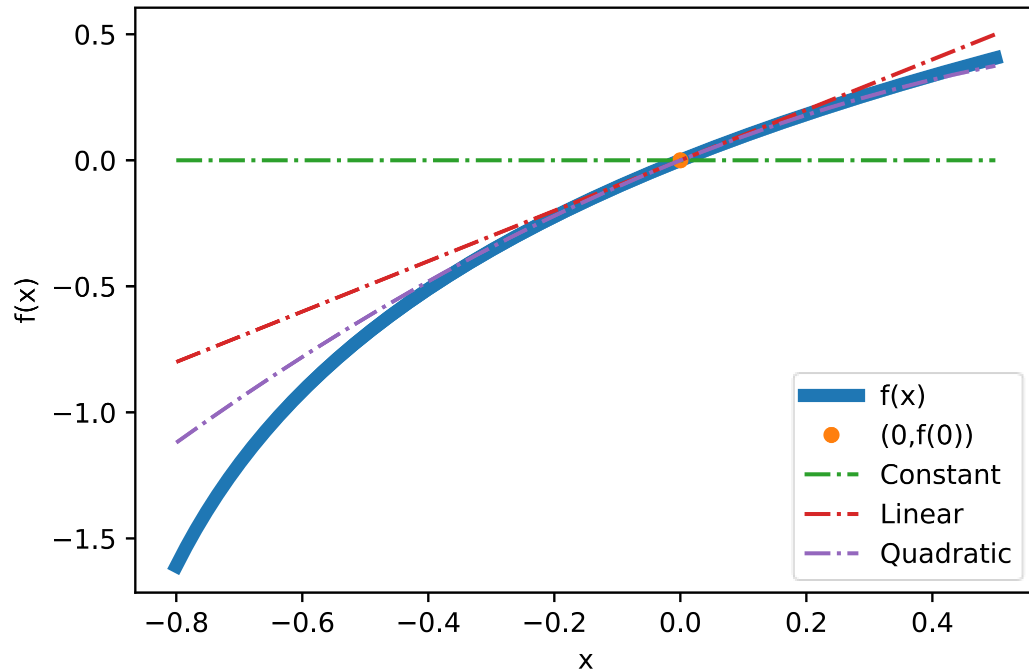
- Best quadratic?

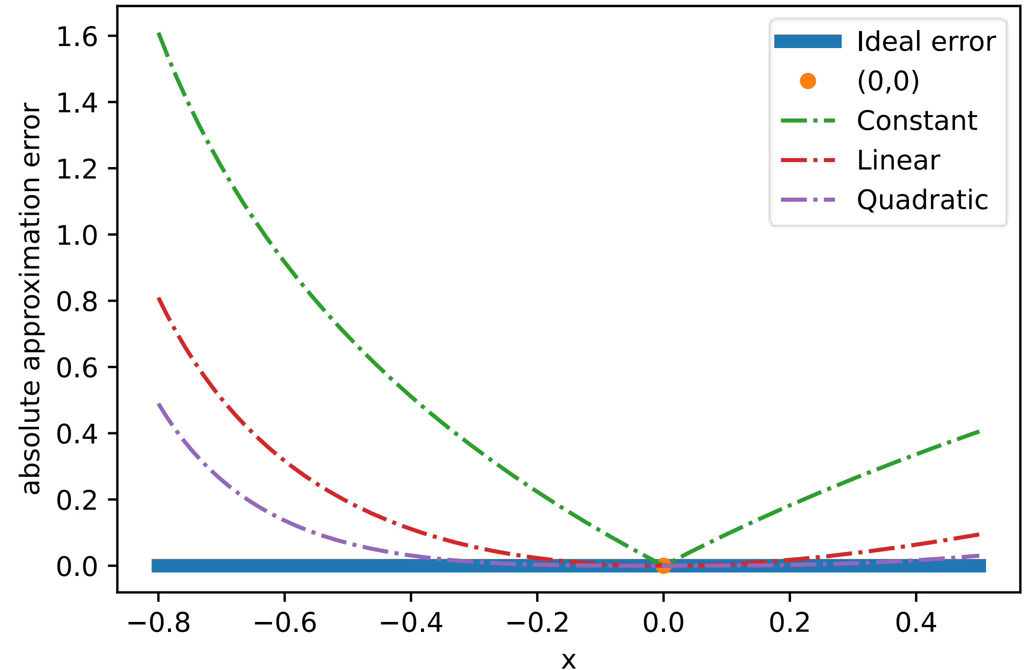
$$\mathrm{P}_2(x) = f(0) + f'(0)x + f''(0)x^2$$

- And so on; this process is called truncating the Taylor series expansion, is based on matching the orders of derivatives between the function and its approximation, and the results are the Taylor polynomials
  - The error has an explicit formula in terms of integrals, and with some ingenuity you can often estimate a usable upper bound on the error
- For the logarithm, the derivatives require just arithmetic so no issue in evaluating the coefficients by hand (or CAS), then a computer algorithm would use these hardcoded coefficients

Minimizing Local Error – Polynomials

As you go higher in order, the approximation gets better in a neighborhood of x=0. For well-behaved functions f, you can show that for every nonzero positive error upper bound, there exists a polynomial order large enough to guarantee the error is within that bound on the desired domain. That's the idea of "convergence". For computer evaluation, you can take the error bound to be on the order of rounding error or otherwise acceptable system error.





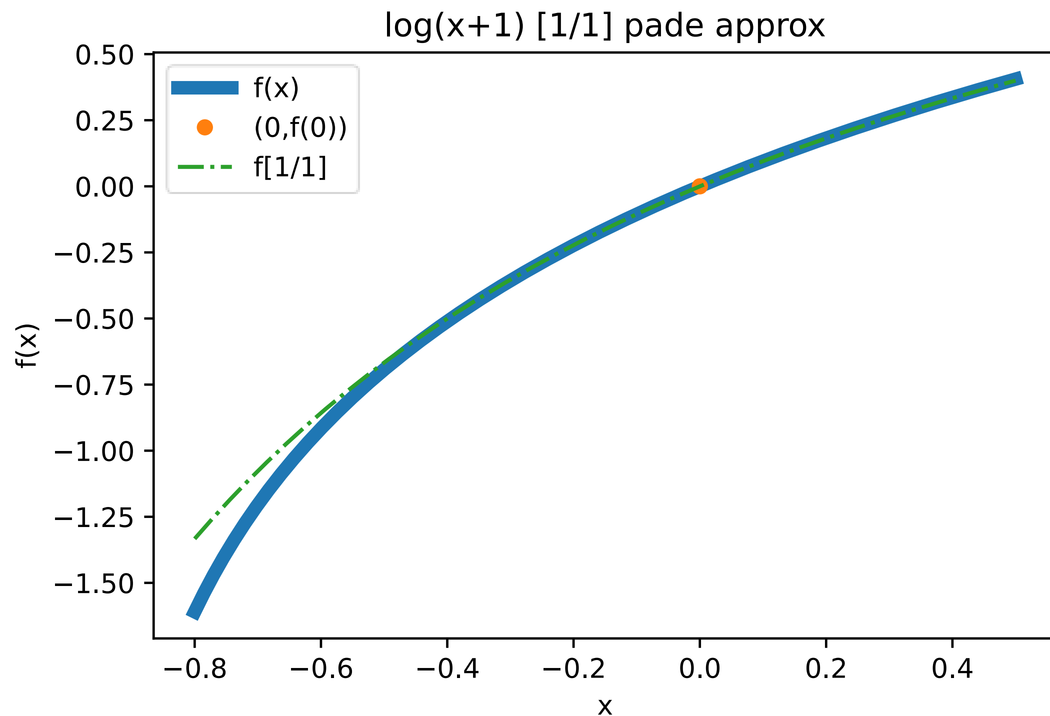
Minimizing Local Error – Rational Functions

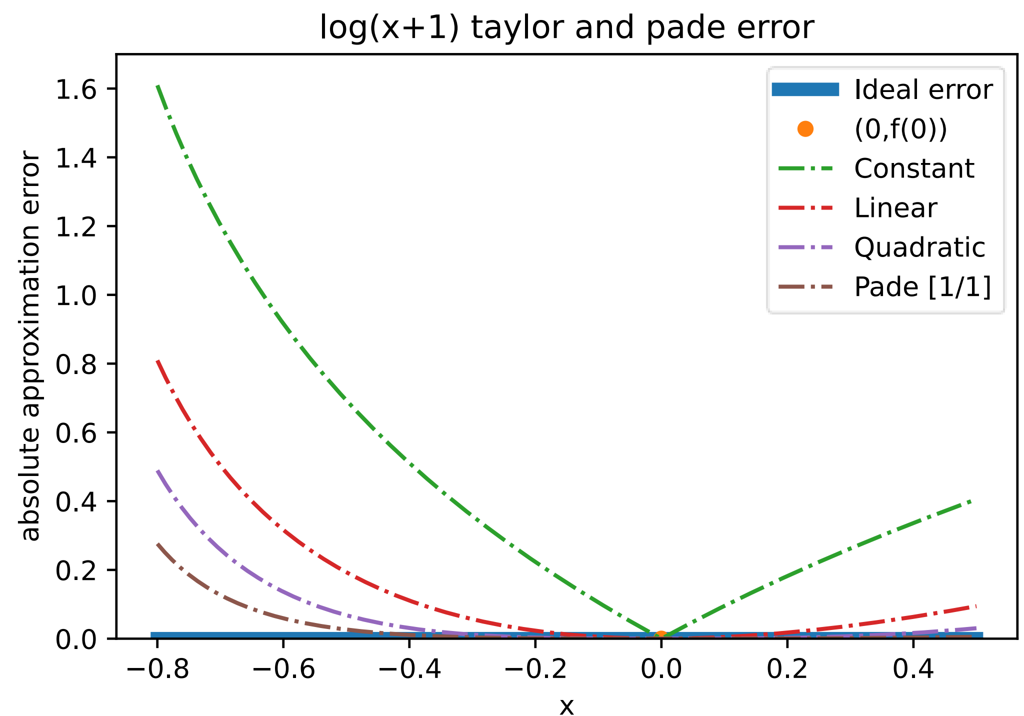
- Derivative matching at a single point works for rational functions too
- That's called the Padé approximation technique; example:
- $R_{[1/1]}(x) = R(x) = \frac{a_0 + a_1 x}{1 + b_1 x}$
- For $f(x) = \log(x + 1)$ around $x = 0$ we get

$$R(x) = \frac{x}{1 + \frac{1}{2}x}$$

Minimizing Local Error – Rational Functions

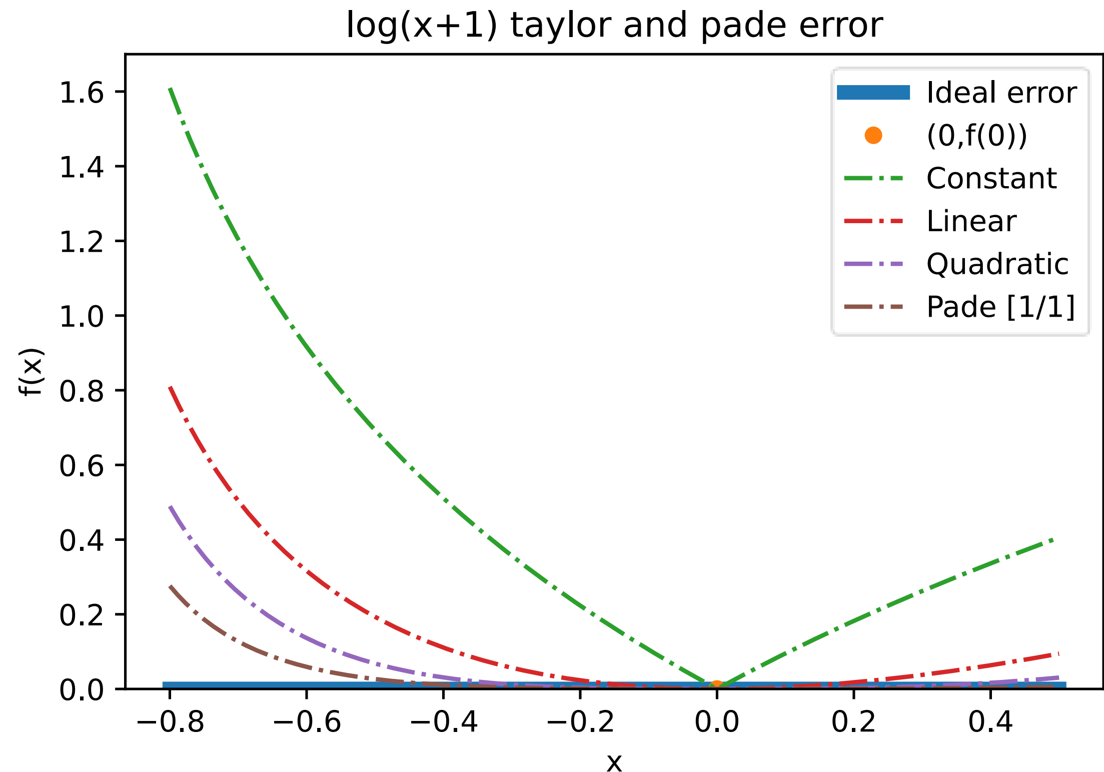
Convergence story is even better for Padé. For a given number of free parameters, a Padé approximant will usually achieve smaller error on a larger domain. For example, here, [1/1]-Padé and quadratic Taylor both have three coefficients, but Padé gets a lower error across the whole domain of interest.





Global or Interval View of Error

- Taylor and Padé are constructed to be good at one point and kind of naturally are good at nearby points by smoothness
- One point is chosen to be special
- Thinking of approximations that are equally good at several points on a domain leads to interpolation
- Approximations that are equally bad everywhere leads to minimax approximants



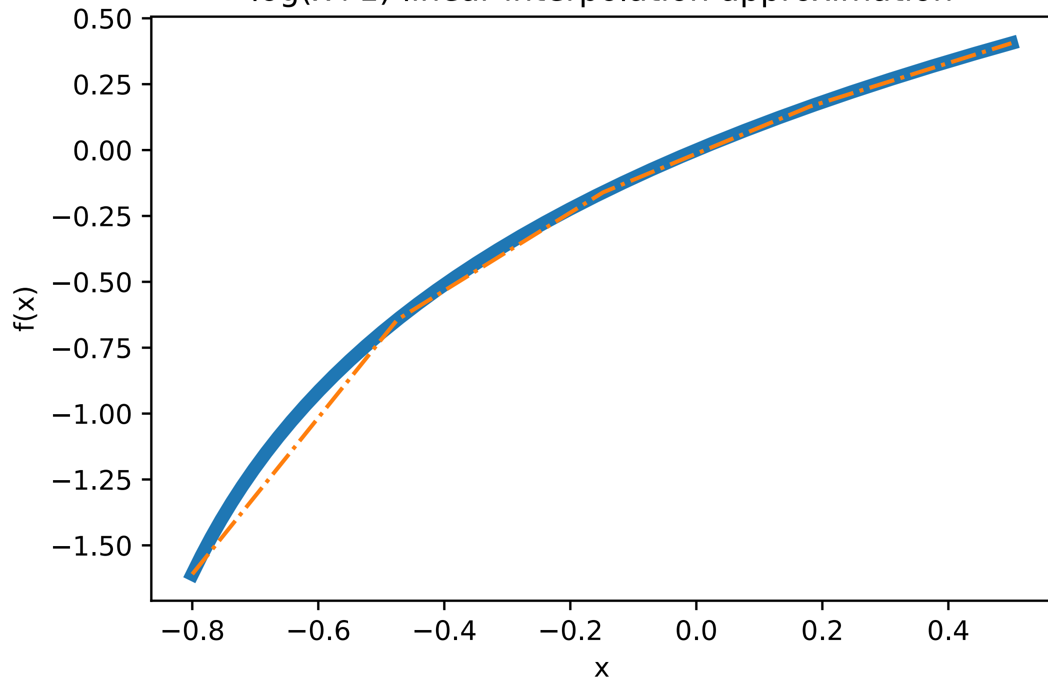
Interpolation

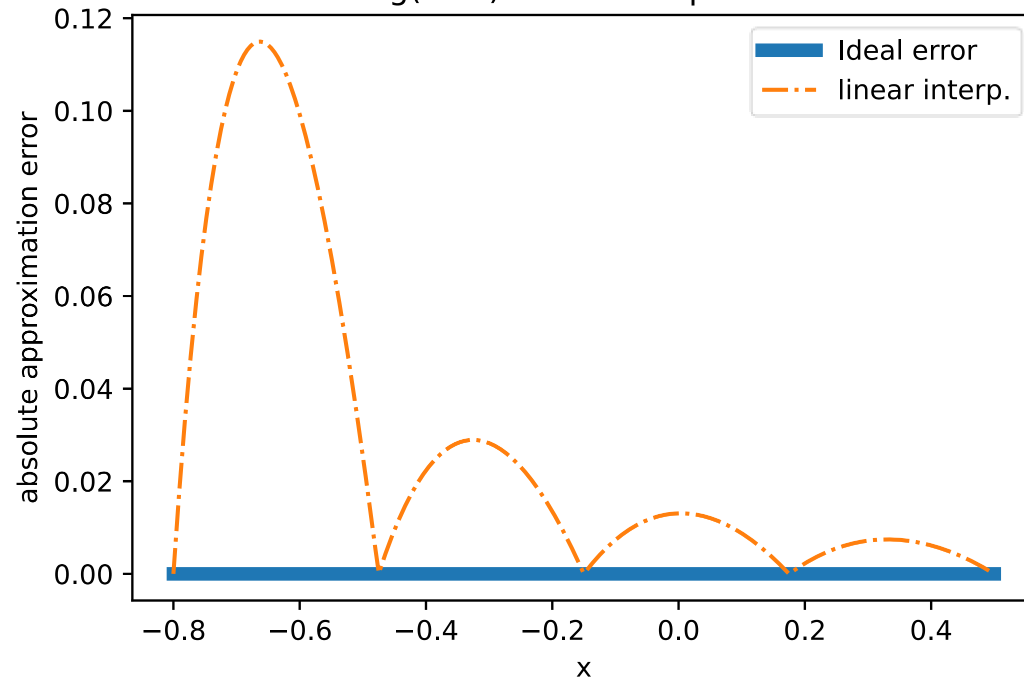
Say I want to pin the error to be zero at several points, so I use a piecewise function that goes between those points

The error isn't zero at one point, its zero at several points!

This is *domain decomposition* + linear polynomial fit on each interval

And with 4 intervals, each with 2 coefficients, we get the best fit yet (previous Padé was e(x)<0.3 on the whole interval)





Polynomial Interpolation

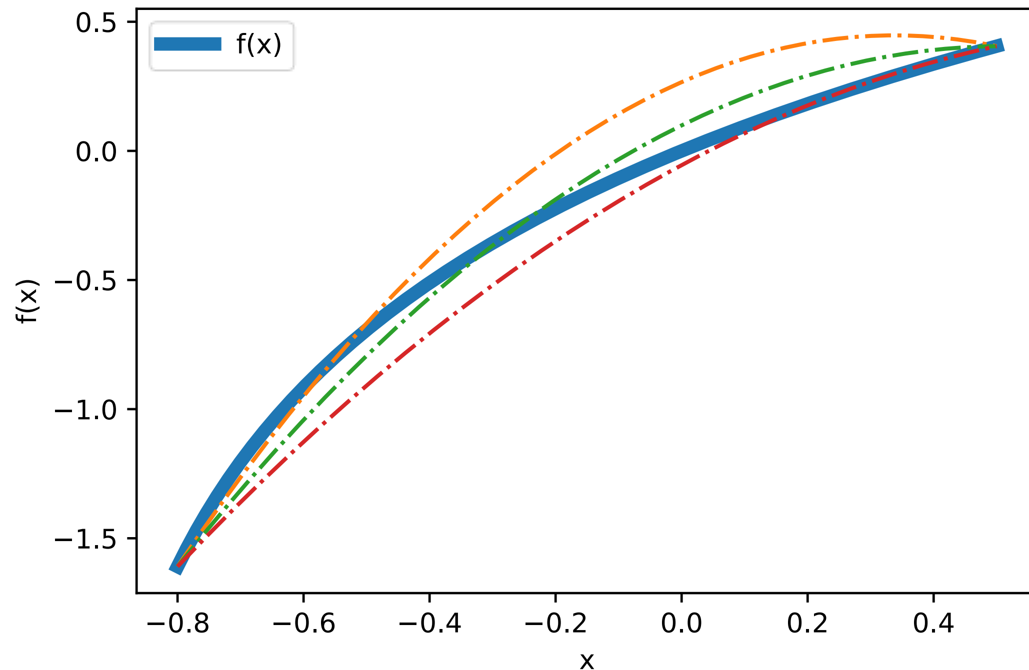
Polynomial interpolation: matches function values at several points on the domain instead of derivatives at one point (Taylor).

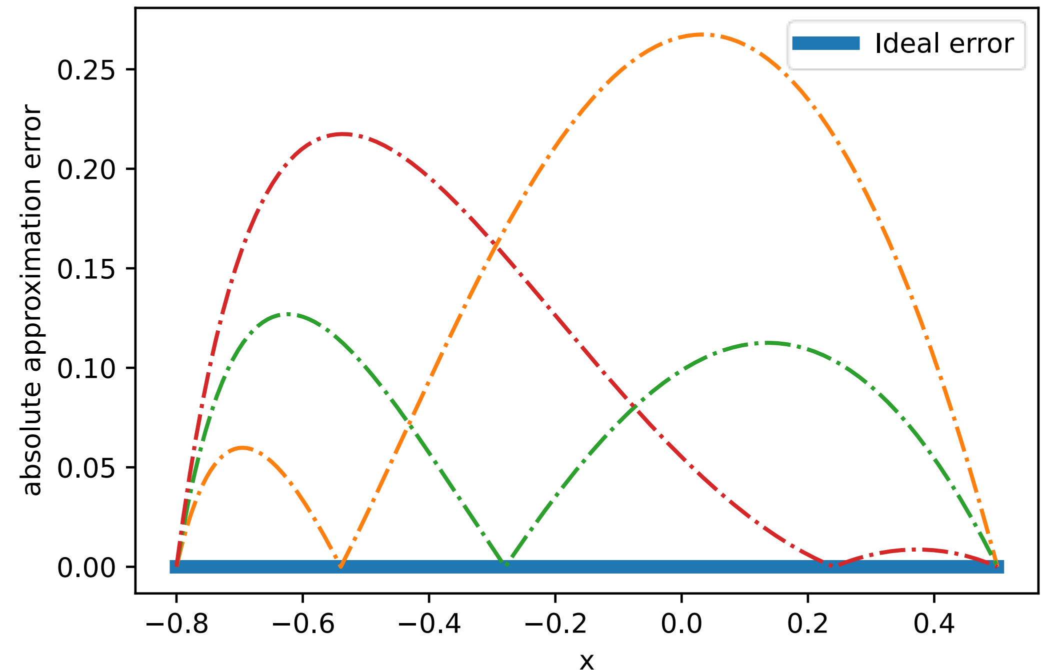
Quadratic example: endpoints and a point in the middle can have exactly zero error. The green error is interesting…

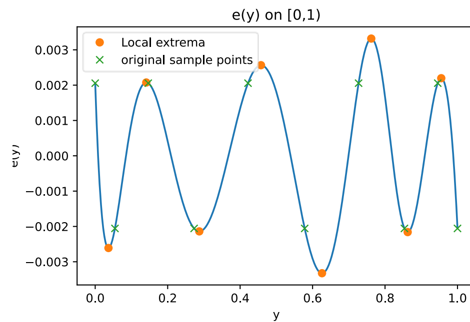
It gets lowest error out of the three, but can I do better?

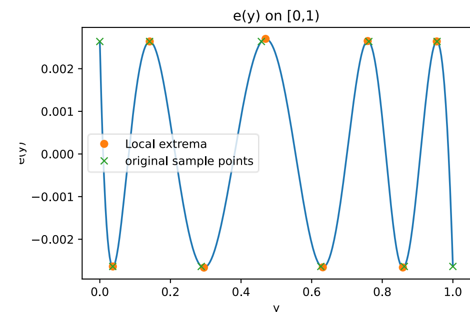
You can distribute the errors more fairly such that the error equioscillates! That is the *minimax* polynomial, minimizes the maximum error.

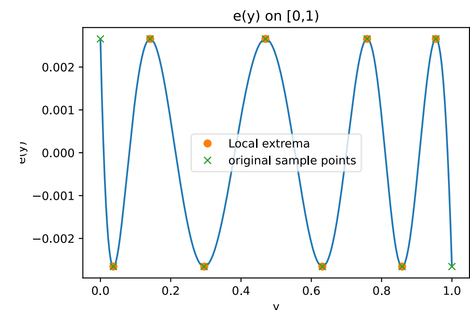
Finding it is an optimization problem: find the set of interpolating points that minimizes the maximal error / makes the error equioscillate











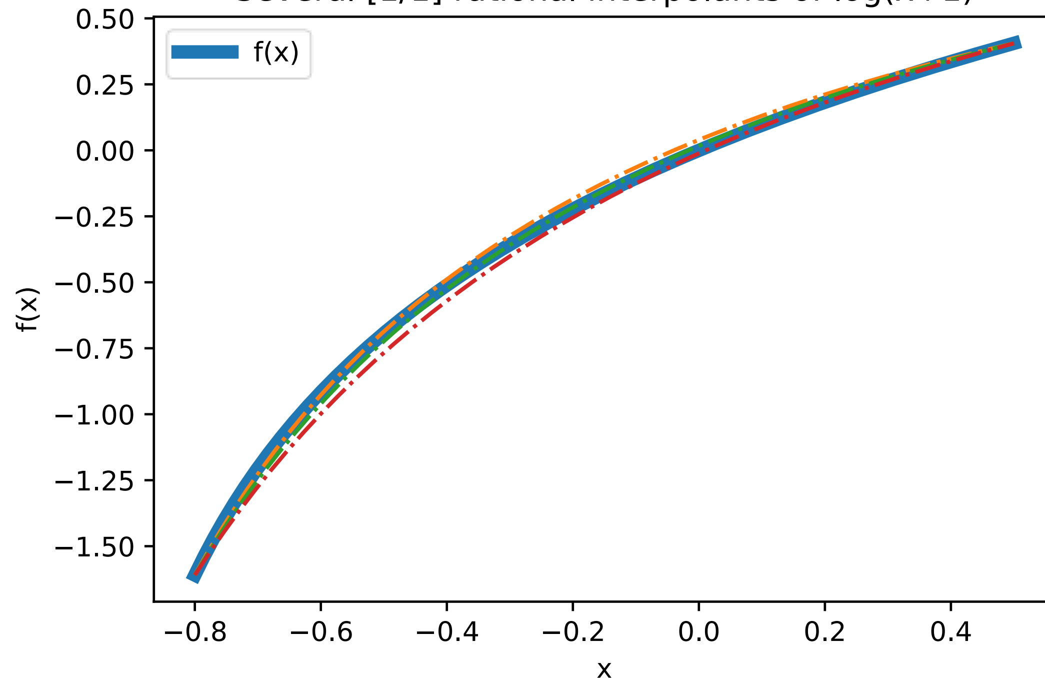
Note on the Remez Exchange

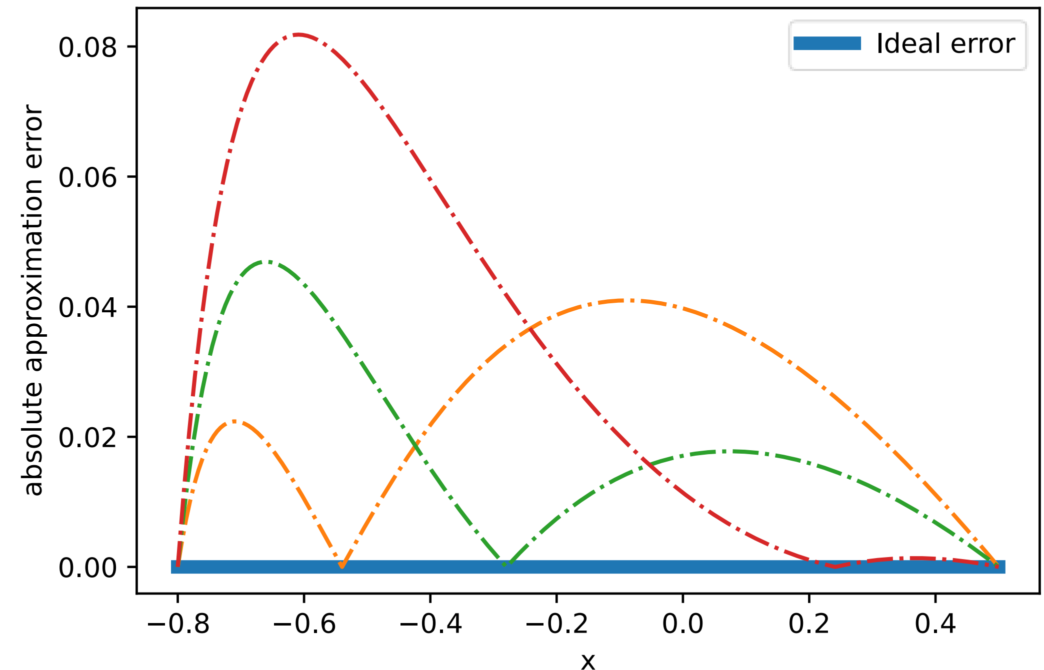
- The most famous method to calculate minimax polynomial approximations is known as the Remez exchange
- It is an iterative algorithm that solves the optimization problem of finding the interpolation points that give you the minimax polynomial
- Pick some points, do a linear system solve to find a polynomial fit that tries to make those the points of maximal error, find the actual points of maximal error with a rootfinding algorithm, replace the points, repeat
- DSP side note: Jim McClellan (my intro DSP prof. at GT!) figured out that if you apply the idea of fitting a polynomial that equioscillates around target values to the frequency response of a filter, you can inverse transform the polynomials to FIR taps. That makes a filter with a desired frequency response with a controllable error around that desired response. Want less error, use higher order fit / more taps. He applied the Remez exchange to solve for the best points in frequency to interpolate between. McClellan’s advisor at Rice was Thomas Parks. Remez exchange applied to FIR filter design = Parks-McClellan filter design algorithm.

Rational Function Interpolation / Minimax

Interpolation idea holds for rational functions too; use DOF to get zero error on a set of interpolating points instead of matching derivatives at a single point (Padé). The equations get more complicated, but it is doable. Or you can just use a generic nonlinear least-squares optimizer to solve the interpolation problem (Levenberg-Marquardt is a popular choice).

Similarly, minimax can be done for rational interpolants by more layers of iteration / optimizer. You can imagine sweeping the middle interpolation point below between -0.55 and -0.25 to find the minimax [1/1]-rational fit





Recap – For Same Number of DOF…

- (And a boatload of caveats on the function well-behavedness)
- Rational fit gets less error than a polynomial
    - Requires division, could be a dealbreaker depending on hardware
- Interpolants get less error than derivative matching at a single point (Taylor and Padé)
    - The only reason to use Taylor or Padé is if what your doing is highly sensitive to derivatives/gradients, not just the function values
- Minimax gets the least error among all interpolants
- So: minimax rational interpolant is the most accuracy per DOF you can get
    - Most complicated to find the coefficients
    - Algorithm idea: 4 layers of optimization! One outermost optimizer to move the interpolation points (the exchange in Remez exchange), one optimizer to find the coefficients that fit the interpolation points (e.g., Levenberg Marquardt), one to find the equioscillation error value (fixed point iteration), and an innermost optimizer to find the extrema of the error (rootfinding)!

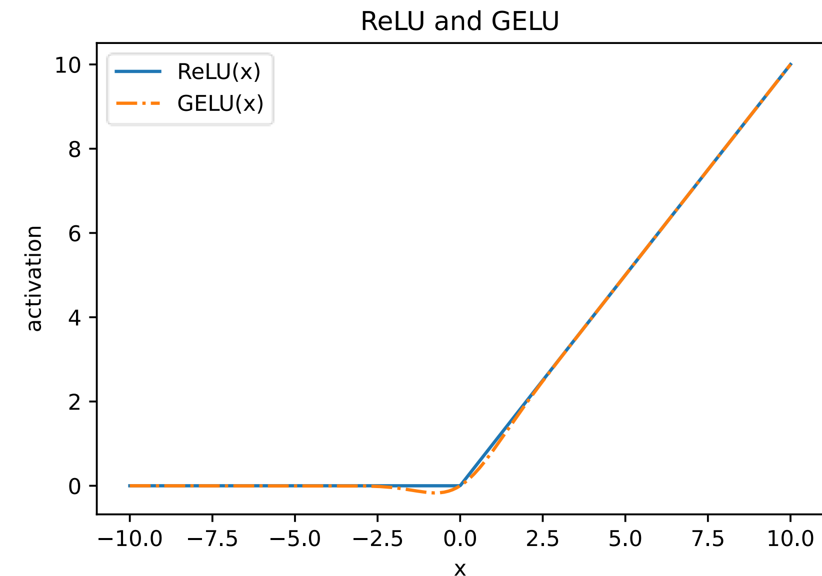
Bag of Tricks

- Almost every numerical code for "special functions" uses a mix of the following
  - What function to model?
  - Symmetry
  - Domain mapping
  - Pulling out asymptotics
  - Domain decomposition
- Let's do a case study that is relevant to recent ML models:

$$f(x) = \mathrm{GELU}(x)$$

What’s a GELU (“Gaussian error linear unit”)

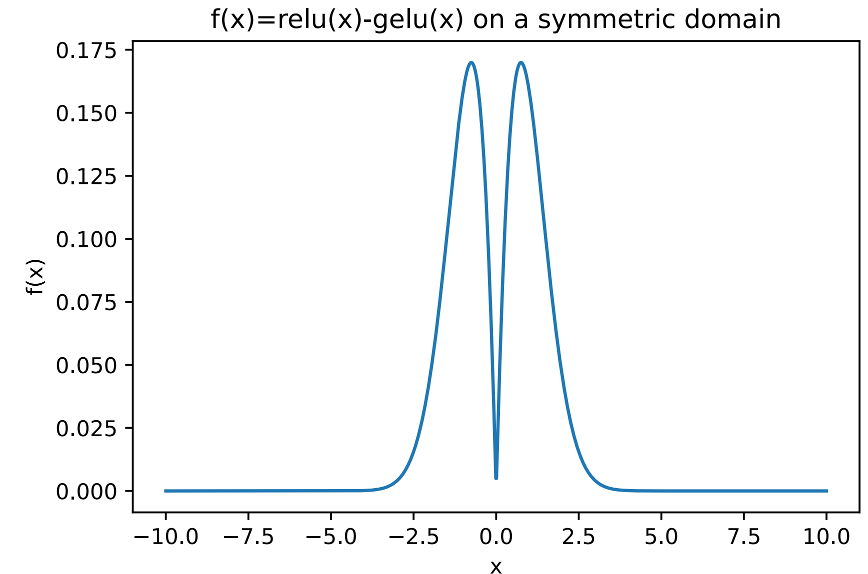
- Like ReLU but smooth everywhere
    - Looks like ReLU far away from x=0
- Has negative outputs near zero for negative inputs near zero
- Wait isn’t that swish?
    - They both have the same structure
        - $f(x) = x \cdot s(x)$
    - But with different sigmoidal functions (generically meaning S-shaped curves)
    - One uses the logistic function, the other uses the Gaussian error function
- $f(x) = \mathrm{GELU}(x) = x \cdot \Phi(x)$
- $\Phi(x) = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)$
- $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt$



Numerical approximation of integrals is another talk
Its surprisingly interrelated with the methods we’re talking about
Slow but accurate algorithms for that: sum up the area under the curve + theoretical error bounds; or use the Taylor series + remainder bounds (derivatives are in terms of exponentials only)
Interpolate between the the slow+accurate points for a fast method

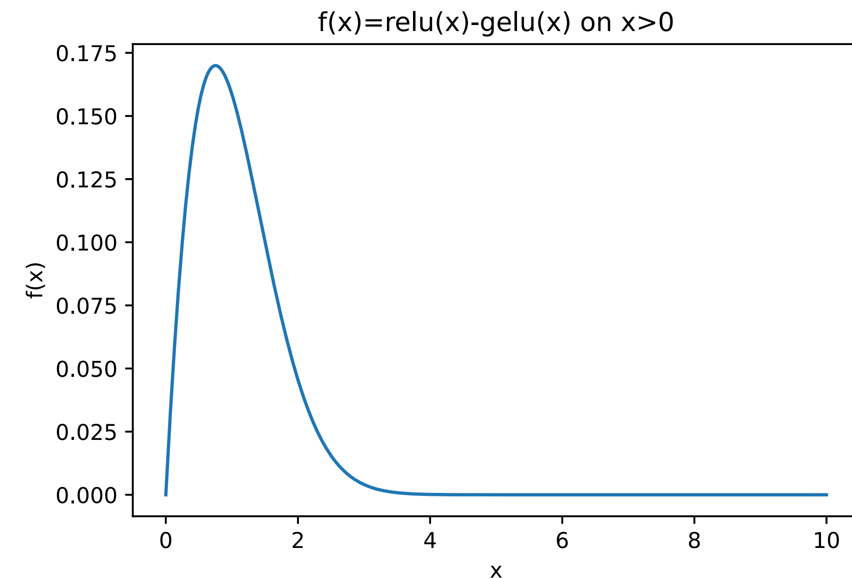
Bag of Tricks – What Function to Model?

- ReLU looks an awful lot like GELU
- It sure looks like ReLU *is* GELU, plus a little bit
- $f(x) = \mathrm{ReLU}(x) - \mathrm{GELU}(x)$
- We can approximate f, and when it comes time to evaluate, we do $\mathrm{GELU}(x) = \mathrm{ReLU}(x) - f(x)$
- ReLU is easy to evaluate, and subtraction is cheap



Bag of Tricks – Symmetry

- That function appears to have even symmetry, $f(x) = f(-x)$
  - Can be proved
- So, we shouldn't fit a polynomial to all of it, we should just fit for x>0 and return $f(|x|)$
- Another symmetry trick: only angles in $[0, \frac{\pi}{2}]$ are needed for trig functions, the rest are reflections and shifts

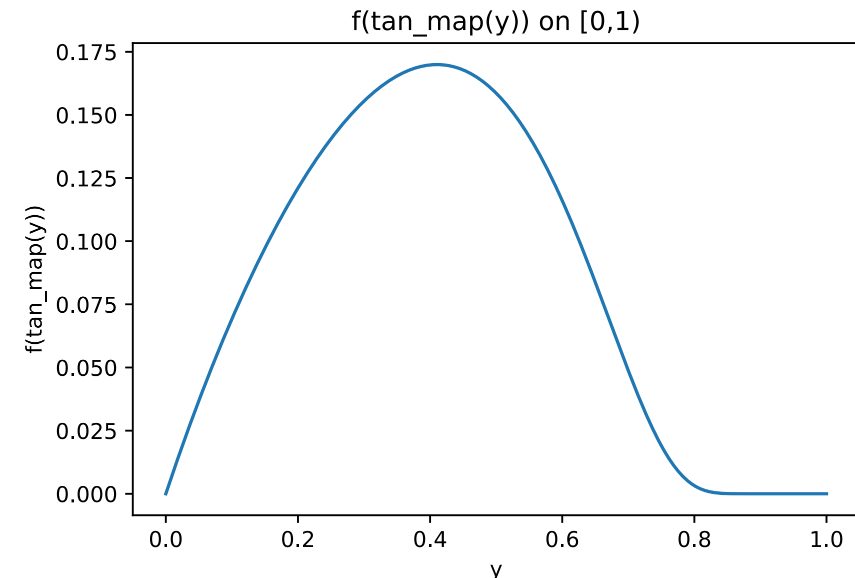


Bag of Tricks – Domain Mapping

- That's a long flat tail decaying to 0 at ∞
- What if I took $[0, \infty)$ and squashed it down to a finite interval, say $[0,1)$?
- Domain mapping (change of variable):

$$g(y) = f\left(\tan\left(\frac{y\pi}{2}\right)\right)$$

- We've used the singularity in the tan to make sure that as y approaches 1, the argument of f goes to infinity
  - That maps the semi-infinite interval into a finite one
- Generally, there's lots of choices and it affects convergence rates. Boyd's "Chebyshev and Fourier Spectral Methods" is the best reference I know of. I picked tan because it worked for my experiments.



Bag of Tricks – Pulling Out Asymptotics

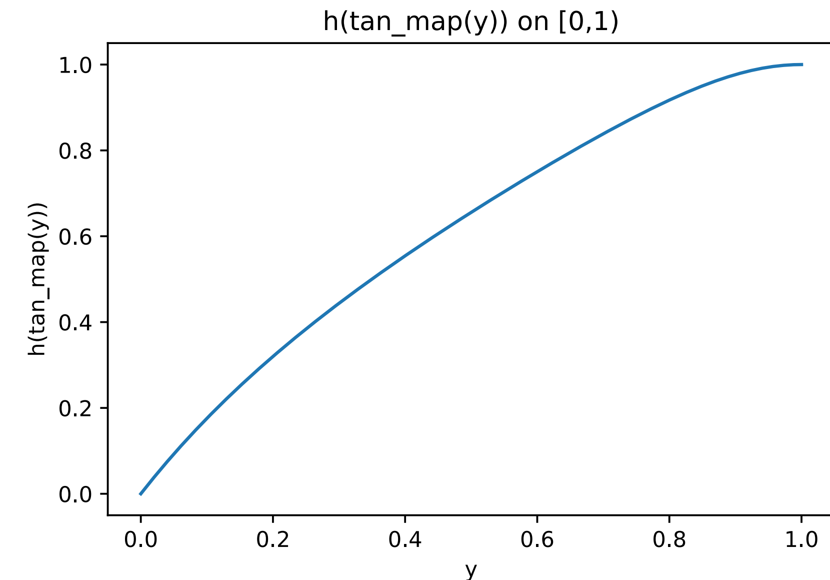
- That function is VERY flat near 1, too flat for a polynomial fit to be very good. That was originally the tail at infinity.
- OTOH, you can show that

$$f(x) \approx \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}, x \gg 0$$

- That's based on asymptotic expansions, a talk for another day
- What if I factor out that behavior and say

$$f(x) = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} h(x)$$

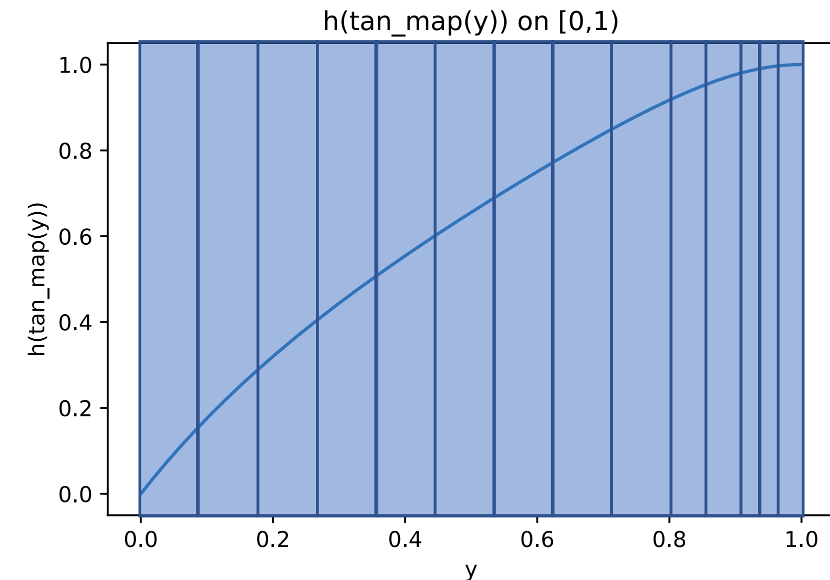
- And then model $h(x)$ (with the same domain mapping trick)



Lots of details skipped about how to evaluate h since it looks like you'd have exponential blow up if you naively multiplied, but this all simplifies to a well-known scaled complementary error function, erfcx and I used scipy implementation of that to make the plots.
Of course, that implementation uses the same tricks we're talking about!

Bag of Tricks – Domain Decomposition

- Instead of modeling the whole function on [0,1), we can chop up the domain into smaller intervals (like we did with linear interpolation) and fit lower order polynomials or rational functions to each piece
- Adaptive algo:
    - Visit each domain and fit as best you can with the order you want
    - If the error on a domain is not tolerable, split that domain in half and revisit the halves as in the previous step

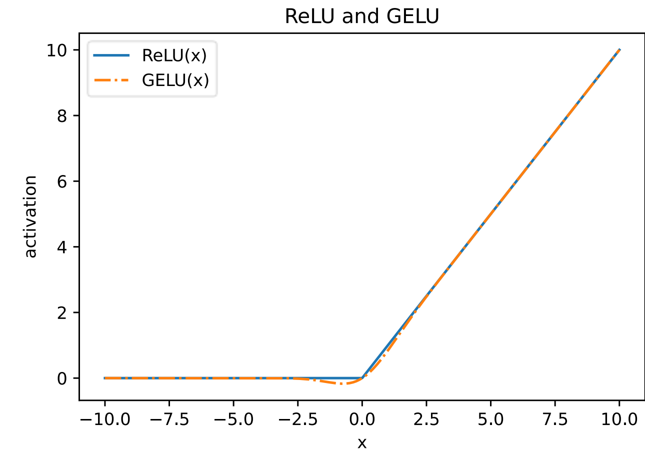


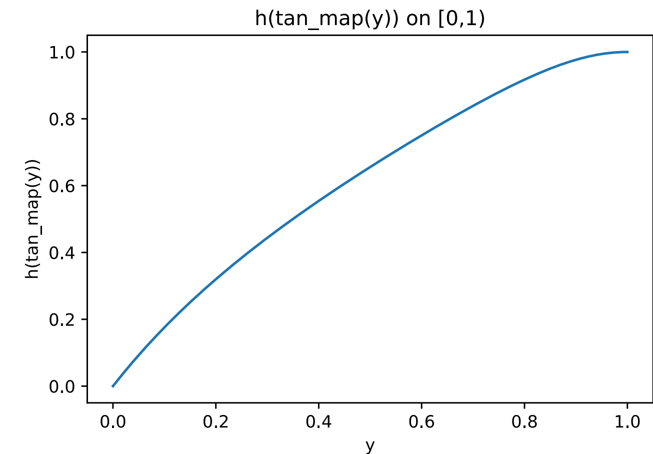
Bag of Tricks – Recap

- Bag of tricks took us from the top plot to the bottom plot; the thing on the bottom requires far fewer DOF for a desired accuracy
- And its effectively valid on the whole real line
- Evaluation would look like:

$$\mathrm{GELU}(x) = \mathrm{ReLU}(x) - \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} h\left(\left|\frac{2\,\mathrm{atan}(x)}{\pi}\right|\right)$$

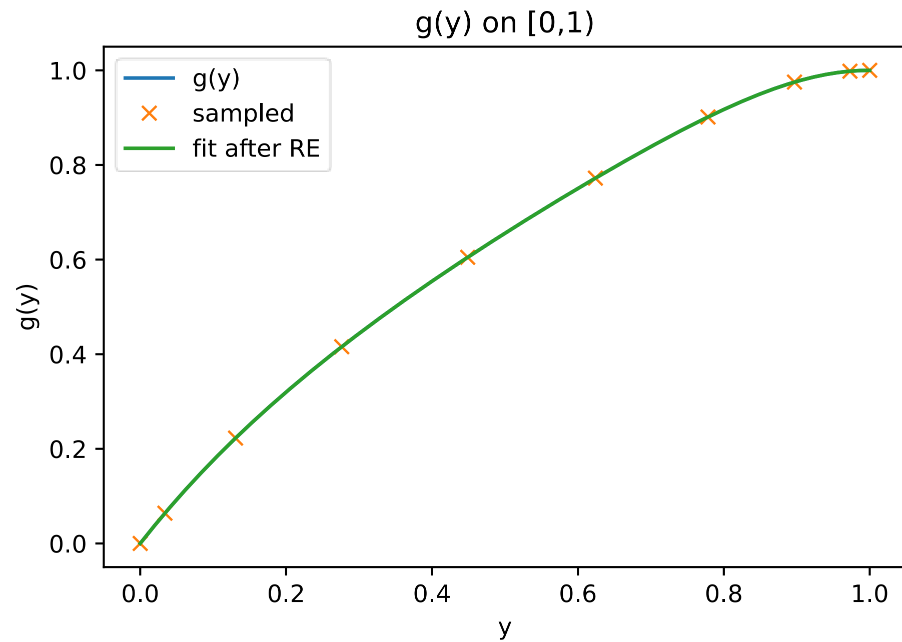
- h is a low order interpolating polynomial or rational, or a domain-decomposed sequence of those things on the interval [0,1)
- If ReLU, subtraction, gaussian evaluation, abs, atan are cheap, this will go fast, and you only used your DOF to model the critical piece

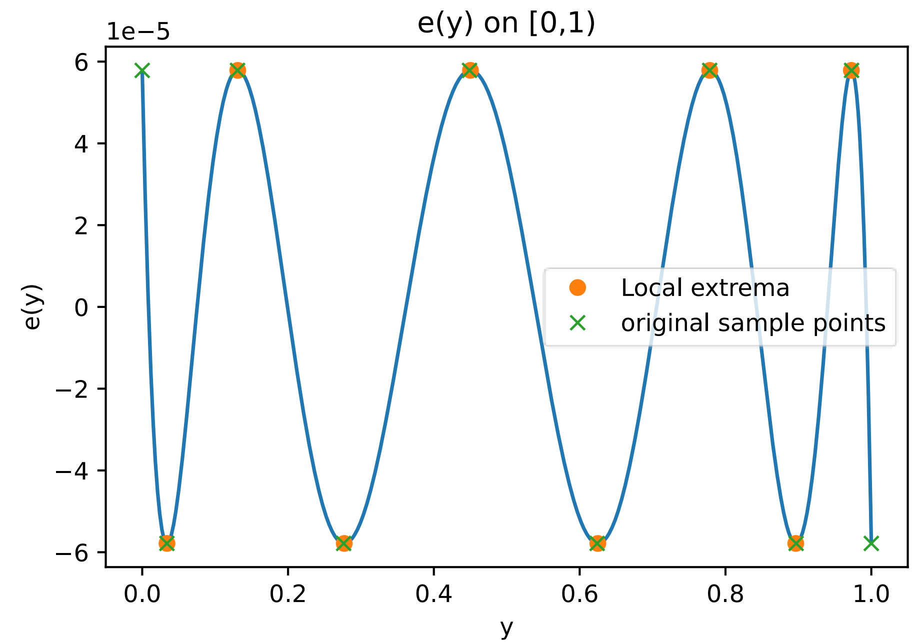




GELU Results

Minimax polynomial fit of order 8 fits GELU with error<6e-5 over the entire real line. Didn't use domain decomposition trick.





Summary / Conclusions

- Polynomial and rational approximants are ways to approximate functions
    - Taylor and Padé approximants match derivatives at a single point
    - Interpolants have exactly zero error at the interpolation points and oscillate in between
- Minimax functions (both polynomials and rational) are improvements of the interpolant idea that find the best set of points to interpolate through to minimize the maximum error
    - This is how you get the most accuracy per DOF, but if you can tolerate more DOF, interpolants on a pre-determined grid (i.e., Chebyshev nodes) perform well enough
- Rational functions get better error per DOF than polynomials
    - But are harder to construct and require division, not just *+
- You can get less error per DOF by carefully selecting what function to approximate, i.e., really separate out the easy to compute parts from the harder to compute parts.
    - In our example, subtracting the ReLU and factoring out the asymptotic really makes a huge difference
- Using symmetry makes sure you're only modeling the portion of the function that can't be described by reflections and shifts of other parts
- Domain mapping can help you get an approximation that's good over unbounded intervals
    - Works for monotonic and bounded functions that converge reasonably quickly
    - Notably doesn't work well for functions that wiggle at infinity like sinc or Bessel J functions because infinite number of wiggles get mapped to finite domain
- Domain decomposition guarantees you can use low order approximants on each interval
    - At the cost of look-up table type calculations to decide which sub-domain your input falls into