Neural Network Accelerated Implicit Filtering: Integrating Neural Network Surrogates With Provably Convergent Derivative Free Optimization Methods

Brian Irwin [1] **Eldad Haber** [1] **Raviv Gal** [2] **Avi Ziv** [2]

Abstract

In this paper, we introduce neural network accelerated implicit filtering (NNAIF), a novel family of methods for solving noisy derivative free (i.e. black box, zeroth order) optimization problems. NNAIF intelligently combines the established literature on implicit filtering (IF) optimization methods with a neural network (NN) surrogate model of the objective function, resulting in accelerated derivative free methods for unconstrained optimization problems. The NN surrogate model consists of a fixed number of parameters, which can be as few as $\approx 1.3 \times 10^4$, that are updated as NNAIF progresses. We show that NNAIF directly inherits the convergence properties of IF optimization methods, and thus NNAIF is guaranteed to converge towards a critical point of the objective function under appropriate assumptions. Numerical experiments with 31 noisy problems from the CUTEst optimization benchmark set demonstrate the benefits and costs associated with NNAIF. These benefits include NNAIF's ability to minimize structured functions of several thousand variables much more rapidly than well-known alternatives, such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and finite difference based variants of gradient descent (GD) and BFGS, as well as its namesake IF.

1. Introduction

Derivative free optimization (DFO), also referred to as black box optimization (BBO) and zeroth order (ZO) optimization, is an important subfield within numerical optimization with many important practical applications. These applications include many problems arising from business, chemistry, computer science, economics, engineering design, finance, mathematics, medicine, operations research, physics, and transportation (see Rios & Sahinidis (2013), Shan & Wang (2010), Kelley (2011), Audet & Hare (2017), Conn et al. (2009), and references within for specific examples). In such problems, the function $f(\mathbf{x}) : \mathcal{X} \mapsto \mathbb{R}$ is often viewed as a black box with unknown inner workings. Given an input $\mathbf{x} \in \mathcal{X}$, the black box produces an output $f(\mathbf{x})$, but does not produce derivatives of $f(\mathbf{x})$. The output $f(\mathbf{x})$ typically contains some added noise, and so we express $f(\mathbf{x})$ using the decomposition (1) where $f_s(\mathbf{x})$ is a smooth function

$$f(\mathbf{x}) = f_s(\mathbf{x}) + \phi(\mathbf{x}) \tag{1}$$

and $\phi(\mathbf{x})$ is noise. Our goal is to solve the optimization problem (2) using only samples from the noisy black box $f(\mathbf{x})$ given in (1) instead of samples from the (inaccessible)

$$\min_{\mathbf{x} \in \mathcal{X}} \left\{ f_s(\mathbf{x}) \right\} \tag{2}$$

smooth function $f_s(\mathbf{x})$.

The nature of the noise $\phi(\mathbf{x})$ in (1) may be deterministic or stochastic. The high frequency periodic function (3) is a

$$\phi(\mathbf{x}) = \sin\left(\omega \left\|\mathbf{x}\right\|_2\right), \quad \omega \gg 0 \tag{3}$$

simple example of deterministic noise. The high frequency sinusoidal oscillations produce a rough output surface for $f(\mathbf{x})$ with many local minima that can trap derivative based optimizers. A more complicated example of deterministic noise is the truncation error in a numerical simulation, such as an adaptive ordinary differential equation (ODE) solver.

When the noise $\phi(\mathbf{x})$ is stochastic, the value of the noise $\phi(\mathbf{x})$ likely changes each time one samples the output of the black box $f(\mathbf{x})$, even if the inputs $\mathbf{x}$ are unchanged. Drawing independent and identically distributed (IID) samples from a uniform distribution (4) is a simple example

$$\phi(\mathbf{x}) \sim \mathcal{U}(-\epsilon, \epsilon), \quad \epsilon > 0 \tag{4}$$

of stochastic noise. A more complicated example of stochastic noise is when the probability density function (PDF) of the stochastic noise $\phi(\mathbf{x})$ is a parametric function with parameters that depend on $\mathbf{x}$, such as when ϵ is replaced with $\epsilon(\mathbf{x})$ in (4).

[1]Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia, Vancouver, Canada [2]Hybrid Cloud Quality Technologies, IBM Research, Haifa, Israel. Correspondence to: Brian Irwin <birwin@eoas.ubc.ca>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

1.1. Key Problem Characteristics

In this paper, we consider applications where traditional derivative based optimization methods are not feasible. In particular, we focus on problems where:

1. The noise $\phi(\mathbf{x})$ in (1) is stochastic.
2. The optimization problem (2) is unconstrained. Mathematically, $\mathbf{x} \in \mathbb{R}^{N_x}$.
3. The noisy black box function $f(\mathbf{x})$ can be sampled at any point in its domain $\mathcal{X} \equiv \mathbb{R}^{N_x}$, typically in parallel.
4. Evaluating the black box function $f(\mathbf{x})$ is very expensive in terms of time and/or computational resources. To illustrate, computing $f(\mathbf{x})$ may take several hours or even days and require multiple processing units.

1.2. Fundamentals Of Implicit Filtering

While there are a variety of techniques for handling DFO problems (see the recent review in Larson et al. (2019)), we explore the use of neural networks to accelerate implicit filtering (IF) optimization methods (see Kelley (2011) for a comprehensive reference regarding IF methods). IF methods are a hybrid of a grid-search method and a gradient-based local optimization method, and were designed to handle noisy objective function evaluations. As IF methods only involve sampling the function $f(\mathbf{x})$ directly, IF methods are attractive for optimization problems where derivatives are unavailable. IF methods have been proven to give useful results in fields that range from automotive engineering (see Choi et al. (2000)) to hardware verification (see Gal et al. (2021)) to hydrology (see Battermann et al. (2002)) to semiconductor design (see Stoneking et al. (1992)).

The grid-search component of IF methods can be viewed as a trust region based direct search method. At each iteration, the grid-search looks for a better point by sampling the function $f(\mathbf{x})$ along the boundary of a trust region. To illustrate, searching for a better point may consist of sampling $f(\mathbf{x})$ along the vertices of a grid shaped trust region. When a point with a better value of $f(\mathbf{x})$ is found, the grid-search takes that point, and a new trust region centred at the new point is used for future grid-searches until an even better point is found. If the grid-search along the boundary of the trust region fails to find a better point, the method reduces the trust region size and tries to find a better point along the boundary of the smaller trust region.

IF methods improve upon the grid-search approach discussed in the previous paragraph by using the sampled values of $f(\mathbf{x})$ at each iteration to construct an estimate of the gradient $\nabla f_s(\mathbf{x})$. The gradient estimate is then used to perform a gradient descent (GD), Newton, or quasi-Newton update in order to accelerate convergence. This improvement can work well if the gradient estimate is sufficiently accurate. However, the estimate of $\nabla f_s(\mathbf{x})$ is often not sufficiently accurate until relatively late iterations of IF when the sampled points are close together. This improvement is thoroughly discussed in Kelley (2011).

A major disadvantage of IF methods is that, apart from using a quasi-Newton update to estimate curvature, they ignore evaluations of $f(\mathbf{x})$ from previous iterations. Assuming storage is relatively cheap compared to evaluating the noisy black box function $f(\mathbf{x})$, not using the (typically many) evaluations of $f(\mathbf{x})$ computed during the course of IF seems wasteful. Previously computed values of $f(\mathbf{x})$, even in places far from an optimum of $f_s(\mathbf{x})$, typically contain useful information about the structure of the objective function $f_s(\mathbf{x})$. When harnessed appropriately, such information can reduce the amount of computation required to solve (2) using only samples from $f(\mathbf{x})$.

1.3. Main Contribution: Neural Network Acceleration

In this work, we show how to use neural network surrogate models to leverage previously computed values of $f(\mathbf{x})$ to accelerate the progress of IF optimization methods. In particular, we use a neural network (NN) to build an approximate model of the objective function $f_s(\mathbf{x})$ using all previous samples from the black box $f(\mathbf{x})$. The neural network surrogate model is then used to suggest new sampling points distinct from the grid-search points of IF, which can be much better than the grid-search points. We show that such an approach can at best dramatically accelerate IF optimization methods, while at worst having almost the same speed as standard IF methods.

Building a neural network surrogate model can be an expensive task on its own, depending on the amount of data required for the NN surrogate to become sufficiently accurate, and the amount of computation required to train the NN to fit the samples of $f(\mathbf{x})$. However, for the problems we focus on in this paper, fitting and evaluating the NN surrogate model is cheap compared to evaluating the black box function $f(\mathbf{x})$. In this situation, using a NN to build a surrogate model of $f_s(\mathbf{x})$ with samples from $f(\mathbf{x})$ can reduce the total number of evaluations of $f(\mathbf{x})$, and thus translate to a large computational cost reduction overall.

In addition, one may argue that it is possible to not use implicit filtering at all, and instead simply sample the black box function $f(\mathbf{x})$, approximate the objective function $f_s(\mathbf{x})$ using a neural network surrogate model trained on samples from $f(\mathbf{x})$, and then minimize the neural network surrogate model. However, this approach is not competitive in theory nor in practice. As training the NN surrogate to fit the sampled function values may converge slowly, the overall convergence of this approach may be slow. Furthermore, to

the authors' knowledge, there is no proof of convergence for such an approach. On the other hand, IF optimization methods have a strong theoretical background, complete with established convergence properties (see Section 4).

Therefore, improving IF methods by including search points chosen by a NN surrogate model of the objective function $f_s(\mathbf{x})$ allows one to enjoy both worlds, taking points that are suggested by the NN surrogate model whenever they provide sufficient progress, and relying on points obtained by standard IF methods when the NN surrogate model performs poorly. As evaluating the black box function $f(\mathbf{x})$ is expensive, we emphasize that for the problems this paper focuses on, compared to other areas of machine learning, the number of available data points is relatively small. Thus, it is unreasonable to assume that the NN surrogate will yield a very good approximation to the function everywhere.

1.4. Related Works

Conceptually, it is useful to think of building a neural network surrogate model in relation to Newton's method (see Nocedal & Wright (2006)) or quasi-Newton methods, such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method (see Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970)). Newton-like methods build a quadratic approximate model of the objective function, while the NN implements a differentiable nonlinear approximation of the objective function. A NN model has more expressive power than a quadratic model, which means a NN can more effectively model the global structure of the objective function. This increased expressive power compared to a quadratic model comes with the tradeoffs of the NN being more expensive to use in practice and more difficult to work with from a convergence theory perspective. Nonetheless, the spirit of both Newton-like methods and NN acceleration is to use a model of the objective function (i.e. surrogate model) to help choose better points.

With the above in mind, Bayesian optimization (BO) methods (see Močkus (1975), Močkus (1989), Brochu et al. (2010), and Frazier (2018)), such as Gaussian Process (GP) based methods, also build and optimize with a surrogate model when choosing a new point. Specifically, BO methods optimize an acquisition function, such as the expected improvement (EI), obtained from a probabilistic surrogate (the Bayesian posterior specifically). As BO methods use a probabilistic surrogate, they naturally incorporate a measure of uncertainty, unlike Newton-like methods. BO methods are for problems with bound constrained domains, such as $\mathcal{X} \equiv [0,1]^{N_x}$, and not the unconstrained domain $\mathcal{X} \equiv \mathbb{R}^{N_x}$.

Evolution strategy methods (see Beyer (2001), Beyer & Schwefel (2002), and Hansen et al. (2015)), such as the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) method of Hansen & Ostermeier (2001), are also popular for DFO problems. These methods mimic the selection, variation, and recombination aspects of biological evolution. Covariance matrix adaptation is an important component of CMA-ES, as it effectively produces a local quadratic approximation of the objective function, similar to Newton-like methods. Thus, CMA-ES can be viewed as an evolutionary technique that employs an adaptable quadratic surrogate of the objective function.

1.5. Organization Of The Paper

The rest of this paper is organized as follows. Section 2.1 reviews the mathematical background of IF optimization methods, and Section 2.2 describes the type of residual neural network surrogate models used in this work. Next, Section 3 introduces neural network accelerated implicit filtering (NNAIF), the main contribution of this paper. Section 4 presents a convergence result for NNAIF. Section 5 presents the results of numerical experiments comparing the performance of variants of NNAIF to well-known alternatives, including CMA-ES and finite difference based variants of gradient descent and BFGS, on a variety of noisy minimization problems of dimension $N_x = 2$ to $N_x = 9000$ from the CUTEst optimization benchmark set. Section 6 concludes the paper and provides directions for future work.

2. Mathematical Background

In this section, we review the mathematics of both implicit filtering optimization methods and the type of residual neural networks used as surrogate models in the numerical experiments in Section 5. Throughout the paper, bold uppercase letters (e.g. $\mathbf{I}$) represent matrices, bold lowercase letters (e.g. $\mathbf{x}$) represent vectors, and non-bold letters (e.g. ω) represent scalars.

2.1. Implicit Filtering

As previously mentioned, one attractive way to solve noisy black box optimization problems is implicit filtering. Below, we provide an overview of implicit filtering. For more detail, see Kelley (2011), which introduces IF optimization methods, applies them to several problems, and reviews their convergence properties and implementation details.

Given the current point $\mathbf{x}_k$ at iteration k, a simple way to find a possible better point is to sample the black box function $f(\mathbf{x})$ in a trust region around the current point $\mathbf{x}_k$. To this end, we evaluate $f(\mathbf{x})$ at points $\mathbf{x}_k + h_k \mathbf{v}_j$ for $j \in \{1, \ldots, J\}$, where the $\mathbf{v}_j$ are unit vectors (i.e. $\|\mathbf{v}_j\|_2 = 1$). The parameter h_k represents the trust region (or stencil) size and defines the resolution of the search. J is the number of search directions, which determines how many new points to sample $f(\mathbf{x})$ at.

Traditionally, the matrix $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_J] \in \mathbb{R}^{N_x \times J}$ is

the central difference stencil $\mathbf{V}_{\mathrm{CD}} \coloneqq [\mathbf{I}, -\mathbf{I}] \in \mathbb{R}^{N_x \times 2N_x}$. However, the directions $\mathbf{v}_j$ can be chosen as points on another grid, such as the asymmetric positive basis (a-PBS) stencil defined by

$$\mathbf{V}_{\mathrm{a-PBS}} \coloneqq \left[\mathbf{I}, -\frac{1}{\sqrt{N_x}}\mathbf{1}\right] \in \mathbb{R}^{N_x \times (N_x+1)}\,, \qquad (5)$$

where $\mathbf{1} \coloneqq [1, 1, \ldots, 1]^{\mathrm{T}} \in \mathbb{R}^{N_x}$. At times, it is also desirable to draw directions $\mathbf{v}_j$ randomly from some predefined distribution, such as the Rademacher distribution or uniformly on the surface of the N_x dimensional hypersphere.

Using the matrix of directions $\mathbf{V}$, the vector of sampled values of the black box $\mathbf{f}_k$ defined by

$$\mathbf{f}_k \coloneqq [f(\mathbf{x}_k + h_k\mathbf{v}_1), \ldots, f(\mathbf{x}_k + h_k\mathbf{v}_J)]^{\mathrm{T}} \in \mathbb{R}^J\,, \quad (6)$$

and the difference vector $\delta(f, \mathbf{x}, \mathbf{V}, h) \in \mathbb{R}^J$ given by

$$\delta(f, \mathbf{x}, \mathbf{V}, h) \coloneqq \begin{bmatrix} f(\mathbf{x} + h\mathbf{v}_1) - f(\mathbf{x}), \\ \vdots \\ f(\mathbf{x} + h\mathbf{v}_J) - f(\mathbf{x}) \end{bmatrix}\,, \qquad (7)$$

we define the *stencil gradient* $\nabla f_{S(\mathbf{x},h,\mathbf{V})} \in \mathbb{R}^{N_x}$ as the solution of the regularized linear least squares problem

$$\underset{\mathbf{q}\in\mathbb{R}^{N_x}}{\arg\min}\left\{\frac{1}{2}\left\|h\mathbf{V}^{\mathrm{T}}\mathbf{q} - \delta(f, \mathbf{x}, \mathbf{V}, h)\right\|_2^2 + \frac{\beta}{2}\left\|\mathbf{q}\right\|_2^2\right\} \quad (8)$$

with $\beta > 0$ being a small regularization parameter that guarantees the solution of (8) exists and is unique.

The stencil size h_k of implicit filtering shrinks by a factor $\tau_{\mathrm{tr}} \in (0, 1)$ if either a stencil failure occurs or the stencil gradient is sufficiently small. A *stencil failure* occurs when

$$f(\mathbf{x}_k) \leq f(\mathbf{x}_k + h_k\mathbf{v}_j)\,, \quad \forall j \in \{1, \ldots, J\}\,. \qquad (9)$$

Algorithm 1 summarizes the general structure of implicit filtering, which is designed to partially mimic a finite difference based gradient descent or Newton-like method when stencil failure does not occur and the stencil gradient is larger than $\mathcal{O}(h_k)$. The arguments of Algorithm 1 are the black box function $f(\mathbf{x})$, initialization point $\mathbf{x}_0$, initial stencil size $h_0 > 0$, minimum stencil size $h_{\min} \geq 0$, shrinkage factor $\tau_{\mathrm{tr}} \in (0, 1)$, stencil gradient tolerance $\tau_{\mathrm{gr}} > 0$, regularization parameter $\beta > 0$, and number of directions J.

Algorithm 1 Implicit Filtering Minimization Routine

procedure IF-MIN($f(\mathbf{x})$, $\mathbf{x}_0$, h_0, $h_{\min}$, τ_{tr}, τ_{gr}, β, J)
  Measure $f_0 \leftarrow f(\mathbf{x}_0)$
  while $h_k > h_{\min}$ **do**
    Measure $f(\mathbf{x}_k + h_k\mathbf{v}_j)$ for $j \in \{1, \ldots, J\}$
    Compute $\nabla f_{S(\mathbf{x}_k,h_k,\mathbf{V})}$ by solving (8)
    if (9) is True or $\left\|\nabla f_{S(\mathbf{x}_k,h_k,\mathbf{V})}\right\|_2 \leq \tau_{\mathrm{gr}} h_k$ **then**
      Set $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k$ and $h_{k+1} \leftarrow \tau_{\mathrm{tr}} h_k$
    else
      Compute $f^{\mathrm{cs}}_{\mathrm{try}} \coloneqq \min_{j\in\{1,\ldots,J\}}\{\mathbf{f}_k\}$
      Update inverse Hessian estimate $\mathbf{H}_k$
      Compute $\mathbf{s}_k \leftarrow -\mathbf{H}_k\nabla f_{S(\mathbf{x}_k,h_k,\mathbf{V})}$
      Use a line search along $\mathbf{s}_k$ to obtain $f^{\mathrm{ls}}_{\mathrm{try}}$, $\mathbf{x}^{\mathrm{ls}}_{\mathrm{try}}$
      if $f^{\mathrm{cs}}_{\mathrm{try}} < f^{\mathrm{ls}}_{\mathrm{try}}$ **then**
        Set $\mathbf{x}_{k+1} \leftarrow \arg\min_{j\in\{1,\ldots,J\}}\{\mathbf{f}_k\}$
      else
        Set $\mathbf{x}_{k+1} \leftarrow \mathbf{x}^{\mathrm{ls}}_{\mathrm{try}}$
      end if
      Set $h_{k+1} \leftarrow h_k$
    end if
  end while
  return $\mathbf{x}_{k+1}$ and h_{k+1}
end procedure

2.2. Residual Neural Network Surrogate Models

Our goal is to build a relatively inexpensive surrogate model of the function $f_s(\mathbf{x})$ such that it can be probed to obtain an approximate minimizer of $f_s(\mathbf{x})$. Neural networks are a class of function approximators that use decomposition to approximate a function. In this work, we chiefly use residual neural networks (see He et al. (2016) and Li et al. (2018)), which have proven to be easy to train for difficult and highly nonlinear tasks. The networks we use have the following general structure for $p \in \{1, \ldots, P-2\}$

$$\mathbf{y}_0 = \mathbf{x} \qquad (10)$$
$$\mathbf{y}_1 = \mathbf{G}_0\sigma_0(\mathbf{K}_0\mathbf{y}_0 + \mathbf{b}_0) \qquad (11)$$
$$\mathbf{y}_{p+1} = \mathbf{W}_p\mathbf{y}_p + \mathbf{G}_p\sigma_p(\mathbf{K}_p\mathbf{y}_p + \mathbf{b}_p) \qquad (12)$$
$$\mathbf{y}_P = \mathbf{K}_{P-1}\mathbf{y}_{P-1} + \mathbf{b}_{P-1} \qquad (13)$$

From here on, we choose $\mathbf{W}_p$ to be the identity matrix $\mathbf{I}$, and the dimensions of $\mathbf{y}_p$ and $\mathbf{y}_{p+1}$ are always consistent. The P sets of parameters $\theta_p = \{\mathbf{G}_p, \mathbf{K}_p, \mathbf{b}_p\}, p \in \{0, \ldots, P-2\}$ and $\theta_{P-1} = \{\mathbf{K}_{P-1}, \mathbf{b}_{P-1}\}$ are trainable parameters. For the neural networks used in this work, the $\mathbf{K}_p$ are dense matrices. However, for other applications, one may consider sparse matrices, such as convolutions or other special structures. For the experiments in Section 5, we choose $\mathbf{G}_p = \alpha\mathbf{I}$ with $\alpha > 0$, but one could also choose $\mathbf{G}_p = -\alpha\mathbf{K}_p^{\mathrm{T}}$. We set the activation functions $\sigma_p(\cdot)$ as $\mathrm{ReLU(z)} \coloneqq \max(0, \mathrm{z})$. This type of network is stable and robust as shown in Haber & Ruthotto (2017), especially when the intrinsic dimension of the output is smaller than the input. For brevity, we concatenate all the trainable neural network parameters into a single notation $\mathbf{\Theta} = \{\theta_0, \ldots, \theta_{P-1}\}$.

By choosing $\mathbf{y}_0 = \mathbf{x}$, we can forward propagate to obtain a vector $\mathbf{y}_P(\mathbf{x}, \boldsymbol{\Theta}) \in \mathbb{R}^{N_{\text{out}}}$, where N_{out} is the dimension of the neural network output. Using the vector $\mathbf{y}_P(\mathbf{x}, \boldsymbol{\Theta})$, we now want to obtain a surrogate $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta})$ to the smooth function $f_s(\mathbf{x})$. There are many different approaches to obtain such an approximation. For example, it is possible to choose a quadratic model $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}) = \frac{1}{2}\|\mathbf{y}_P(\mathbf{x}, \boldsymbol{\Theta})\|_2^2$, which is bounded from below. One can also simply set $N_{\text{out}} = 1$ and choose $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}) = \mathbf{y}_P(\mathbf{x}, \boldsymbol{\Theta})$. Choices of $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta})$ that incorporate specific information about the true function $f_s(\mathbf{x})$ can also be used.

Once chosen, the surrogate $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta})$ can be used to guide where to probe the black box function $f(\mathbf{x})$, and fit to samples of the black box using a loss function $\Psi(\mathbf{x}, \boldsymbol{\Theta})$, like

$$\Psi(\mathbf{x}, \boldsymbol{\Theta}) = \frac{1}{2}\mathbf{r}(\mathbf{x}, \boldsymbol{\Theta})^{\mathrm{T}}\mathbf{r}(\mathbf{x}, \boldsymbol{\Theta}) \tag{14}$$

for some residual $\mathbf{r}(\mathbf{x}, \boldsymbol{\Theta})$. By using a matrix of M points $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_M] \in \mathbb{R}^{N_x \times M}$ and corresponding vector of black box values $\mathbf{f}(\mathbf{X}) \coloneqq [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_M)]^{\mathrm{T}} \in \mathbb{R}^M$, we can train the neural network given by (10) - (13) and assess its parameters $\boldsymbol{\Theta}$ by minimizing $\Psi(\mathbf{x}, \boldsymbol{\Theta})$ over the dataset given by $\mathbf{X}$ and $\mathbf{f}(\mathbf{X})$. In this work, we use a squared error to estimate the neural network parameters. In (14), this corresponds to choosing $\mathbf{r}(\mathbf{x}, \boldsymbol{\Theta}) = f(\mathbf{x}) - \widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta})$. Once the neural network is trained, we have an approximation to the objective function $f_s(\mathbf{x})$ that can be used to accelerate implicit filtering.

3. Accelerating Implicit Filtering

We now combine implicit filtering with the neural network surrogate model. We start by using a set of initial points $\mathbf{X}_0$ and corresponding function evaluations $\mathbf{f}(\mathbf{X}_0)$ to train the neural network surrogate model $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta})$. At each iteration of Algorithm 1, we obtain J new points $\mathbf{X}_k \coloneqq \mathbf{x}_k\mathbf{1}^{\mathrm{T}} + h_k\mathbf{V}$ and their corresponding function values $\mathbf{f}_k$. These new points and function values can be added to the set of all previous points and function values, and used to retrain the neural network surrogate model by minimizing a loss function $\Psi(\mathbf{x}, \boldsymbol{\Theta})$. This gives an approximately optimal parameter vector $\boldsymbol{\Theta}_k$. After the neural network is retrained, we hold an approximate surrogate model $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}_k)$ of $f_s(\mathbf{x})$.

This surrogate model $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}_k)$ can be used to obtain an approximate minimizer of the objective function $f_s(\mathbf{x})$. To this end, we can apply standard optimization techniques to the differentiable neural network surrogate model $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}_k)$ to propose a new point to evaluate $f(\mathbf{x})$ at. At each iteration k, we use at most a fixed number $\ell_{\max}$ of iterations of a standard optimization method. Algorithm 2 demonstrates this surrogate model optimization procedure with gradient descent. In line 4 of Algorithm 2, the learning rate μ_ℓ can be set using standard techniques, such as a fixed choice, backtracking line search, or the Adaptive Moment Estimation (ADAM) technique (see Kingma & Ba (2014)). In line 5, the gradient of the surrogate with respect to only the function argument $\mathbf{x}$ is computed using automatic differentiation (AD). In line 6, we track the size of the update $\Delta\mathbf{x}$. For an appropriate norm $\|\cdot\|$, the iteration terminates if $\|\Delta\mathbf{x}\| > h_k$, which guarantees that we stay within the implicit filtering trust region.

Algorithm 2 Surrogate Model Gradient Descent

1: **procedure** SURR-MIN-GD($\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}_k)$, $\mathbf{x}_k$, h_k, $\ell_{\max}$)
2: Set $\mathbf{z}_0 \leftarrow \mathbf{x}_k$ and $\ell \leftarrow 0$
3: **while** $\ell < \ell_{\max}$ and $\|\Delta\mathbf{x}\| \leq h_k$ **do**
4: Choose the step size μ_ℓ
5: Set $\mathbf{z}_{\ell+1} \leftarrow \mathbf{z}_\ell - \mu_\ell \nabla_{\mathbf{x}} \widehat{f}_s(\mathbf{z}_\ell, \boldsymbol{\Theta}_k)$
6: Compute $\Delta\mathbf{x} = \mathbf{z}_{\ell+1} - \mathbf{z}_0$
7: **end while**
8: **return** $\mathbf{z}_{\ell+1}$
9: **end procedure**

The goal of Algorithm 2 is not to obtain the global minimum of the surrogate model, as the surrogate model $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}_k)$ is only an approximation of the objective function $f_s(\mathbf{x})$. A reduction in the value of the black box $f(\mathbf{x})$ is sufficient. Therefore, we typically use a small number of steps $\ell_{\max}$ to ensure a reduction in the value of the surrogate model. The point returned by Algorithm 2 clearly reduces the value of the surrogate model, and thus we use it as a potential candidate point for improving upon the current point $\mathbf{x}_k$. This is an intuitive use of the surrogate model $\widehat{f}_s(\mathbf{x}, \boldsymbol{\Theta}_k)$.

However, there is another use of the surrogate model that can improve performance. Algorithm 1 does not use any form of filter or quality control when choosing new search points. Without a filter or some valid assumption, the value of $f(\mathbf{x})$ at the new search points is basically random. However, we can use the surrogate model to propose new points that are more likely to be of good quality and decrease $f_s(\mathbf{x})$. Rather than choosing a point $\mathbf{x}_k + h_k\mathbf{v}_j$ and computing its value $f(\mathbf{x}_k + h_k\mathbf{v}_j)$, we instead first compute its surrogate value $\widehat{f}_s(\mathbf{x}_k + h_k\mathbf{v}_j, \boldsymbol{\Theta}_k)$ and use this value as part of a filter. If the filter test

$$\widehat{f}_s(\mathbf{x}_k + h_k\mathbf{v}_j, \boldsymbol{\Theta}_k) \leq f(\mathbf{x}_k) \tag{15}$$

is true, then we pass the point $\mathbf{x}_k + h_k\mathbf{v}_j$ to the black box function $f(\mathbf{x})$ and sample the value of $f(\mathbf{x}_k + h_k\mathbf{v}_j)$.

Using the surrogate model to filter points can be very effective when the surrogate model provides a reasonably good approximation to $f_s(\mathbf{x})$. However, this is not always the case, especially during early iterations of Algorithm 1 when

only a small number of points are available for training the surrogate model. As a result, we divide the search directions into two groups. The first group, called the exploration group, chooses points on a grid or at random as described in Section 2.1. The second group, called the exploitation group, uses the surrogate model to filter points based on (15). We have found experimentally that starting with a large number of exploration points, and then reducing them to about 20% of the total points, can yield good results.

Algorithm 3 combines the surrogate filtering and optimization with implicit filtering into a family of methods we call Neural Network Accelerated Implicit Filtering (NNAIF). The arguments in Algorithm 3 are the initial stencil size $h_0 > 0$, a stencil size $h_{\min}^{\text{surr}} \geq 0$ below which we no longer train the surrogate model, a minimum required decrease in the surrogate model value $\varepsilon_{\text{dec}}^{\text{surr}} \geq 0$, the number of exploration directions J_{st} and exploitation directions J_f, and a maximum number of iterations $\iota_{\max}$.

Algorithm 3 Neural Network Accelerated Implicit Filtering

procedure NNAIF(h_0, $h_{\min}^{\text{surr}}$, $\varepsilon_{\text{dec}}^{\text{surr}}$, J_{st}, J_f, $\iota_{\max}$)
  Set $J \leftarrow J_{st} + J_f$
  while $\iota < \iota_{\max}$ **do**
    Choose J_{st} directions $\mathbf{V}_{\text{st}}$ with no filter
    Choose J_f directions $\mathbf{V}_f$ filtered using (15)
    Combine all directions into $\mathbf{V} \leftarrow [\mathbf{V}_{\text{st}}, \mathbf{V}_f]$
    Measure $f(\mathbf{x}_k + h_k \mathbf{v}_j)$ for $j \in \{1, \ldots, J\}$
    Store the J evaluations of $f(\mathbf{x})$
    if $h_k > h_{\min}^{\text{surr}}$ **then**
      Train $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_k)$ with stored data
    end if
    Set $\mathbf{x}_{\text{try}}^{\text{surr}}$ as the output of Algorithm 2
    Compute $f_{\text{try}}^{\text{surr}} \coloneqq f(\mathbf{x}_{\text{try}}^{\text{surr}})$
    if $f_{\text{try}}^{\text{surr}} < f(\mathbf{x}_k) - \varepsilon_{\text{dec}}^{\text{surr}}$ **then**
      Set $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_{\text{try}}^{\text{surr}}$ and $h_{k+1} \leftarrow h_k$
    else
      Set $\mathbf{x}_{k+1}$, h_{k+1} to the outputs of Algorithm 1
    end if
  end while
  return $\mathbf{x}_{k+1}$ and h_{k+1}
end procedure

The idea behind Algorithm 3 is to use the surrogate function $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_k)$ obtained by training a neural network to guide the optimization. However, since the amount of data needed for $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_k)$ to become sufficiently accurate may be rather large, Algorithm 3 relies on Algorithm 1 to compensate whenever $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_k)$ fails to make sufficient progress. From experiments, we have observed that when $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_k)$ is trained on a sufficient number of points, it significantly aids convergence initially, and then does not provide a significant advantage near the end of the optimization process.

Finally, an important point to consider is the overhead time due to the surrogate model in Algorithm 3. This added time compared to Algorithm 1 is due to training the surrogate model and approximately minimizing it. While this may not be a trivial amount of time, for the problems focused on in this paper where evaluations of $f(\mathbf{x})$ are expensive and can require significant computational resources, this time is relatively insignificant, and well spent in the sense that it can significantly reduce the total computational time by avoiding evaluations of $f(\mathbf{x})$. As a concrete example, in our numerical experiments, the added time introduced by the surrogate model at each iteration is on the order of seconds or less, not minutes or hours. Furthermore, the cost of retraining the surrogate model can be reduced by using techniques such as warm starting and incremental training.

4. Convergence Of NNAIF

In this section, we outline how Algorithm 3 inherits the asymptotic convergence properties of implicit filtering. For in-depth treatments of the convergence properties of implicit filtering, we refer the reader to Gilmore & Kelley (1995), Bortz & Kelley (1998), Kelley (1999), Choi & Kelley (2000), and Kelley (2011). We assume the function $f(\mathbf{x})$ is bounded below and can be decomposed as in (1). Following Chapter 5 of Kelley (2011), the key observation is that stencil failure (i.e. none of the function values on the stencil improve upon the base point $\mathbf{x}_k$) with a positive spanning set of directions is sufficient to conclude that $\|\nabla f_s\|_2 = \mathcal{O}(h_k)$. The unfamiliar reader may refer to Appendix A for a definition of a positive spanning set, as well as Conn et al. (2009).

Denote the stencil S defined by the set of directions $\mathbf{V}$ as

$$S(\mathbf{x}, h, \mathbf{V}) \coloneqq \{\mathbf{z} \mid \mathbf{z} = \mathbf{x} + h\mathbf{v}_j, 1 \leq j \leq J\} \tag{16}$$

and define the local norm of the noise as

$$\|\phi\|_{S(\mathbf{x},h,\mathbf{V})} \coloneqq \max_{\mathbf{z} \in \{\mathbf{x}\} \cup S(\mathbf{x},h,\mathbf{V})} \{|\phi(\mathbf{z})|\} \tag{17}$$

and the condition number of the positive spanning set $\mathbf{V}$ as

$$\kappa(\mathbf{V}) \coloneqq \sqrt{N_x} \min \{ \|\mathbf{A}\|_\infty \} \tag{18}$$

where $\|\mathbf{A}\|_\infty \coloneqq \sup_{\mathbf{x} \neq 0} \{ \frac{\|\mathbf{A}\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \}$ and the $2N_x \times J$ matrix $\mathbf{A}$ is constrained such that its entries are nonnegative and $\mathbf{A}\mathbf{V}^{\mathrm{T}} = [\mathbf{U}, -\mathbf{U}]^{\mathrm{T}}$ for some orthogonal matrix $\mathbf{U}$. With these definitions in hand, we now state a key theorem. The proof is originally given on pages 75 and 76 of Kelley (2011), but we include it in Appendix B for completeness.

Theorem 5.7 - Kelley, 2011. *Let $f(\mathbf{x})$ satisfy (1). Let ∇f_s be Lipschitz continuous with Lipschitz constant L. Let $\mathbf{V}$ be a positive spanning set. Then stencil failure implies that*

$$\|\nabla f_s(\mathbf{x})\|_2 \leq \kappa(\mathbf{V})\left(\frac{Lh}{2} + \frac{\|\phi\|_{S(\mathbf{x},h,\mathbf{V})}}{h}\right) \tag{19}$$

Proof. See Appendix B. □

As a result of (19), Algorithm 3 is guaranteed to converge to a critical point of $f_s(\mathbf{x})$ when the conditions of Theorem 5.7 are satisfied, $h_{\min} = 0$, and

$$\lim_{k\to\infty} \frac{\|\phi\|_{S(\mathbf{x}_k,h_k,\mathbf{V})}}{h_k} = 0\,. \tag{20}$$

Mathematically,

$$\liminf_{k\to\infty} \|\nabla f_s(\mathbf{x}_k)\|_2 = 0\,. \tag{21}$$

To see this, observe that an iteration of Algorithm 3 ends with either a decrease in $f(\mathbf{x})$ or stencil failure, and let $\{\mathbf{x}_{k_i}\}$ be an infinite subsequence of $\{\mathbf{x}_k\}$ for which the iteration terminates with stencil failure. As $h_{\min} = 0$, $h_{k_i} \to 0$ (i.e. h eventually tends to zero), which combines with (19) and (20) to give (21).

5. Numerical Experiments

In this section, we investigate the empirical behaviour of NNAIF via numerical experiments with a variety of well-known optimization test problems. Software for the experiments uses the PyTorch (Paszke et al., 2019) and PyCUTEst (Fowkes et al., 2022) libraries. All experiments were run on a desktop computer equipped with an NVIDIA RTX 2080 TI GPU and the Ubuntu 20.04 LTS Linux operating system.

5.1. Visualizing Neural Network Surrogate Models

To build intuition regarding how NN surrogate models behave as NNAIF progresses, Figure 1 visualizes the evolution of a NN surrogate model when NNAIF is used to minimize the well-known Rosenbrock function (Rosenbrock, 1960). Here, NNAIF samples from the noisy black box $f(\mathbf{x})$ which decomposes as in (1) with the smooth objective function $f_s(\mathbf{x})$ being the Rosenbrock function and the noise $\phi(\mathbf{x})$ being stochastic and distributed according to a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 10$ (i.e. $\phi(\mathbf{x}) \sim \mathcal{N}(0, 100)$). Overall, Figure 1 demonstrates that even a crude NN surrogate can be useful. The initial NN surrogate shown in the bottom left of Figure 1 resembles a linear model within the circular trust region, which provides a reasonable descent direction. The NN surrogate at iteration $k = 5$ shown in the bottom right of Figure 1 captures the high-level structure of the Rosenbrock function.

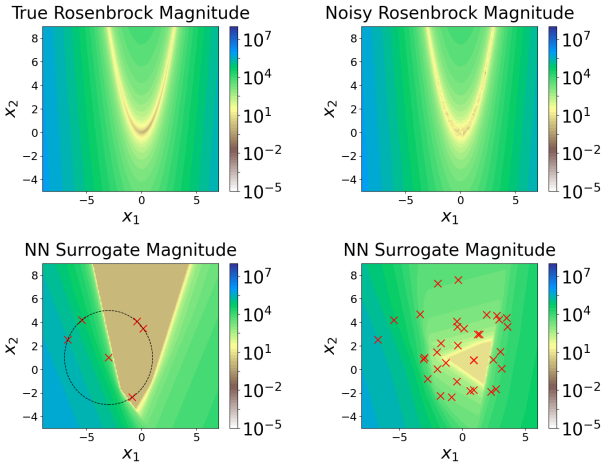

Figure 1. Visualizing the evolution of a NN surrogate model of the Rosenbrock function as NNAIF progresses. Top left: absolute values $|f_s(\mathbf{x})|$ of the Rosenbrock function without noise added. Top right: absolute values $|f(\mathbf{x})|$ of the Rosenbrock function with $\phi(\mathbf{x}) \sim \mathcal{N}(0, 100)$ stochastic noise added. Bottom left: absolute values $\left|\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_0)\right|$ of NN surrogate model fit to 6 initial evaluations of $f(\mathbf{x})$; the 6 training data points are shown via red x marks and the initial trust region with radius $h_0 = 4$ is shown via the black circle. Bottom right: absolute values $\left|\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_5)\right|$ of NN surrogate fit to more data at iteration $k = 5$.

5.2. CUTEst Problems With Additive Uniform Noise

In this subsection, we compare the performance of the methods shown in Figure 2 on 15 problems from the CUTEst optimization benchmark set (see Gould et al. (2019), Gould et al. (2015), and Appendix C). In Appendix D, we do the same for 16 additional CUTEst problems. As an overview, some of the selected CUTEst problems can be interpreted as least squares type problems (e.g. ARGTRIGLS), some of the problems are ill-conditioned or singular type problems (e.g. BOXPOWER), some of the problems are well known nonlinear optimization test problems (e.g. ROSENBR) or extensions of them (e.g. CHNROSNB, SROSENBR), and some of the problems come from application scenarios (e.g. COATING). As shown in Figure 3, Figure 4, and Table 1, the selected CUTEst problems vary in size from $N_x = 2$ to $N_x = 9000$. Noise $\phi(\mathbf{x})$ is added to each CUTEst problem by sampling from a uniform distribution $\mathcal{U}(-\epsilon, \epsilon)$, where for each problem $\epsilon = 10^{-4}\,|f_s(\mathbf{x}_0)|$ to ensure that the noise $\phi(\mathbf{x})$ does not initially dominate the smooth function $f_s(\mathbf{x})$.

As $f(\mathbf{x})$ is expensive to evaluate, we set a budget of 2000 evaluations of $f(\mathbf{x})$ in the experiments shown in this subsection and Appendix D, and terminate each optimization method shown in Figure 2 once the 2000 evaluation budget is exceeded. Each optimization method uses one set of hyperparameters for all 31 CUTEst problems (i.e. we do

not tune each method for each individual CUTEst problem), apart from the number of directions J in which to sample $f(\mathbf{x})$ at per iteration. All the optimization methods, excluding Estimation of Multivariate Normal Algorithm (EMNA) and CMA-ES, use the deterministic a-PBS stencil given by (5) for the low dimensional ($0 < N_x \leq 50$) problems, and randomly sample 30 points uniformly on the surface of the N_x dimensional hypersphere for all other problems. EMNA and CMA-ES use population sizes of $N_x + 1$ for the low dimensional problems, and 30 for all other problems. Both CMA-ES and EMNA use an initial standard deviation $\sigma_0 = 10^{-1}$. We use the active version of CMA-ES. EMNA keeps the fittest half of the population at each iteration.

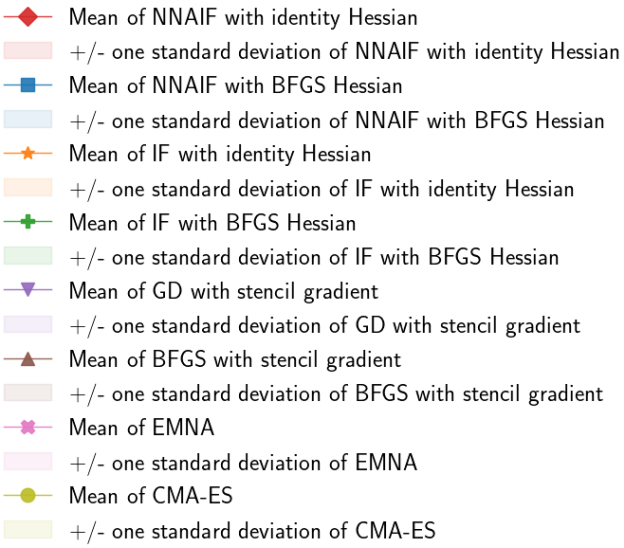

Figure 2. Legend for Figures 3, 4, and 5 in Section 5 and Figures 6, 7, and 8 in Appendix D. Means and standard deviations are calculated using 25 independent runs of each method per problem.

With regards to (10) - (13), the NN surrogate used by NNAIF has $\mathbf{K}_0 \in \mathbb{R}^{50 \times N_x}$, $\mathbf{K}_p \in \mathbb{R}^{50 \times 50}$, $\mathbf{K}_{P-1} \in \mathbb{R}^{1 \times 50}$, and $P = 7$, giving a total of $|\mathbf{\Theta}| = 12851 + 50N_x$ trainable parameters. We use the simple choice $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}) = \mathbf{y}_P(\mathbf{x}, \mathbf{\Theta})$. We use the ADAM optimizer to train $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_k)$, and to approximately minimize $\widehat{f}_s(\mathbf{x}, \mathbf{\Theta}_k)$ with respect to $\mathbf{x}$ in Algorithm 2. For NNAIF, $h_{\min}^{\text{surr}} = 10^{-2}$, $\varepsilon_{\text{dec}}^{\text{surr}} = 10^{-3}$, and we attempt to set $J_{st} = 20$ and $J_f = 10$, but if the filter test (15) does not accept J_f points within 20 tries, we ignore it for the remaining points. For both NNAIF and IF, we set $h_0 = 10$, $h_{\min} = 10^{-8}$, $\tau_{\text{tr}} = 0.5$, $\tau_{\text{gr}} = 10^{-2}$, and $\beta = 10^{-5}$. For the stencil gradient based BFGS and GD methods, the stencil size is held fixed $\forall k$ at resolution $h_k = 10^{-4}$. We set the initial step size as 1 for the NNAIF, IF, and stencil gradient backtracking line searches, and the backtracking factor τ_{ls} as 0.1. We allow up to 3 backtracks for NNAIF and IF, as recommended in Kelley (2011), and up to 75 backtracks for the stencil gradient based methods. NNAIF and IF use the Armijo condition with $c = 0$, while the stencil gradient based methods use $c = 10^{-4}$.

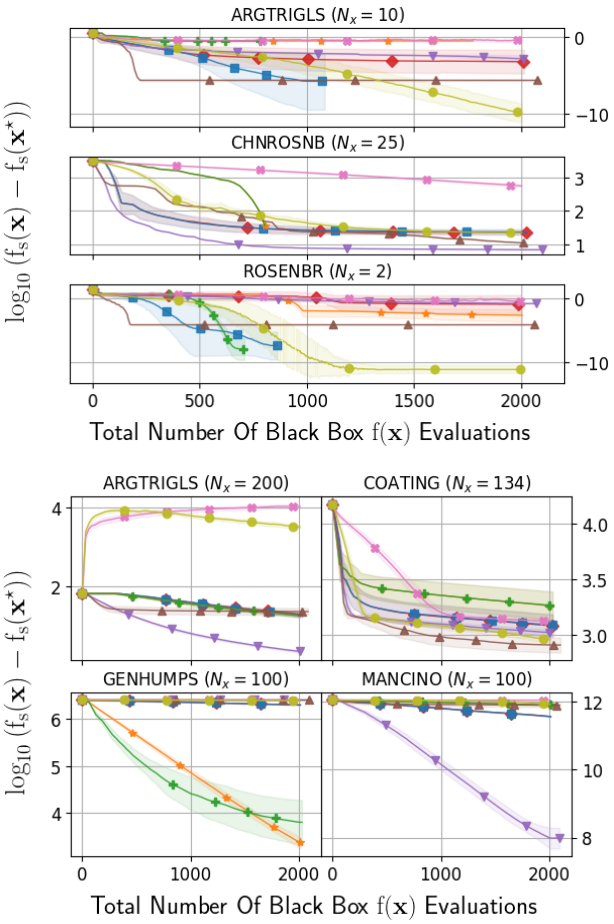

Figure 3. Logarithm of optimality gap vs. number of black box evaluations for low dimensional ($0 < N_x \leq 50$, top half) and medium dimensional ($50 < N_x \leq 200$, bottom half) CUTEst problems with additive uniform random noise. Legend in Figure 2.

Figure 5 shows the evolution of per iteration runtimes for each optimization method for two low dimensional CUTEst problems from Appendix D. A single NNAIF iteration takes about 3 seconds at most (see Appendix E for more analysis).

6. Conclusions

In this paper, we introduced a novel accelerated DFO method for unconstrained optimization problems called NNAIF. NNAIF demonstrates how even crude neural network surrogate models of the objective function $f_s(\mathbf{x})$ can be used to improve the performance of IF optimization methods, while maintaining the established convergence properties of IF methods. In our numerical experiments, NNAIF demonstrates a unique ability to reduce high dimensional structured objective functions by several orders of magni-

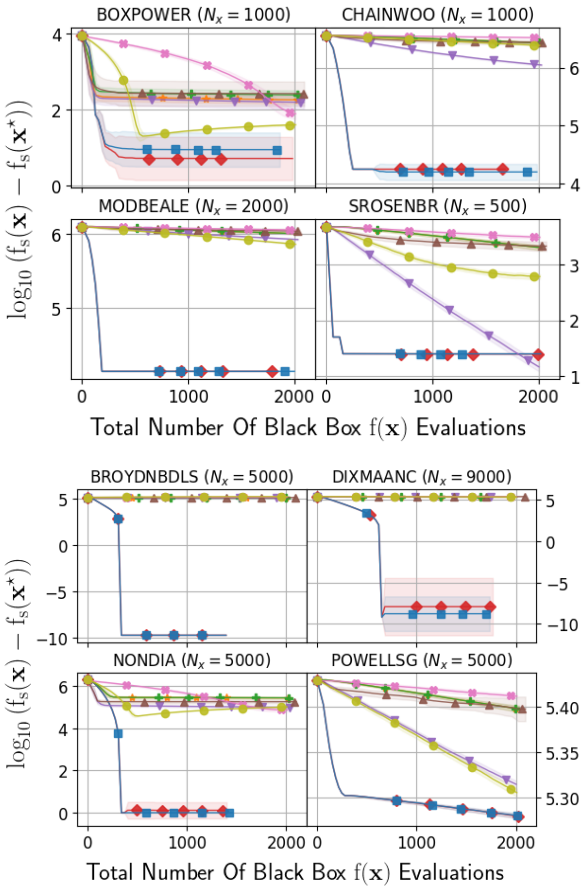

Figure 4. Logarithm of optimality gap vs. number of black box evaluations for high dimensional ($200 < N_x \leq 3000$, top half) and very high dimensional ($3000 < N_x$, bottom half) CUTEst problems with additive uniform random noise. Legend in Figure 2.

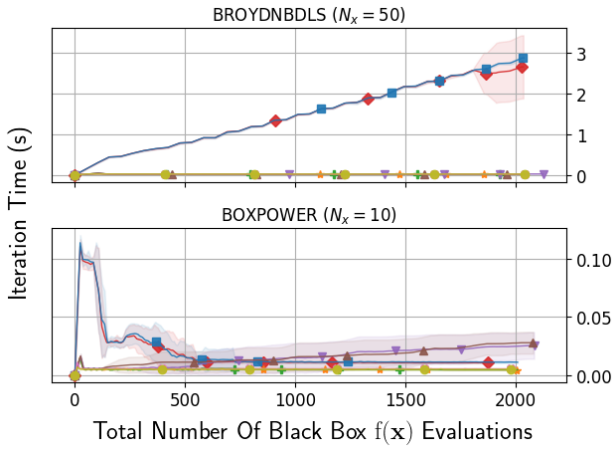

Figure 5. Optimization method iteration time measured in seconds vs. number of black box evaluations for low dimensional ($0 < N_x \leq 50$) CUTEst problems with largest (upper half) and smallest (lower half) maximum iteration time $t_k^{\max}$. NNAIF iteration times increase as the training dataset for the NN surrogate grows, and decrease when $h_k \leq h_{\min}^{\mathrm{surr}}$ and training stops. Legend in Figure 2.

Table 1. Summary of best NNAIF (red diamond or blue square in Figure 2) final results vs. best alternative (i.e. non-NNAIF) final results. The best final mean of $\log_{10}(f_s(\mathbf{x}) - f_s(\mathbf{x}^\star))$ is shown.

CUTEst	NNAIF	Best Alternative	
Figure 3	$\log_{10}(f_s(\mathbf{x}) - f_s(\mathbf{x}^\star))$		Method
ARGTRIGLS	-5.77E0	-9.91E0	CMA-ES
CHNROSNB	1.36E0	8.43E-1	GD
ROSENBR	-7.36E0	-1.11E1	CMA-ES
ARGTRIGLS	1.27E0	3.51E-1	GD
COATING	3.08E0	2.91E0	BFGS
GENHUMPS	6.30E0	3.36E0	IF
MANCINO	1.16E1	7.99E0	GD
Figure 4	$\log_{10}(f_s(\mathbf{x}) - f_s(\mathbf{x}^\star))$		Method
BOXPOWER	7.13E-1	1.60E0	CMA-ES
CHAINWOO	4.20E0	6.05E0	GD
MODBEALE	4.15E0	5.86E0	CMA-ES
SROSENBR	1.39E0	1.16E0	GD
BROYDNBDLS	-9.72E0	5.05E0	GD
DIXMAANC	-8.77E0	5.33E0	GD
NONDIA	-1.86E-8	4.87E0	EMNA
POWELLSG	5.28E0	5.31E0	CMA-ES

tude using only several hundred noisy evaluations of the objective function. As these experiments use NNAIF with a relatively small neural network surrogate with $\approx 4.6 \times 10^5$ parameters or less, and training taking seconds, not minutes, or less per iteration on a GPU, they suggest NNAIF holds the potential to significantly accelerate the solution of DFO problems in diverse areas, such as economics, engineering, and medicine. Developing variants of NNAIF for bound constrained optimization problems typically approached via Bayesian optimization, such as hyperparameter selection, remains an important direction for future work.

Data And Software Availability

At the time of writing, the CUTEst test problems used in the numerical experiments are available at `https://www.cuter.rl.ac.uk/Problems/mastsif.shtml`. Code and documentation for performing experiments is available at `https://github.com/0x4249/NNAIF`.

Acknowledgements

BI and EH acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of British Columbia (UBC). RG and AZ acknowledge the support of International Business Machines Corporation (IBM). The authors also thank the anonymous reviewers and meta-reviewer for their time and comments.

References

Audet, C. and Hare, W. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, 2017. doi: 10.1007/978-3-319-68913-5.

Battermann, A., Gablonsky, J. M., Patrick, A., Kelley, C. T., Kavanagh, K. R., Coffey, T., and Miller, C. T. Solution of a groundwater control problem with implicit filtering. *Optimization and Engineering*, 3(2):189–199, 06 2002. doi: 10.1023/A:1020967403960.

Beyer, H.-G. *The Theory of Evolution Strategies*. Natural Computing Series. Springer Berlin Heidelberg, 2001. ISBN 978-3-662-04378-3. doi: 10.1007/978-3-662-04378-3.

Beyer, H.-G. and Schwefel, H.-P. Evolution strategies – a comprehensive introduction. *Natural computing*, 1(1): 3–52, 2002. doi: 10.1023/A:1015059928466.

Bongartz, I., Conn, A. R., Gould, N. I. M., and Toint, P. L. CUTE: constrained and unconstrained testing environment. *ACM Transcations on Mathematical Software*, 21 (1):123–160, March 1995. doi: 10.1145/200979.201043.

Bortz, D. M. and Kelley, C. T. *The Simplex Gradient and Noisy Optimization Problems*, pp. 77–90. Birkhäuser Boston, Boston, MA, 1998. ISBN 978-1-4612-1780-0. doi: 10.1007/978-1-4612-1780-0_5.

Brochu, E., Cora, V. M., and de Freitas, N. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010. URL https://arxiv.org/abs/1012.2599.

Broyden, C. G. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970. ISSN 0272-4960. doi: 10.1093/imamat/6.1.76.

Choi, T. and Kelley, C. T. Superlinear convergence and implicit filtering. *SIAM Journal on Optimization*, 10(4): 1149–1162, 01 2000. doi: 10.1137/S1052623499354096.

Choi, T., Eslinger, O., Kelley, C., David, J., and Etheridge, M. Optimization of automotive valve train components with implicit filtering. *Optimization and Engineering*, 1 (1):9–27, 06 2000. doi: 10.1023/A:1010071821464.

Conn, A., Scheinberg, K., and Vicente, L. *Introduction to Derivative-Free Optimization*. SIAM, Philadelphia, 2009. doi: 10.1137/1.9780898718768.

Fletcher, R. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 01 1970. ISSN 0010-4620. doi: 10.1093/comjnl/13.3.317.

Fowkes, J., Roberts, L., and Bürmen, A. Pycutest: an open source python package of optimization test problems. *Journal of Open Source Software*, 7(78):4377, 2022. doi: 10.21105/joss.04377.

Frazier, P. I. A tutorial on bayesian optimization, 2018. URL https://arxiv.org/abs/1807.02811.

Gal, R., Haber, E., Irwin, B., Saleh, B., and Ziv, A. How to catch a lion in the desert: on the solution of the coverage directed generation (CDG) problem. *Optimization and Engineering*, 22:217–245, March 2021. doi: 10.1007/s11081-020-09507-w.

Gilmore, P. and Kelley, C. T. An implicit filtering algorithm for optimization of functions with many local minima. *SIAM Journal on Optimization*, 5(2):269–285, 05 1995. doi: 10.1137/0805015.

Goldfarb, D. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24 (109):23–26, 1970. ISSN 00255718, 10886842. doi: 10.2307/2004873.

Gould, N. I. M., Orban, D., and Toint, P. L. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 4 2015. doi: 10.1007/s10589-014-9687-3.

Gould, N. I. M., Orban, D., and contributors. The Constrained and Unconstrained Testing Environment with safe threads (CUTEst) for optimization software. https://github.com/ralna/CUTEst, August 2019.

Haber, E. and Ruthotto, L. Stable architectures for deep neural networks. *Inverse Problems*, 34(1), 2017. doi: 10.1088/1361-6420/aa9a90.

Hansen, N. and Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 06 2001. doi: 10.1162/106365601750190398.

Hansen, N., Arnold, D. V., and Auger, A. *Evolution Strategies*, pp. 871–898. Springer Handbooks. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-43505-2. doi: 10.1007/978-3-662-43505-2_44.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016. doi: 10.1007/978-3-319-46493-0_38.

Kelley, C. *Implicit Filtering*. SIAM, Philadelphia, 2011. doi: 10.1137/1.9781611971903.

Kelley, C. T. *Iterative Methods for Optimization*. SIAM, 1999. doi: 10.1137/1.9781611970920.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. URL http://arxiv.org/abs/1412.6980.

Larson, J., Menickelly, M., and Wild, S. M. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, 2019. doi: 10.1017/S0962492919000060.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pp. 6391–6401, December 2018. doi: 10.5555/3327345.3327535.

Močkus, J. On bayesian methods for seeking the extremum. In Marchuk, G. I. (ed.), *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pp. 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg. ISBN 978-3-540-37497-8. doi: 10.1007/3-540-07165-2_55.

Močkus, J. *Bayesian Approach to Global Optimization: Theory and Applications*, volume 37 of *Mathematics and Its Applications, Soviet Series*. Springer Netherlands, 1989. doi: 10.1007/978-94-009-0909-0.

Nocedal, J. and Wright, S. *Numerical Optimization*. Springer, New York, 2006. doi: 10.1007/978-0-387-40065-5.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Rios, L. M. and Sahinidis, N. V. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56: 1247–1293, 2013. doi: 10.1007/s10898-012-9951-y.

Rosenbrock, H. H. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 01 1960. ISSN 0010-4620. doi: 10.1093/comjnl/3.3.175.

Shan, S. and Wang, G. G. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization*, 41: 219–241, 2010. doi: 10.1007/s00158-009-0420-2.

Shanno, D. F. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24 (111):647–656, 1970. ISSN 00255718, 10886842. doi: 10.2307/2004840.

Stoneking, D., Bilbro, G., Gilmore, P., Trew, R., and Kelley, C. Yield optimization using a GaAs process simulator coupled to a physical device model. *IEEE Transactions on Microwave Theory and Techniques*, 40(7):1353–1363, 07 1992. doi: 10.1109/22.146318.

A. Definition Of Positive Spanning Set

A positive spanning set $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_J] \in \mathbb{R}^{N_x \times J}$ of directions is a set of directions such that any vector $\mathbf{x} \in \mathbb{R}^{N_x}$ can be written as

$$\mathbf{x} = \sum_{j=1}^{J} a_j \mathbf{v}_j = \mathbf{V}\mathbf{a} \tag{22}$$

for some coefficient vector $\mathbf{a} \in \mathbb{R}^J$ where $a_j \geq 0, \forall j \in \{1, \ldots, J\}$. Note that the coefficient vector $\mathbf{a}$ is not necessarily unique.

B. Proof of Theorem 5.7 - Kelley, 2011

Stencil failure implies that

$$\mathbf{V}^{\mathrm{T}} \nabla f_s(\mathbf{x}) \leq \left(\frac{Lh}{2} + \frac{\|\phi\|_{S(\mathbf{x},h,\mathbf{V})}}{h} \right) \mathbf{1}\,, \tag{23}$$

where the inequality in (23) is understood componentwise. Now, let $\mathbf{U}$ be an $N_x \times N_x$ orthogonal matrix such that

$$\mathbf{C} = [\mathbf{U}, -\mathbf{U}]$$

for which the matrix $\mathbf{A}$ with nonnegative entries satisfies $\mathbf{A}\mathbf{V}^{\mathrm{T}} = \mathbf{C}$ and $\kappa(\mathbf{V}) = \sqrt{N_x}\,\|\mathbf{A}\|_\infty$. Since $\mathbf{A}$ has nonnegative entries, we see that (23) implies that

$$\begin{aligned} \mathbf{C}^{\mathrm{T}} \nabla f_s(\mathbf{x}) &= \mathbf{A}\mathbf{V}^{\mathrm{T}} \nabla f_s(\mathbf{x}) && (24)\\ &\leq \mathbf{A}\mathbf{1} \left(\frac{Lh}{2} + \frac{\|\phi\|_{S(\mathbf{x},h,\mathbf{V})}}{h} \right) && (25)\\ &\leq \|\mathbf{A}\|_\infty \left(\frac{Lh}{2} + \frac{\|\phi\|_{S(\mathbf{x},h,\mathbf{V})}}{h} \right) \mathbf{1}\,. && (26) \end{aligned}$$

Since the columns of $\mathbf{C}^{\mathrm{T}} \nabla f_s$ include both $\partial f_s / \partial u_j$ and $-\partial f_s / \partial u_j$ for $1 \leq j \leq N_x$, we have

$$\|\nabla f_s(\mathbf{x})\|_2 \leq \sqrt{N_x}\,\|\mathbf{A}\|_\infty \left(\frac{Lh}{2} + \frac{\|\phi\|_{S(\mathbf{x},h,\mathbf{V})}}{h} \right). \tag{27}$$

C. The CUTEst Optimization Benchmark Set

The Constrained and Unconstrained Testing Environment (CUTE), introduced by Bongartz et al. (1995), provides a versatile environment for testing small and large scale nonlinear optimization algorithms. CUTE provides a collection of optimization test problems, which we refer to as CUTE

problems. CUTE also provides a problem classification system designed to help the user identify CUTE problems from a particular class of optimization problem. The CUTE problem classification system takes into account the problem objective function, the problem constraints, the smoothness of the problem, and the origin of the problem. At the time of writing, a specification of the CUTE problem classification system can be found at https://www.cuter.rl.ac.uk/Problems/classification.shtml.

The most up to date descendant of CUTE is the Constrained and Unconstrained Testing Environment with safe threads (CUTEst), introduced in Gould et al. (2015). CUTEst uses the same problem classification system as CUTE, and the same simple input format (SIF) file format for storing the optimization test problems. At the time of writing, SIF files and descriptions of all CUTEst problems can be found at https://www.cuter.rl.ac.uk/Problems/mastsif.shtml. Many of the CUTEst optimization test problems have options defined in their SIF file that change the number of variables $\mathbf{x}$. Thus, CUTEst problems can range from dimension $N_x = 1$ to $N_x = 10^5$.

D. Extended Numerical Experiments

Figure 7 presents the results of additional numerical experiments with low and medium dimensional CUTEst problems, and Figure 8 presents the results of additional numerical experiments with high and very high dimensional CUTEst problems. Table 2 summarizes the best final results from Figures 7 and 8.

E. Optimization Method Empirical Runtimes

Figure 5 in Section 5 and Figure 6 to the right illustrate how the per iteration runtimes for each optimization method evolve as each method progresses. Note that an upper bound on the number of possible iterations of an optimization method given a fixed budget of black box $f(\mathbf{x})$ evaluations (denote this fixed budget by $\mathrm{B}_f \geq 0$) is given by B_f/J, where J is the minimum number of $f(\mathbf{x})$ evaluations required by a single iteration of the optimization method. Letting $t_k^{\max}$ be the slowest iteration time, a conservative upper bound on the total running time t_{total} is given by

$$t_{\text{total}} \leq \left(\frac{\mathrm{B}_f}{J}\right) t_k^{\max} . \tag{28}$$

Using the iteration time data for all low dimensional problems in Figures 3 and 7, and that $J = N_x + 1$ for low dimensional problems, (28) suggests that all NNAIF computations should take less than 2 minutes total (i.e. $t_{\text{total}} \leq 120$ seconds). Similarly, for the medium, high, and very high dimensional problems in Figures 3, 4, 7, and 8, using that $J = 30$, (28) suggests that all NNAIF computations should take less than 4 minutes total (i.e. $t_{\text{total}} \leq 240$ seconds).

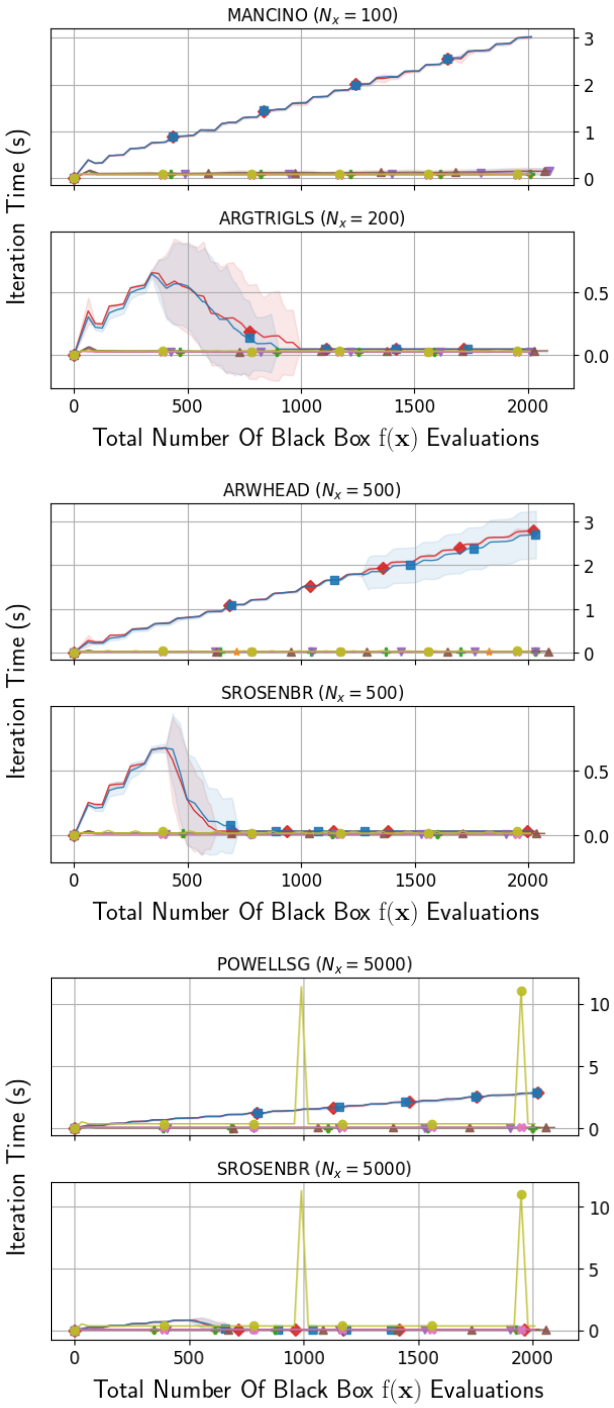

Figure 6. Optimization method iteration time measured in seconds vs. number of black box evaluations for medium dimensional ($50 < N_x \leq 200$, top third), high dimensional ($200 < N_x \leq 3000$, middle third), and very high dimensional ($3000 < N_x$, bottom third) CUTEst problems with largest (upper half of each third) and smallest (lower half of each third) maximum iteration time $t_k^{\max}$. Legend provided in Figure 2.

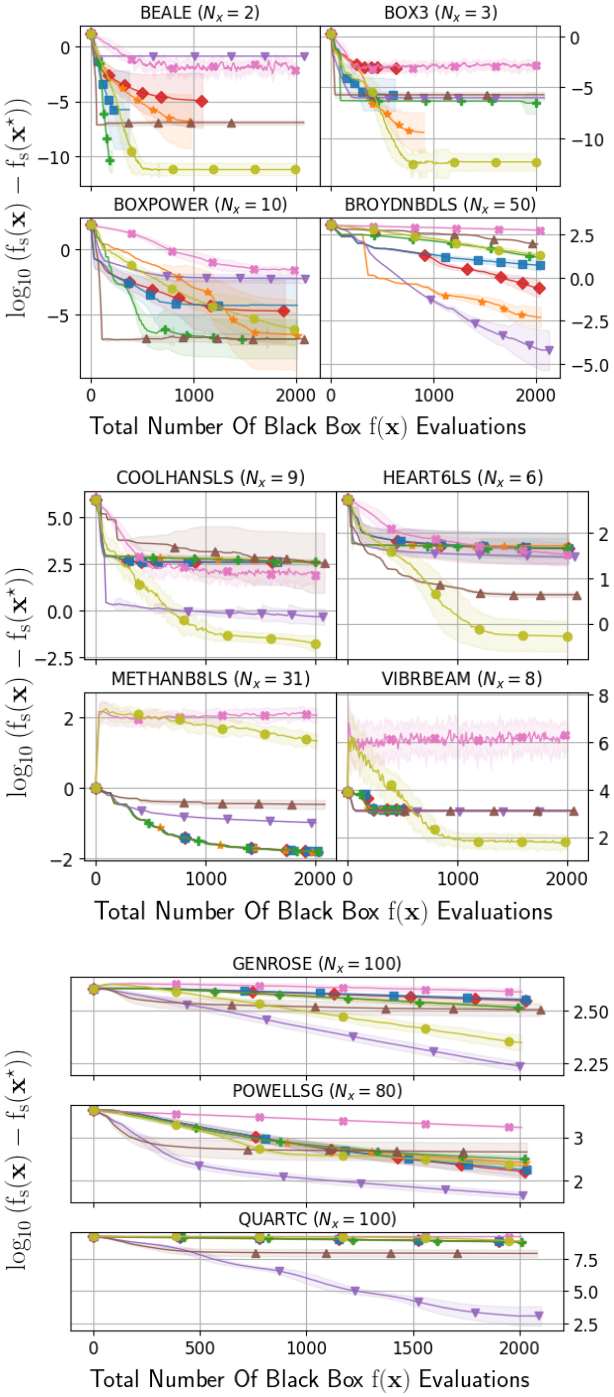

Figure 7. Base 10 logarithm of optimality gap vs. number of black box $f(\mathbf{x})$ evaluations for additional low dimensional ($0 < N_x \leq 50$, top two thirds) and medium dimensional ($50 < N_x \leq 200$, bottom third) CUTEst problems with uniform random noise added. Legend provided in Figure 2.

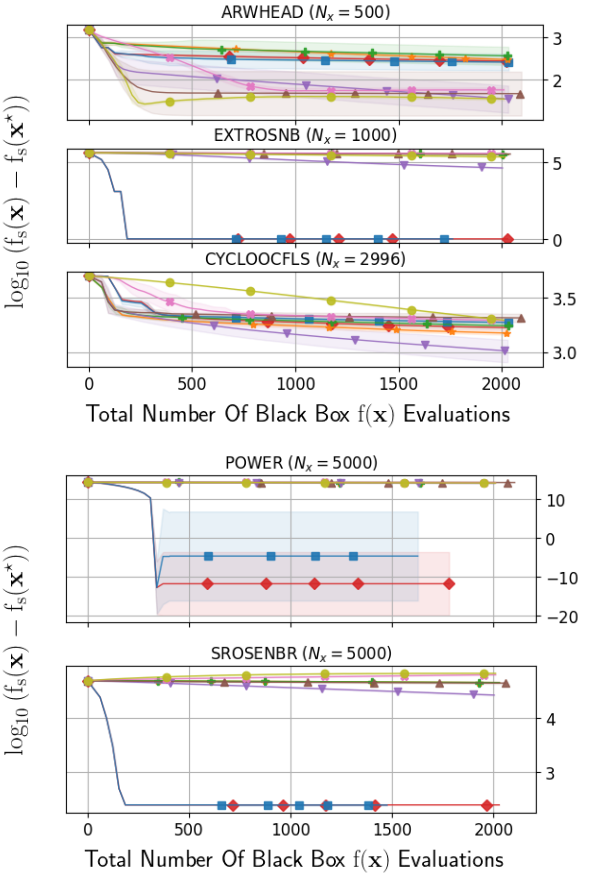

Figure 8. Base 10 logarithm of optimality gap vs. number of black box $f(\mathbf{x})$ evaluations for additional high dimensional ($200 < N_x \leq 3000$, top half) and very high dimensional ($3000 < N_x$, bottom half) CUTEst problems with uniform random noise added. Legend provided in Figure 2.

Table 2. Summary of best NNAIF (red diamond or blue square in Figure 2) final results vs. best alternative (i.e. non-NNAIF) final results. The best final mean of $\log_{10}(f_s(\mathbf{x}) - f_s(\mathbf{x}^\star))$ is shown.

CUTEst	NNAIF	Best Alternative	
Figure 7	$\log_{10}(f_s(\mathbf{x}) - f_s(\mathbf{x}^\star))$		Method
BEALE	-5.76E0	-1.12E1	CMA-ES
BOX3	-5.74E0	-1.23E1	CMA-ES
BOXPOWER	-4.73E0	-6.88E0	IMFIL
BROYDNBDLS	-6.05E-1	-4.19E0	GD
COOLHANSLS	2.60E0	-1.75E0	CMA-ES
HEART6LS	1.65E0	-2.84E-1	CMA-ES
METHANB8LS	-1.78E0	-1.80E0	IMFIL
VIBRBEAM	3.15E0	1.81E0	CMA-ES
GENROSE	2.55E0	2.23E0	GD
POWELLSG	2.20E0	1.66E0	GD
QUARTC	8.87E0	3.06E0	GD
Figure 8	$\log_{10}(f_s(\mathbf{x}) - f_s(\mathbf{x}^\star))$		Method
ARWHEAD	2.40E0	1.53E0	GD
EXTROSNB	-4.10E-5	4.61E0	GD
CYCLOOCFLS	3.22E0	3.01E0	GD
POWER	-1.17E1	1.39E1	GD
SROSENBR	2.40E0	4.42E0	GD