

Visualizing Optimizers using Chebyshev Proxies and Fatou Sets

R. Winchenbach¹  and N. Thuerey¹ 

¹TU Munich, Germany

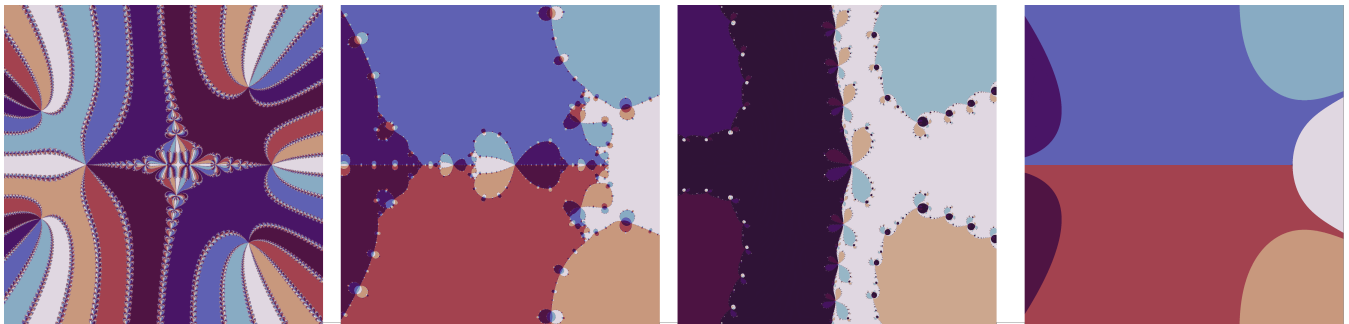


Figure 1: Images visualizing the different convergence behaviors of Newton's method (leftmost), Halley's method (center-left), Newton's method as an optimizer (center-right) and Adam (rightmost) with point of convergence color coded based on polar angle. Note that in this case the Hessian matrix is not always positive definite so Newton's method as an optimizer yields maxima for most points in the complex domain. Using our toolkit it is possible to visualize the differences in convergence for different optimization approaches for polynomials of very high degrees and can be used as a tool to better visualize how these methods behave for different functions and hyperparameters.

Abstract

With recent advances in optimization many different optimization approaches have been proposed, especially regarding the optimization of weights for neural networks. However, comparing these approaches in a visually succinct and intuitive manner is difficult to do, especially without relying on simplified toy examples that may not be representative. In this paper, we present a visualization toolkit using a modified variant of Fatou sets of functions in the complex domain to directly visualize the convergence behavior of an optimizer across a large range of input values. Furthermore, we propose an approach of generating test functions based on polynomial Chebyshev proxies, with polynomial degrees up to 11217, and a modification of these proxies to yield functions that are strictly positive with known global minima, i.e., roots. Our proposed toolkit is provided as a cross platform open source framework in C++ using OpenMP for parallelization. Finally, for monomorphic functions the process generates visually interesting fractals, which might also be interesting from an artistic standpoint.

CCS Concepts

• **Mathematics of computing** → Computations on polynomials; • **Human-centered computing** → Scientific visualization;

1. Introduction

Optimization approaches have long been an important aspect in many fields of research, including root finding and in finding local and global minima of functions. With the rise of machine learning and neural networks there has been a growing interest in the optimization of parameters, especially for high dimensional problems that commonly occur during the training of neural networks. However, these large scale problems often make it difficult to visualize how different optimization approaches differ regarding convergence, except by choosing toy examples that only show the op-

timization trajectory for single initial guesses for a simple function that can readily be chosen to impose a bias on the evaluation and these are generally not standardized across methods. Furthermore, mathematical derivations regarding the convergence behavior of different optimizers are very useful and important, but they do not provide an intuitive and comparable understanding of the convergence behavior to non experts. This problem is only exacerbated when different hyperparameters of an optimization approach can significantly change the convergence behavior in a variety of manners that are important to visualize.

In this paper, we now propose a visualization toolkit that can directly visualize the convergence behavior of a variety of optimization approaches by generating a visual representation of the basins of convergence using a Fatou set like approach. In this regard, we propose an approach of generating arbitrarily difficult test functions by approximating a real valued scalar function using Chebyshev proxy functions, using polynomials of up to degree 11217, and expanding this proxy function to a complex valued domain. This extension provides a holomorphic basis for our evaluations, where we then provide a brief derivation of the Jacobian and Hessian matrices of the Chebyshev proxy using a modified variant of Clenshaw's algorithm. Using these analytical derivatives, we can then evaluate root searching methods directly in the complex plane, whereas for optimization approaches we propose a simple modification of the Chebyshev proxy that ensures the function being positive everywhere, except for at the roots of the proxy function, which serve as global minima of the function. We then provide open source framework using C++ and OpenMP for this process and use this framework to compare a variety of optimization approaches and hyperparameters.

2. Chebyshev Proxy Functions

In optimization theory many different test functions exist to evaluate the properties of a given optimizer, e.g., the Rosenbrock function [Ros60], which are often difficult to optimize yet provide analytic information about any order derivative of the function. However, for a more intuitive understanding of the optimization process it would be beneficial to utilize an easily modifiable set of test functions that can be generated on-the-fly based on the requirements of a specific user. Using such black-box functions directly, however, is difficult as they generally do not provide explicit information about the functions' derivatives and properties and relying on automatic differentiation or finite difference schemes may not always be readily possible or numerically sufficient. Within our framework we utilize such functions indirectly by first approximating them using so called Chebyshev proxies [Boy13], which are a common tool in numerical analysis [DHT14].

These proxy functions work under the assumption that in \mathbb{R} the Weierstrass Approximation Theorem [Wei85] guarantees the existence of an approximating polynomial function f_p , with a polynomial degree n , for every C^0 continuous function f over a compact interval $[a, b] \subset \mathbb{R}$ up to an arbitrarily small error ε , i.e., $\|f - f_p\|_\infty \leq \varepsilon$. However, using equidistantly placed sampling points for the construction of such a proxy can yield increasing errors with increasing degrees, e.g., for Runge's function [Run01]. Chebyshev interpolation, on the other hand, places sampling points at the roots of a Chebyshev basis polynomial (the so-called Chebyshev-Lobato grid), that avoids such artifacts [Boy95]. This process furthermore guarantees uniform geometric convergence for analytic functions with an approximation error $\mathcal{O}(e^{-n\mu})$ [Ber18], with n being the polynomial degree and μ being a positive scaling coefficient, whereas non analytic functions tend to converge much slower [AT17]. The basis of these proxy functions are generally Chebyshev basis polynomials of second kind T_n [DHT14] of degree n , which are defined through the recursive relation [Boy14]

$$T_0(x) = 1; T_1(x) = 2x; T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad (1)$$

which are utilized to construct a Chebyshev polynomial sequence $p(x) = \sum_{i=0}^n c_i T_i(x)$, where the set of coefficients $C = \{c_0, \dots, c_n\}$ are called the Chebyshev coefficients. The coefficients C are calculated by sampling f on a Chebyshev-Lobato grid of size $n+1$ and performing a discrete Fourier transform (DFT) of these function evaluations, which yields a uniquely defined set of Chebyshev coefficients [Boy01]. While the Weierstrass Approximation Theorem guarantees the existence of a polynomial with a given approximation error bound, there is no universal approach to find an optimal approximating degree. Many different heuristic approaches, e.g., [CC60; FW66; SS80; Jac90; BT04; MH02; Boy95; Boy02; Mav94; AT17], exist to determine an estimated optimal polynomial degree n_ε for a given error bound ε using alternative error metrics, such as the round-off plateau behavior of the coefficients of the polynomial sequence, which we utilize it is also used in the widely utilized Chebfun library [DHT14]. Consequently, a uniquely determined proxy function f_p exists of degree n_ε as

$$f_p = \sum_{i=0}^{n_\varepsilon} c_i T_i, \quad \text{with } \|f - f_p\|_\infty \leq \varepsilon. \quad (2)$$

At this point is also important to note that while this approximation process is explained here for real-valued scalar functions, it can also be performed for two-dimensional [TT13] and three-dimensional [HT17] functions, as well as complex-valued functions over the complex-plane [Boy14].

3. Clenshaw's Algorithm and Derivatives

After approximating a given black-box function according to Sec. 2, we now need to evaluate the proxy function and the first and second order derivatives. Additionally, as we are interested in a complex valued function over the complex-plane, we need to perform this evaluation using imaginary numbers. As polynomial functions are holomorphic, we can replace the evaluation of the proxy function f_p at the complex coordinates $z = x + iy$, with $z \in \mathbb{C}$ and $x, y \in \mathbb{R}$, with an evaluation of two functions of two real variables $u(x, y)$ and $v(x, y)$, with $f_p(z) = u(x, y) + iv(x, y)$. Evaluating a Chebyshev polynomial sequence can be done efficiently using Clenshaw's Algorithm [Cle55], which evaluates a given Chebyshev polynomial through the recursive evaluation of

$$b_k(z) = a_k + s_k z b_{k+1}(z) - b_{k+2}(z), \quad s_k = \begin{cases} 1, & k=0, \\ 2, & \text{else.} \end{cases} \quad (3)$$

While this equation can be evaluated directly using $z \in \mathbb{C}$, we will later on require the separate evaluation of u and v , as defined above, and accordingly splitting Clenshaw's algorithm to explicitly evaluate the real and imaginary components is useful. The first thing to consider here is that the coefficients c are only real valued based on the process described in Sec. 2, as we are only aiming to approximate the function on the real numbers. Accordingly, the real valued part b_k^u and the complex valued part b_k^v can be refactored as

$$b_k^u = a_k - b_{k+2}^u + s_k [x b_{k+1}^u - y b_{k+1}^v], \quad (4)$$

$$b_k^v = -b_{k+2}^v + s_k [x b_{k+1}^v + y b_{k+1}^u]. \quad (5)$$

Note that we left out the (x, y) for brevity as all b terms depend on x, y equally. Furthermore, we are interested in the derivatives of the proxy function with respect to the two components x and y , i.e.,

we want to evaluate $\partial/\partial x b_k^u, \partial/\partial y b_k^u, \partial/\partial x b_k^v$ and $\partial/\partial y b_k^v$. As the proxy function is holomorphic we know that, due to the Cauchy-Riemann equations, $\partial/\partial x b_k^u = \partial/\partial y b_k^v$ and $\partial/\partial y b_k^u = -\partial/\partial x b_k^v$, finding all required derivative terms only requires the evaluation of $\partial/\partial x b_k^u$ and $\partial/\partial y b_k^u$. These terms can be found simply by applying the partial derivative operator to equation 4 as:

$$\begin{aligned} \frac{\partial}{\partial x} b_k^u &= -\frac{\partial}{\partial x} b_{k+2}^u + s_k \left[x \frac{\partial}{\partial x} b_{k+1}^u + b_{k+1}^u - y \frac{\partial}{\partial x} b_{k+1}^v \right], \\ \frac{\partial}{\partial y} b_k^u &= -\frac{\partial}{\partial y} b_{k+2}^u + s_k \left[x \frac{\partial}{\partial y} b_{k+1}^u + b_{k+1}^v - y \frac{\partial}{\partial y} b_{k+1}^v \right]. \end{aligned} \quad (6)$$

Finally, for the second order derivatives we only need to explicitly evaluate them for b^u as, due to the Cauchy-Riemann equations applied to the first order derivative, the terms for b^v are analogous. This yields us four required derivative terms:

$$\begin{aligned} \frac{\partial^2 b_k^u}{\partial x^2} &= -\frac{\partial^2 b_{k+2}^u}{\partial x^2} + s \left[x \frac{\partial^2 b_{k+1}^u}{\partial x^2} + 2 \frac{\partial b_{k+1}^u}{\partial x} - y \frac{\partial^2 b_{k+1}^v}{\partial x^2} \right], \\ \frac{\partial^2 b_k^u}{\partial x \partial y} &= -\frac{\partial^2 b_{k+2}^u}{\partial x \partial y} + s \left[x \frac{\partial^2 b_{k+1}^u}{\partial x \partial y} + \frac{\partial b_{k+1}^u}{\partial y} - y \frac{\partial^2 b_{k+1}^v}{\partial x \partial y} - \frac{\partial b_{k+1}^v}{\partial x} \right], \\ \frac{\partial^2 b_k^u}{\partial y \partial x} &= -\frac{\partial^2 b_{k+2}^u}{\partial y \partial x} + s \left[x \frac{\partial^2 b_{k+1}^u}{\partial y \partial x} + \frac{\partial b_{k+1}^u}{\partial x} - y \frac{\partial^2 b_{k+1}^v}{\partial y \partial x} - \frac{\partial b_{k+1}^v}{\partial x} \right], \\ \frac{\partial^2 b_k^u}{\partial y^2} &= -\frac{\partial^2 b_{k+2}^u}{\partial y^2} + s \left[x \frac{\partial^2 b_{k+1}^u}{\partial y^2} - y \frac{\partial^2 b_{k+1}^v}{\partial y^2} - 2 \frac{\partial b_{k+1}^v}{\partial y} \right]. \end{aligned} \quad (7)$$

If we now evaluate the actual proxy function $f_p(z)$ as a complex valued function we can readily find that $f_p(x, y) = b_0^u(x, y) + i b_0^v(x, y)$, $\partial/\partial z f_p(x, y) = \partial/\partial x b_0^u(x, y) + i \partial/\partial y b_0^v(x, y)$ and $\partial^2/\partial z^2 f_p(x, y) = \partial^2/\partial x^2 b_0^u(x, y) + i \partial^2/\partial x \partial y b_0^v(x, y)$. However, the function as constructed thus far is not a well-suited test-case for optimization functions as the function may not have any global minima as it could easily tend to $-\infty$ far from the origin, regardless of how the initial black-box function that was approximated appeared like. Instead, we would like to have a function $\tilde{f}_p(x, y)$ that has global minima and, ideally, a set of known global minima. To achieve this we modify f_p in such a way that it is strictly positive everywhere but retains the original roots of f_p , i.e., the roots of f_p in the complex plane for $z = x + iy$ are the global minima of \tilde{f}_p in the two dimensional plane $x, y \in \mathbb{R}$. Our modification works by directly squaring the real and imaginary components of f_p , i.e., we define

$$\tilde{f}_p(x, y) = \text{Re}(f_p(x, y))^2 + \text{Im}(f_p(x, y))^2. \quad (8)$$

It is important to note that $\tilde{f}_p(x, y)$ is not defined over the complex domain as the component-wise operators, Re and Im, are nowhere complex differentiable. As we previously derived the real and imaginary components of f_p separately, evaluating the gradient $\nabla = \left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right]$ and the Hessian matrix H of f_p can be done straightforwardly using the component wise derivatives. However, there is no guarantee that the Hessian matrix be positive definite, which may be difficult to handle with certain optimization approaches, e.g., Newton's method when used as an optimizer. Finally, it is important to note that all the recursions for all derivative terms can be evaluated in a single iteration over the coefficients, which makes them computationally relatively inexpensive to evaluate.

4. Visualization

A Newton fractal in general describes a boundary set in the complex plane that can be utilized to visualize the convergence of Newton's method, or more precisely, the Newton fractal is the Julia set of a holomorphic function under application of the monomorphic Newton's method. Visualizing the Julia set directly is difficult as the boundary is infinitely thin and, instead, the Fatou set as the complementary of the Julia set is visualized. While initially only applied to Newton's method for root finding, we can readily apply other root finding, or optimization approaches, in the same manner to any polynomial or transcendental function, where, as discussed before, we utilize an underlying Chebyshev proxy function as the basis for the visualization. Such a Fatou set is then the result of evaluating a rectilinear grid over some domain in the complex plane, e.g. $[-1, 1] \times [-i, i]$, based on an iterative evaluation of

$$z_{n+1} = z_n - \Delta z_f(z_n), \quad (9)$$

where for a Newton fractal $\Delta z_f(z_n) = -f(z_n)/\frac{\partial f(z_n)}{\partial z_n}$. An analogous iterative process can be defined over the two dimensional real plane by treating z as a two dimensional value $[x, y]$ instead. This iterative process then can result in three distinct cases: (i) the iterative process converges to a single stable point, i.e., $|z_{n+1} - z_n|_\infty \leq \epsilon$, (ii) the iterative process diverges, i.e., $z_{n+1} = \infty$, or fails, i.e., involves a division by 0, and (iii) yields an astable iteration that does not converge to a single stable point but continues to move in a repeating trajectory. In practice, detecting the first and second case is trivial, whereas detecting the third case is generally not possible as this would implicitly require solving the halting problem, i.e., a trajectory might not have converged after 10^8 iterations, but there is no indication that this trajectory is astable. The only reliable indicator for an astable case would be if the same exact location was visited multiple times with the same state of the optimizer, however, due to the fractal nature of this process, a difference of a single machine precision number might be enough to yield a different behavior.

Visualizing the resulting rectilinear grid of points then involves a color mapping process. For points that yielded either case (ii) or (iii) we utilize a separate color coding that is not included in the usual colormap, i.e., a pure red value indicates case (ii) and a pure green value indicates (iii). For all other values we treat the result of the iterative process as a complex number and use the polar angle of the complex number as input to a colormap that is periodic, where we chose the Twilight colormap [Bec].

At this point it is also important to note why the fractal behavior of the Newton fractals occurs in the first place. However, giving an in-depth explanation of this process is beyond the scope of this paper and for such an explanation the reader is referred to the report by Drexler et. al [DSB97], which covers the mathematical details. In general, an image generated in the process described here is a direct visualization of a Fatou set, which is an aspect of holomorphic dynamics, and describes the set of all points where a small perturbation of the position leads to the same result, or visually speaking, the Fatou set is the set of all points in the solid colored regions. On the other hand, the Julia set is the complementary of the Fatou set and describes the remaining points, i.e., points for which a small perturbation leads to a different results, or visually speaking, the Julia set is the boundary between solid colored regions.

A Newton fractal now arises from the observation, for a more rigorous mathematical argument see [DSB97], that for points in the Julia set any small perturbation of the position can lead to an arbitrarily large difference as the dynamic process continues. More specifically, it is possible to demonstrate that these small perturbations for any point in the Julia set can lead to any arbitrary position in the entire complex plane. Consequently, the neighborhood of a point in the Julia set necessarily includes points in the Fatou set corresponding to all possible behaviors, but such a border is necessarily a fractal due to requiring the boundary being adjacent to all regions at all points along the border. An important aspect in the emergence of this behavior is the underlying holomorphic dynamics that govern the behavior of a holomorphic process, of which a Newton iteration is an example, as these arguments do not hold up for non-holomorphic functions. Specifically, the functions created as described in Sec. 3, are nowhere complex differentiable so their behavior cannot be governed by holomorphic dynamics. In summary then, the fractal appearance, as described here, can only arise if the underlying function is holomorphic in itself. Regardless, the same visualization of the Fatou set can still give important information about the behavior of the optimizer, even if the Julia set is not a fractal.

5. Optimization Methods

Before discussing the results it is useful to discuss the various optimizers and root finding approaches used in our results and how they are defined to build a solid foundation for comparisons.

5.1. Newton's Method

Newton's method in numerical analysis is a simple root-finding approach that iteratively improves an initial guess z_0 of a root through an iterative process with

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}. \quad (10)$$

5.2. Newton's Method as an Optimizer

A common extension of Newton's method is to apply it to an optimization problem where the goal is not to find the roots of a given function but to instead find the minima of a given function. This then yields the iterative process

$$\mathbf{x}_{n+1} = \mathbf{x}_n - H^{-1}(\mathbf{x}_n) \cdot \nabla f(\mathbf{x}_n). \quad (11)$$

For this method it is important to note that H needs to be positive definite as otherwise the process becomes a maximization instead of a minimization. However, if an H is well defined, this can be readily circumvented by first evaluating the Eigenvalues σ_i of H and ensuring that they are all positive valued by adding a diagonal matrix to H with the diagonal elements set to $-\min \sigma_i$.

5.3. Halley's method

Halley's method is the second Householder method for root finding, where Newton's method is the first Householder method. Halley's

method works by including the second order derivative of f as part of the root finding process to yield an iterative process

$$z_{n+1} = z_n - \frac{2f(z_n)f'(z_n)}{2(f'(z_n))^2 - f(z_n)f''(z_n)}. \quad (12)$$

5.4. Gradient Descent

Gradient descent, also known as Steepest Descent, is a straight forward but widely utilized minimization algorithm that is simply dependant on the gradient of a function and a user defined learning rate γ as

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \nabla f(\mathbf{x}_n). \quad (13)$$

While using a user-defined value γ often works, using a line search algorithm to find a choice for γ that satisfies the Wolfe conditions can significantly improve the convergence behavior, albeit at potentially significant computational overhead. An alternative approach is using the Barzilai-Borwein method to determine γ as

$$\gamma_n = \frac{\langle \mathbf{x}_n - \mathbf{x}_{n-1}, \nabla f(\mathbf{x}_n) - \nabla f(\mathbf{x}_{n-1}) \rangle}{|\nabla f(\mathbf{x}_n) - \nabla f(\mathbf{x}_{n-1})|^2} \quad (14)$$

5.5. BFGS

The BFGS algorithm (named after Broyden Fletcher Goldfarb and Shanno) [Fl13] is an optimization approach that does not rely on an explicitly known Hessian matrix, but instead constructs an approximation of the Hessian matrix during the optimization. For numerical convenience, most BFGS implementations construct an approximation of the inverse Hessian matrix directly as this avoids potentially costly inversion operations. The BFGS algorithm now starts with an initial guess of the Hessian, usually chosen as an identity matrix, and then iteratively first evaluating a new search direction $\mathbf{p}_k = -H_k \nabla f(\mathbf{x}_k)$, calculating a new candidate position along the search direction $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$, with α_k determined using a line search to satisfy the Wolfe conditions, and finally updating the Hessian matrix using $\rho_k = 1/\mathbf{y}_k^T \mathbf{s}_k$, with $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$, as [WN*99]

$$H_{k+1} = \left(I - \rho_k \mathbf{s}_k \mathbf{y}_k^T \right) H_k \left(I - \rho_k \mathbf{y}_k \mathbf{s}_k^T \right) + \rho_k \mathbf{s}_k \mathbf{s}_k^T. \quad (15)$$

5.6. Adam

Adam is a momentum based variant of gradient descent and is one of the most widely used optimization methods in deep learning at the moment [KB14] with a citation, on average, every 37 minutes. In contrast to AdaGrad [DHS11], Adam utilizes two separate momenta per component \mathbf{m} and \mathbf{v} with an additional correction applied to them based on the iteration count $k \geq 1$. Note that for the explanation here a one dimensional optimization for x is assumed and any other component is evaluated analogously. Initially both momentum terms are initialized to zero and are updated according to

$$\begin{aligned} \mathbf{m}_{k+1} &= \beta_1 \mathbf{m}_k + (1 - \beta_1) \frac{\partial}{\partial x} f(\mathbf{x}_k) \\ \mathbf{v}_{k+1} &= \beta_2 \mathbf{v}_k + (1 - \beta_2) \left(\frac{\partial}{\partial x} f(\mathbf{x}_k) \right)^2, \end{aligned} \quad (16)$$

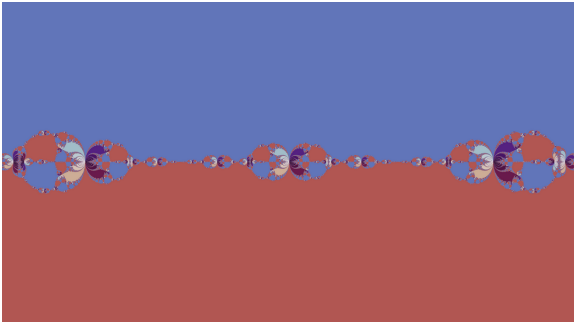


Figure 2: This figure visualizes $f(x) = |x|$ for Newton's method. For this function, even though there should exist a root at $z = 0$, there is no clearly visible basin of convergence towards this root.

where usually $\beta_1 = 0.9$ and $\beta_2 = 0.999$. These values are then corrected for their bias from being initialized as zero as $\hat{\mathbf{m}}_{k+1} = \mathbf{m}_{k+1} / (1 - \beta_1^k)$ and $\hat{\mathbf{v}}_{k+1} = \mathbf{v}_{k+1} / (1 - \beta_2^k)$. The final updated according to Adam, with $\alpha = 10^{-4}$ and $\epsilon = 10^{-8}$, is then defined as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \frac{\hat{\mathbf{m}}_{k+1}}{\sqrt{\hat{\mathbf{v}}_{k+1} + \epsilon}}. \quad (17)$$

6. Results

We implemented our approach in an open source framework [Win22] and ran all our evaluations on a system with an Intel i7-11900K with 64GB of RAM and an Nvidia 3060 Ti, with all computations being performed on the CPU side with parallelization using OpenMP. All computations are performed in double precision arithmetic with a resolution of 3840×2160 for the computed images (resampled to 1280×720 in this document, for full resolution images please refer to the supplementary material) for the domain $[-16/9, 16/9] \times [-1, 1]$. In the results, Newton refers to using the Newton iterations to find the roots of the function, Halley refers to using Halley's method, NRI-O refers to using Newton's method for optimization, NRI-HC refers to using NRI-O but correcting the Hessian to be positive definite, GD refers to Gradient Descent, GD-BB refers to GD with the Barzilai-Borwein method. Adam and BFGS refer to the optimization methods with the same names. For the visual results please refer to Tables 1 and 2.

The first function we considered for our evaluation is a simple polynomial of $f(z) = z^3 - 2z + 1$, which shows a relatively rare behavior for Newton's method and was, accordingly, interesting as a basis for comparisons. For Newton's method in this case there exist loops, between $0 + 0i$ and $1 + 0i$, that attract any iterations falling within their basin of influence and never converge to an actual root of the function. If we compare this result with using Halley's method we can observe a significantly less complicated structure, i.e., the individual patches of equal convergence point are much larger. If Newton's method is used as an optimizer, without correcting the Hessian, the results show no fractal like behavior but also indicate that the method becomes an optimizer for large ranges of starting points. Correcting the hessian on the other hand yields three well defined solid regions of convergence, similar to

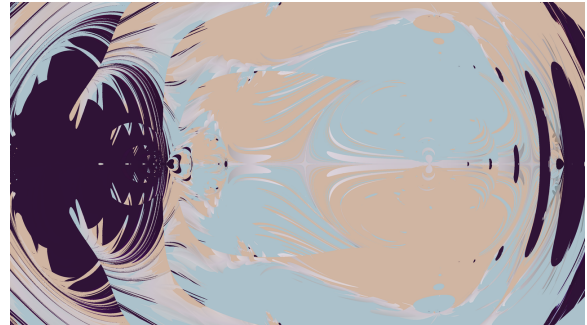


Figure 3: This figure visualizes $f(x) = x^3 - 2x + 1$ using BFGS. While all points in this function do converge to a root, i.e., a global minima, the results are significantly less stable than for other optimization approaches whilst requiring significantly higher computational requirements in this case.

gradient descent. Using GD-BB yields a significantly different behavior as the adaptive learning rate in this method in this specific scenario yields a basin of convergence towards $1 + 0i$, i.e., to one of the points of the attracting cycle, which is not present for a normal GD approach. Finally, Adam yields a similar result to GD and NRI-HC, but shows some islands in the blue and tan regions where the momentum term causes the method to *jump* to the other region.

For the second test function, $f(x) = e^{-x} \sin(x)$, we see similar results for most methods, except for NRI-HC, GD and Adam. In this case, NRI-HC does not yield well defined regions of convergence, as in the previous case, and GD does not converge for large parts of the domain towards a minima, indicated by the gradient between the dark and light region, even after 8192 iterations. Adam, on the other hand, shows a well defined convergence behavior.

Similarly, for the third function $f(x) = \tan(x)$, Adam is the optimizer that shows the best convergence behavior, except for a small region of starting values that failed to converge after the maximum iteration count was reached. NRI-HC and NRI-O, for this case, significantly diminish the basin of convergence towards the real root at $z = 0$. An interesting observation here is that NRI-O yields a fractal pattern, whereas NRI-HC does not. In this scenario, GD and GD-BB, again behave very differently as the adaptive learning rate of GD-BB ensures a convergent behavior across the domain, whereas GD diverges towards ∞ for large parts of the domain. Furthermore, the basins of convergence towards many of the roots are significantly larger for GD-BB and, for some, GD has no basins of convergence for some of the roots.

For the fourth test function, $f(x) = \operatorname{erf}(x)$, we observe results that, overall, are very similar, however, there are two notable exceptions. Firstly, the shape of the basin of convergence towards $z = 0$ is significantly different between Newton's method and Halley's method, whereas for previous examples the results were very comparable. Secondly, using Newton's method as an optimizer, without correcting the Hessian, in this case yields a fractal like behavior where the shape appears to be heavily distorted by the central basin of convergence.

For the final four test cases, see Table 2, we observe a gener-

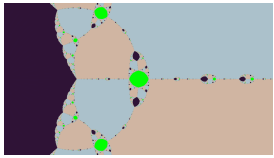
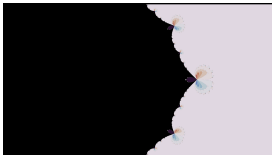

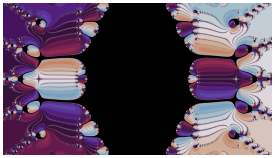

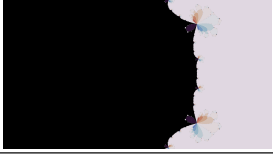
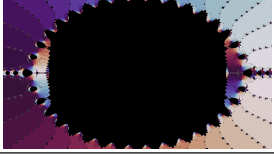

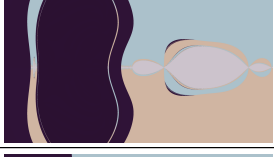
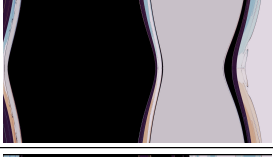
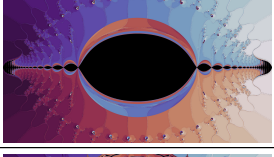
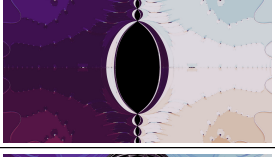

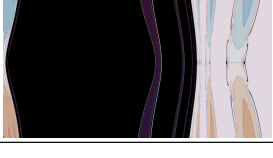

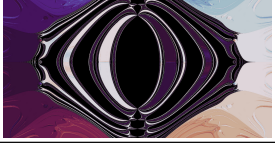



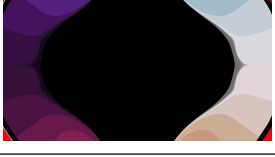

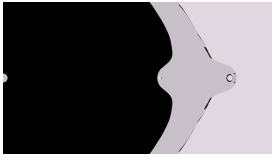
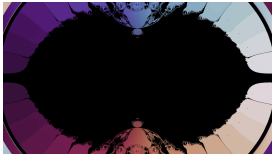


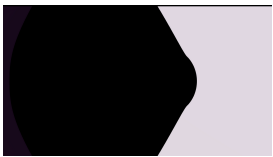


	$f(x) = x^3 - 2x + 1$	$f(x) = e^{-x} \sin(x)$	$f(x) = \tan(x)$	$f(x) = \operatorname{erf}(x)$
Newton-It. as root finder				
Halley's Method				
Newton-It. as optimizer				
Newton-It. using Hessian				
Gradient-Descent				
Gradient-Descent Barzilai-Borwein				
Adam				

Table 1: Results for the first four test functions

ally similar behavior for the first two test cases, where the fractal like-appearance for Newton's method as an optimizer in test case II is very noticeable and, contrary to any prior test case, GD-BB also yields a fractal pattern. For the final two test cases, we observe fundamentally different appearances of the resulting images for all optimizers, which was not as noticeable before hand, except for the teaser image (see Fig. 1). Furthermore, for these two cases, Gradient Descent either did not converge towards any minima or diverged very quickly, which made it impossible to utilize for this case without some adaptive learning rate, be it momentum based or using GD-BB. Finally, the Adam optimizer still yields the smoothest, and largest, basins of convergences across all test func-

tions, which supports the wide adoption of this optimizer for a wide range of tasks.

As penultimate evaluation we considered two further cases. Firstly, we considered a C^0 continuous starting function, namely $f(x) = |x|$, which resulted in a Chebyshev proxy of degree 11217. Even though the polynomial degree was very large, the process still worked, albeit at a much reduced computational performance, and it still yielded an interesting behavior, i.e., Fig. 2 does not show any basin of convergence towards $z = 0$. Finally, we considered the results of applying BFGS to $f(x) = x^3 - 2x + 1$, see Fig. 3, but we observed a significantly more complicated convergence behavior. Investigating these results more closely indicates that the primary

	Test Case I	Test Case II	Test Case III	Test Case IV
Newton-It. as root finder				
Halley's Method				
Newton-It. as optimizer				
Newton-It. using Hessian				
Gradient-Descent				
Gradient-Descent Barzilai-Borwein				
Adam				

Table 2: Test functions based on approximation of arbitrary functions

issue here stems from the approximation of the Hessian, as the iterative process can move very rapidly in these test functions, which means that the Hessian matrix is significantly changing as the process proceeds in such a way that the iterative construction of the Hessian can become error prone. In a worst case this would result in a predicted direction for the optimization that is orthogonal to the gradient of the function and, consequently, any multiple of the predicted direction would not yield an improvement of the process, which in turn causes the process to halt prematurely. Finally, we demonstrate the influence of different hyperparameters for the Adam optimizer for $f(x) = x^5 - 0.5$ in Fig. 4. While in this case different hyperparameters yielded different results, Adam always remained stable and never diverged even at large learning rates or

low momentum parameters. Furthermore, even though the trajectory shown in the results can become oscillatory, this did not negatively impact the overall convergence and, at worst, increased the number of required iterations.

7. Conclusions

In this paper we have presented an intuitive and visual way of visualizing different behaviors in different optimizers regarding the optimization of two dimensional functions based on a flexible test function creation process reliant on Chebyshev proxy functions. Using our open source framework we were able to visualize stark differences in the convergence behavior between different methods

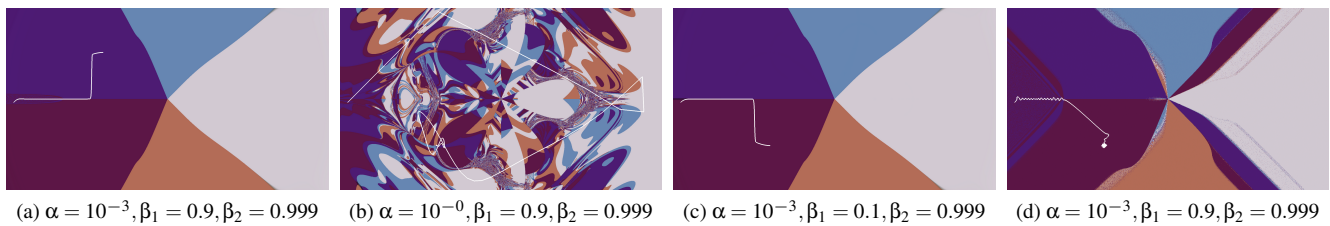


Figure 4: Evaluation of the differences that choosing different hyperparameters make for the Adam optimizer for the function $f(x) = x^5 - 1$ with a trajectory for a single starting point shown for all choices.

on both a fundamental level, i.e., certain methods showing clear fractal like behavior, and on a more nuanced level, i.e., demonstrating differences due to a choice of hyperparameter. Overall, the resulting images are not just visually interesting but also highlight potential areas of future investigation, i.e., if the proposed visual approach can be extended to turn the visual results into quantifiable numbers that allow for objective comparisons between different optimization methods, especially in higher dimensions. Finally, our framework works with a wide variety of input functions, ranging from simple low degree polynomials with stable regions for some methods, to polynomial approximations of piecewise continuous functions requiring over a polynomial degree of over 10000.

References

- [AT17] AURENTZ, JARED L and TREFETHEN, LLOYD N. “Chopping a Chebyshev series”. *ACM Transactions on Mathematical Software (TOMS)* 43.4 (2017), 1–21 [2](#).
- [Bec] BECHTOLD, BASTIAN. *Twilight – A Circular Color Map*. Accessed: 2022-07-04. URL: <https://github.com/bastibe/twilight> [3](#).
- [Ber18] BERNSTEIN, SERGE. “Quelques remarques sur l’interpolation”. *Mathematische Annalen* 79.1 (1918), 1–12 [2](#).
- [Boy01] BOYD, JOHN P. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001 [2](#).
- [Boy02] BOYD, JOHN P. “Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding”. *SIAM Journal on Numerical Analysis* 40.5 (2002), 1666–1682 [2](#).
- [Boy13] BOYD, JOHN P. “Finding the zeros of a univariate equation: proxy rootfinders, Chebyshev interpolation, and the companion matrix”. *SIAM review* 55.2 (2013), 375–396 [2](#).
- [Boy14] BOYD, JOHN P. *Solving Transcendental Equations: The Chebyshev Polynomial Proxy and Other Numerical Rootfinders, Perturbation Series, and Oracles*. Vol. 139. SIAM, 2014 [2](#).
- [Boy95] BOYD, JOHN P. “A Chebyshev polynomial interval-searching method (“Lanczos economization”) for solving a nonlinear equation with application to the nonlinear eigenvalue problem”. *Journal of Computational Physics* 118.1 (1995), 1–8 [2](#).
- [BT04] BATTLES, ZACHARY and TREFETHEN, LLOYD N. “An extension of MATLAB to continuous functions and operators”. *SIAM Journal on Scientific Computing* 25.5 (2004), 1743–1770 [2](#).
- [CC60] CLENSHAW, CHARLES W and CURTIS, ALAN R. “A method for numerical integration on an automatic computer”. *Numerische Mathematik* 2.1 (1960), 197–205 [2](#).
- [Cle55] CLENSHAW, CHARLES W. “A note on the summation of Chebyshev series”. *Mathematics of Computation* 9.51 (1955), 118–120 [2](#).
- [DHS11] DUCHI, JOHN, HAZAN, ELAD, and SINGER, YORAM. “Adaptive subgradient methods for online learning and stochastic optimization.” *Journal of machine learning research* 12.7 (2011) [4](#).
- [DHT14] DRISCOLL, T. A, HALE, N., and TREFETHEN, L. N. *Chebfun guide*. 2014 [2](#).
- [DSB97] DREXLER, M., SOBEY, IAN, and BRACHER, CHRISTIAN. “Fractal Characteristics of Newton’s Method on Polynomials”. (Jan. 1997) [3, 4](#).
- [Fle13] FLETCHER, ROGER. *Practical methods of optimization*. John Wiley & Sons, 2013 [4](#).
- [FW66] FRASER, W and WILSON, MW. “Remarks on the Clenshaw-Curtis quadrature scheme”. *SIAM Review* 8.3 (1966), 322–327 [2](#).
- [HT17] HASHEMI, BEHNAM and TREFETHEN, LLOYD N. “Chebfun in three dimensions”. *SIAM Journal on Scientific Computing* 39.5 (2017), C341–C363 [2](#).
- [Jac90] JACOBS, STANLEY J. “A pseudospectral method for two-point boundary value problems”. *Journal of Computational Physics* 88.1 (1990), 169–182 [2](#).
- [KB14] KINGMA, DIEDERIK P and BA, JIMMY. “Adam: A method for stochastic optimization”. *arXiv preprint arXiv:1412.6980* (2014) [4](#).
- [Mav94] MAVRIPLIS, CATHERINE. “Adaptive mesh strategies for the spectral element method”. *Computer methods in applied mechanics and engineering* 116.1-4 (1994), 77–86 [2](#).
- [MH02] MASON, JOHN C and HANDSCOMB, DAVID C. *Chebyshev polynomials*. CRC press, 2002 [2](#).
- [Ros60] ROSENBRACK, HOHO. “An automatic method for finding the greatest or least value of a function”. *The computer journal* 3.3 (1960), 175–184 [2](#).
- [Run01] RUNGE, C. “Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten”. *Zeitschrift für Mathematik und Physik* 46.224-243 (1901), 20 [2](#).
- [SS80] SLOAN, IH and SMITH, WE. “Product integration with the Clenshaw–Curtis quadrature scheme”. *SIAM Rev* 8 (1980), 322–327 [2](#).
- [TT13] TOWNSEND, ALEX and TREFETHEN, LLOYD N. “An extension of Chebfun to two dimensions”. *SIAM Journal on Scientific Computing* 35.6 (2013), C495–C518 [2](#).
- [Wei85] WEIERSTRASS, K. “Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen”. *Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu Berlin* 2 (1885), 633–639 [2](#).
- [Win22] WINCHENBACH, RENE. *Optimizer Visualization Project*. Accessed: 2022-07-09. 2022. URL: <https://github.com/wi-rc/chebFractals> [5](#).
- [WN*99] WRIGHT, STEPHEN, NOCEDAL, JORGE, et al. “Numerical optimization”. *Springer Science* 35.67-68 (1999), [7, 4](#).