NORTHWESTERN UNIVERSITY

Methods for Derivative-Free Optimization with Applications in Machine Learning

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

By

Melody Qiming Xuan

EVANSTON, ILLINOIS

September 2023

ABSTRACT

Methods for Derivative-Free Optimization with Applications in Machine Learning

Melody Qiming Xuan

Derivative-free optimization (DFO) has received growing attention due to important problems arising in practice. Various research communities, ranging from machine learning to engineering design, have adopted distinct DFO methods. In this thesis, we present extensive studies as a meaningful step towards a comprehensive understanding of DFO methods. We study the relative merits of this disparate class of methods in a controlled setting, where the derivatives of the functions are unavailable and the functions values are subject to noise.

In Chapter 2, we provide numerical investigations of finite-difference-based approaches against interpolation-based methods for solving unconstrained, (nonlinear) least-square and constrained optimization problems. This approach relies on finite differences to perform gradient approximations and applies existing gradient-based nonlinear optimization solvers. The numerical results demonstrate the effectiveness of finite-difference-based methods for both deterministic and noisy problems, when a proper implementation of finite differences is employed. The empirical results suggest that finite-difference-based

methods are competitive with IBO methods, which are among the most reliable and efficient DFO solvers.

We present in Chapter 3 an adaptive procedure to choosing a reliable finite differencing interval for noisy DFO problems, using a bisection approach. When noise is present in the objective function, the finite differencing interval needs to be chosen meticulously to balance the truncation error and noise. In particular, the optimal interval depends on the noise level of the function and a bound on the higher-order derivative. We show that the proposed adaptive procedure produces near-optimal differencing intervals without explicitly approximating higher-order derivatives. We demonstrate its robustness and accuracy on selected problems.

In Chapter 4, we investigate general constrained DFO, where the constraints are analytically available. We propose to extend unconstrained interpolation-based optimization methods by combining them with a general-purpose nonlinear programming solver. This is a feasible method that constructs quadratic models of the objective and generates feasible iterates at every iteration. We discuss its effectiveness and limitations in this chapter. Our numerical results suggest that, when the objective is expensive to evaluate and the cost for evaluating constraints is negligible, this seemingly expensive approach is competitive with a state-of-the-art constrained DFO solvers that employs finite differences.

Next, in Chapter 5 we consider a DFO problem that arises in Natural Language Processing. With the wide success of pre-trained large language models (LLMs) and their growing model size, it is imperative to consider strategies that efficiently adapt pre-trained general LLMs to individual language tasks. In particular, we consider prompt design, which makes use of different prompts to condition LLMs for different tasks. We propose

to treat prompt design as a noisy DFO problem, wherein we search for a prompt that optimizes performance without accessing the parameters of the underlying LLM. We present competitive results for solving prompt design using DFO methods and identify the relative merits of NEWUOA, FD L-BFGS and CMA-ES.

The observations in Chapter 5 motivate us to further analyze the relative strengths of NEWUOA and CMA-ES in Chapter 6. In particular, we consider a broader range of unconstrained problems from the CUTEst problem set. We observe that, contrary to the noiseless problems, while NEWUOA still demonstrates greater efficiency in achieving a desired level of accuracy, it is adversely affected by noise in function evaluations and consequently achieves lower accuracy compared to CMA-ES. Therefore, we investigate the impact of noise on IBO methods and briefly discuss strategies to improve the final accuracy of IBO methods in the presence of noise.

Acknowledgements

I am deeply thankful to all those whose support and contributions have made this dissertation possible.

First and foremost, I would like extend my sincerest gratitude to my advisor, Jorge Nocedal, for his unwavering support and guidance throughout my Ph.D. journey. I consider myself immensely fortunate to have you as my advisor, mentor, friend, and someone I will always look up to. Thank you for providing me with all the opportunities, insights and guidance, and for your compassion, patience and understanding. I never could imagine having a better advisor in this academic journey.

I could not have completed my work without the efforts of many inspiring researchers with whom I have interacted and learned from. I would like to extend my sincere thanks to Andreas Waechter and Frank E. Curtis for serving on my thesis committee. I am especially grateful for their feedback and insights into various research topics. I would also like to express my gratitude to Ermin Wei and Richard Byrd for providing many insightful comments. I also had the privilege of collaborating with many exceptional researchers, including Figen, Michael and Yuchen.

I would like to thank the IEMS faculty for all the inspiring courses and conversations. I am also grateful to the IEMS staff, including Agnes, Brittany, Colleen, Kendall, Johnathan, and Stephen for helping me with many matters. I am also thankful to my IEMS first-year cohort and friends for all the cherished moments and support.

I am profoundly grateful to the members of the Optimization Lab at L375: Albert, Alejandra, Michael, Raghu, Ruby, Shigeng, Shima, Xiaochun, Xinyi and Yuchen. Thank you for all the inspiring discussions, support, endless conversations and all the great times we have had.

Finally, I would like to extend my heartfelt gratitude to my family and friends. Mom and Dad, thank you for your unconditional love and support as always. Mandy, thank you for being such a sweet little sister who brings all the laughters. Oliver, thank you so much for being in my life, for always supporting me and seeing the best in me, for all the memories we have created together and for the much more to come. To my most adorable furry family member Doodle, thank you for showing up in my life four years ago and filling my days with boundless joy - although you have no idea what I am writing about. To all my friends, especially Huiming, thank you for your encouragement and sharing many great memories with me.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

The problem of minimizing a smooth function when its derivatives are not available has been widely studied in the literature [29, 59, 69]. A variety of methods have been proposed to solve general unconstrained derivative-free optimization (DFO) problems, with some being extended to nonlinear least squares and general constrained problems. Various research communities have indeed adopted distinct DFO methods to solve important problems arising from machine learning to engineering design. The objective function in those applications is often expensive to evaluate and may be perturbed with noise, e.g., computational noise, while DFO methods are typically designed for noiseless problems. In this thesis, we study several disparate classes of DFO methods in a controlled setting for different classes of problems.

Let us consider a problem of the form

$$\min_{x \in \Omega} \quad \phi(x) \tag{1.0.1}$$

where $\phi : \mathbb{R}^n \to \mathbb{R}$ is a smooth function and $\Omega \subseteq \mathbb{R}^n$ denotes the feasible set. The feasible set Ω is specified by a set of equality and inequality constraints:

$$\Omega = \{x \in \mathbb{R}^n | \psi_i(x) = 0, \forall i \in \mathcal{E}; \psi_j(x) \leq 0, \forall j \in \mathcal{I}\}, \tag{1.0.2}$$

where $\psi : \mathbb{R}^n \to \mathbb{R}^m$ is smooth, and $\mathcal{E}$ and $\mathcal{I}$ are finite index sets that can be empty (the problem reduces to unconstrained optimization if $\mathcal{E} = \mathcal{I} = \emptyset$). We assume that the objective values $\phi(x)$ are only accessible through a (noisy) zero-order oracle such that the derivatives of ϕ are not available. Following the taxonomy of constraints [35], we assume that the constraints can be analytically available with access to their derivatives (class A) or only accessible via a (noisy) zero-order oracle (class S).

Formally, we define the noisy zero-order oracles of the objective and constraints as:

$$f(x) = \phi(x) + \epsilon(x) \tag{1.0.3}$$

$$c_i(x) = \psi_i(x) + \epsilon_i(x) \tag{1.0.4}$$

where $\epsilon(\cdot)$ and $\epsilon_i(\cdot)$ model the noise, which can be either deterministic or stochastic. In practice, such evaluation errors can stem from finite precision arithmetics as well as stochastic optimization problems encountered in machine learning applications. When the noise is uniformly bounded, we refer to ϵ_f and ϵ_c as the noise levels of the functions respectively, where

$$|\epsilon(x)| \leq \epsilon_f, \quad |\epsilon_i| \leq \epsilon_c. \tag{1.0.5}$$

In the case when the noise terms are stochastic, we denote the standard deviations σ_f, σ_c of the noise terms as the noise levels of the objective and constraints respectively. The noise level can be estimated via sampling (stochastic) and/or the difference table (deterministic) [70]. Thus we assume throughout this thesis that the noise levels of the functions are known.

Several approaches have been developed for solving problem (1.0.1), including gradient approximation based methods, interpolation-based optimization (IBO) methods and randomized algorithms. In Chapter 2, we consider unconstrained, nonlinear least-squares and constrained optimization problems that can be only accessed via a noisy zeroth-order oracle with stochastic noise. We provide a sophisticated adaptive finite-difference differencing interval selection scheme in Chapter 3 for bounded noise. In Chapter 4, we study (1.0.1) where both $\mathcal{E}$ and $\mathcal{I}$ can be nonempty and the constraints are available in analytical forms. In Chapter 5, we present an application of noisy unconstrained DFO problem arising in large language models. In Chapter 6 we consider unconstrained DFO problems again with noisy evaluations to study the behavior of DFO methods and discuss potential strategies for improving the final accuracy of interpolation-based optimization methods.

In the remainder of this chapter, we provide an overview of several classes of methods that can be applied to solve (1.0.1) and discuss their advantages and disadvantages, respectively. For methods that are not covered in this thesis, such as direct search and deterministic global optimization methods, see [59].

1.1. Gradient Approximations

One simple yet effective approach to solving (1.0.1) is to perform approximations to the derivatives and apply derivative-based optimization solvers. Since the idea remains the same for both the objective and constraints, we will describe gradient approximation based on $f(x)$. Overall, such approach computes an estimate $g(x)$ of the gradient $\nabla\phi(x)$

via forward differences (FD)

$$g(x) = \sum_{i=1}^{N} \frac{f(x + hu_i) - f(x)}{h} u_i \tag{1.1.1}$$

or central differences (CD)

$$g(x) = \sum_{i=1}^{N} \frac{f(x + hu_i) - f(x - hu_i)}{2h} u_i \tag{1.1.2}$$

where $\{u_1, u_2, ..., u_N\}$ is a set of directions and h is the differencing interval or sampling radius [8]. We describe two strategies of constructing gradient approximations below.

1.1.1. Finite Differences

For standard finite differences, (1.1.1) and (1.1.2) reduce to

$$[g(x)]_i = \frac{f(x + h_i e_i) - f(x)}{h_i} \quad \text{(FD)} \tag{1.1.3}$$

$$[g(x)]_i = \frac{f(x + h_i e_i) - f(x - he_i)}{2h_i} \quad \text{(CD)} \tag{1.1.4}$$

respectively for $i = 1, ..., n$, with $N = n$ and $u_i = e_i$ where e_i denotes the unit vector along the i-th coordinate.

When no noise is present in the function ϕ with $\epsilon(x) \equiv 0$, the estimated gradient $g(x)$ of $\nabla\phi(x)$ can be computed by either (1.1.3) or (1.1.4) using

$$h_i = \max(1, |x_i|)\sqrt{\epsilon_M} \quad \text{(FD)} \tag{1.1.5}$$

$$h_i = \max(1, |x_i|)\sqrt[3]{\epsilon_M} \quad \text{(CD)} \tag{1.1.6}$$

where ϵ_M denotes the machine precision. Such choice of differencing is designed to handle roundoff errors and is common in practice [7]. The central differences scheme (1.1.4) requires more function evaluations but provides a more accurate estimate of the gradient than the forward differences (1.1.3) scheme.

However, in the presence of noise in the function, the choice of the finite differencing interval can be delicate. To see this, consider the case of applying forward differences to a function with bounded noise. By Taylor expansion and (1.0.3), we have

$$\mathbb{E}[([g(x)]_i - [\nabla\phi(x)]_i)^2] \leq \frac{L_i^2 h_i^2}{4} + \frac{2\epsilon_f^2}{h_i^2} \tag{1.1.7}$$

where L_i is an upper bound for $|e_i^T \nabla^2 \phi(x) e_i|$ [71]. Therefore, the choice of the differencing interval h_i must carefully balance the truncation error and the noise. If h_i is chosen too large, the truncation error resulting from the first term on the right-hand side of (1.1.7) damages the accuracy of the gradient approximation. On the other hand, if h_i is too small, the noise term dominates the approximation error. A near-optimal choice of the forward differencing interval can be obtained by minimizing the right-hand side of (1.1.7). With the noise level ϵ_f and bound of second derivative L_i in hand, we can obtain

$$h_i = \sqrt[4]{8}\sqrt{\frac{\epsilon_f}{L_i}} \tag{1.1.8}$$

Similarly, one can compute the expected error on the central differences case and the resulting differencing interval is

$$h_i = \sqrt[3]{\frac{3\epsilon_f}{M_i}} \tag{1.1.9}$$

where M_i is the bound on the third derivative along the i-th coordinate direction.

Since the bounds on the second and third derivative are not available, one can estimate them by employing the procedure proposed by Gill et. al or Moré and Wild [40, 71] in practice. A more robust procedure that avoids direct estimation of the bounds is proposed in [92].

Finite differences enjoy two appealing features as they can be incorporated into existing derivative-based optimization solvers and can be easily parallelized in the function evaluations. However, when noise is present, one must carefully choose the differencing interval, which requires knowledge of the noise level and bounds on the higher-order derivatives. Kelly developed an implicit filtering method with diminishing differencing intervals, assuming that the noise diminishes as the iterate approaches the solution [21, 55]. Berahas et al. proposed a finite-difference-based L-BFGS method for solving problems with non-diminishing noise, combining strategies for noise level and bound on higher-order derivatives [7]. Such requirements call for a sophisticated and robust procedure to further improve the performance of finite-difference-based methods.

1.1.2. Gaussian Smoothing

An alternative for approximating the gradient is to employ samplings along random directions, with the most popular choice being Gaussian directions [73]. Such approach is referred to as Gaussian Smoothing and has been proved efficient for applications in reinforcement learning [22, 88]. In particular, it considers (1.1.3) or (1.1.4) by sampling standard Gaussian directions $u_i \sim \mathcal{N}(0, I)$ for $i = 1, ..., N$.

To motivate such choice, consider the Gaussian smoothed version of function f:

$$f_h(x) = \mathbb{E}_{u\sim\mathcal{N}(0,I)}[f(x+hu)] = \int_{\mathbb{R}^n} f(x+hu)\pi(u|0,I)du$$

where $\pi(u|0,I)$ denotes the density function of the standard Gaussian distribution evaluated at u. It can be shown that, the gradient of smoothed function $F(x)$ can be computed as

$$\nabla f_h(x) = \frac{1}{h}\mathbb{E}_{u\sim\mathcal{N}(0,I)}[f(x+hu)u] = \frac{1}{h}\mathbb{E}_{u\sim\mathcal{N}(0,I)}[(f(x+hu)-f(x))u]$$

Therefore, by employing the idea of sample average approximations, one can obtain (1.1.3) for the forward difference case. We can verify that, when noise is not present, it follows that for $h_i \to 0$,

$$\mathbb{E}_{u\sim\mathcal{N}(0,I)}[g(x)] = \nabla\phi(x).$$

When the function evaluation is noisy, similar as in the finite differences case, there are two sources of error. Because of the Gaussian distributed directions, $[g(x)]_i$ can be arbitrarily large. Additionally, the performance of this approach is sensitive to the choice of the sampling radius h. As illustrated in [9], the randomized gradient approximation can perform significantly worse than the deterministic methods. See [8, 9] for further discussions on gradient approximations.

1.2. Interpolation-Based Optimization (IBO) Methods

Interpolation-based optimization (IBO) methods have been extensively studied for unconstrained DFO problems due to their robustness and efficiency; see [29] for a thorough description. IBO methods typically construct a quadratic model of the objective via linear

interpolation and generate the next iterate using a trust-region framework. While finite differences require at least $n+1$ function evaluations at every iteration, IBO methods can make progress with only one function evaluation. However, it demands higher algebra cost due to interpolation. We provide a basic description of a general unconstrained IBO method in Algorithm 1.1, conceding that the details may vary for different IBO methods.

At each iteration k, IBO methods construct a quadratic model of the objective around the current iterate x_k

$$m_k(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T H_k s, \tag{1.2.1}$$

where g_k and H_k are computed by interpolating points in the poised interpolation set $Y_k = \{y_0, y_1, y_2, ..., y_m\} \subseteq \mathbb{R}^n$:

$$f(y_i) = m(y_i), \quad i = 0, 1, ..., m$$

This can be done uniquely with a cost of $\mathcal{O}(n^6)$ when $m = (n+1)(n+2)/2$. The cost can be reduced to $O(m^2)$ by employing a minimum Frobenius norm update of the Hessian approximation [78].

IBO methods aim to approximate the objective within a neighborhood of the current iterate x_k

$$B(x_k, \Delta_k) = \{x \in \mathbb{R}^n | \|x - x_k\|_2 \leq \Delta_k\}, \tag{1.2.2}$$

where Δ_k is referred to as the trust region radius. Note that other norms could also be used in (1.2.2). For unconstrained optimization, a trust region subproblem is solved at

each iteration to obtain a candidate step s_k for the next iterate

$$\min_{s \in B(0,\Delta_k)} m_k(x_k + s). \tag{1.2.3}$$

By evaluating the ratio test

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}, \tag{1.2.4}$$

the algorithm accepts the step s_k if ρ_k is greater than or equal to a nonnegative user-defined value. The next iterate is then defined as $x_k + s_k$ and the trust region Δ_k may be increased. Otherwise, the method rejects the step and checks whether the interpolation set Y_k is sufficiently poised. The interpolation set Y_k is maintained and updated at every iteration to reflect the geometry of the objective around the current iterate x_k. It is required to satisfy certain geometry conditions to ensure quality of the interpolated. If Y_k is not well-poised, the algorithm improves the interpolation set for better model quality and leaves Δ_k unchanged to avoid premature shrinkage of the trust region. If the model is sufficiently accurate, the algorithm follows classical trust region methods by reducing the trust region [29].

Algorithm 1.1. IBO algorithm framework for solving unconstrained DFO problems.

Parameters: $0 \leq \eta < 1$, and $\Delta_0 > 0$.
Choose x_0, a starting point, and construct an initial poised interpolation set Y_0.
Let $k = 0$.
while no convergence test is satisfied **do**
    Build a local (quadratic) model (1.2.1) using interpolation set Y_k.
    Compute a step s_k by solving the trust region subproblem (1.2.3).
    Compute the ratio defined by (1.2.4).
    if $\rho_k \geq \eta$ **then**
        Set $x_{k+1} = x_k + s_k$.
        Choose $\Delta_{k+1} \geq \Delta_k$.
        Update Y_k to include x_{k+1}.
    else if Y_k needs to be improved **then**
        Set $x_{k+1} = x_k$.
        Set $\Delta_{k+1} = \Delta_k$.
        Improve Y_k using a geometry-improving procedure.
    else
        Set $x_{k+1} = x_k$.
        Choose $\Delta_{k+1} < \Delta_k$.
        Update Y_k to include $x_k + s_k$.
    end if
    $Y_{k+1} = Y_k$
    $k = k + 1$
end while

Unconstrained IBO methods have been extended by Powell to solve bound constrained [79], linear constrained [76, 80] and inequality constrained problems [77]. When the constraints are convex and the projection operator is known, [24, 51] propose to enforce the constraints in the trust region subproblem (1.2.3) and establish convergence analysis. A few works have been proposed to handle general (potentially nonconvex) constraints [11, 23, 26, 83]. However, no convergence result is known for general constrained IBO methods.

Surprisingly, as demonstrated in [93], IBO methods can be directly applied to noisy functions without modifications to their noiseless counterparts and achieve competent performance. Some noisy variants of IBO methods handle unconstrained stochastic DFO problems by maintaining model accuracy with high probability [17, 20]. Other works explore different strategies for extending unconstrained IBO to noisy problems, including [18, 99], in which the ratio test is modified or restarts are employed.

1.3. Covariance Matrix Adaptation - Evolution Strategy (CMA-ES)

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a randomized DFO method that conducts iterative sampling based on a multivariate Gaussian distribution with adaptive parameters. Many variants of this approach have been proposed to adopt different sampling and parameter update strategies, although the main idea remains the same [46].

To elaborate, at each iteration k, the algorithm `CMA-ES` samples new candidate solutions from a multivariate Gaussian distribution $\mathcal{N}(m_k, \sigma_k^2 C_k)$, where σ_k is the stepsize, m_k is the mean vector and C_k is the covariance matrix. In contrast to Gaussian Smoothing,

in which the sampling distribution is fixed throughout the optimization procedure, this algorithm updates the parameters of the Gaussian distribution using information from past iterations: 1) the mean vector is computed as a weighted average of the best samples in the current iteration 2) the covariance matrix is modified via low-rank updates such that the variance along promising directions (i.e., directions that are more likely to reduce the objective) is increased 3) the stepsize σ_k is updated based on the cumulative path from past iterations to facilitate convergence. We provide a formal algorithm statement for basic `CMA-ES` in Algorithm 1.2.

After sampling λ new candidates $\{x_1, ..., x_\lambda\}$ from the given Gaussian distribution $\mathcal{N}(m_k, \sigma_k^2 C_k)$ at iteration k, the algorithm evaluate at those points and rank them such that $f(x_{(1)}) \leq f(x_{(2)}) \leq ... \leq f(x_{(\lambda)})$. The new mean m_{k+1} is then moved to a weighted average of the best μ sampled points at the current iteration

$$m_{k+1} = \sum_{i=1}^{\lambda} w_i x_{(i)} = m_{(k)} + \sum_{i=1}^{\lambda} w_i (x_{(i)} - m^{(k)}). \tag{1.3.1}$$

where w_i are pre-specified weights such that $\sum_{i=1}^{\mu} w_i = 1, w_1 \geq w_2 \geq ... \geq w_\mu > 0$.

Since reestimating the covariance matrix from scratch using $\{x_1, ..., x_\lambda\}$ can be unstable, the algorithm performs a rank-μ update that exploits information from all past samples, and a rank-one update, which captures the correlation between consecutive steps in the past. The scale of the distribution depends on the stepsize σ_k, which depends on the cumulative path in the past. Intuitively, the length of the cumulative path is shorter if the past steps cancel out each and longer if they point at similar directions.

Algorithm 1.2. (μ, λ) CMA-ES

Parameters: $c_\sigma, d_\sigma, c_c, c_1, c_\mu$ (parameters for updates); $\lambda \geq \mu > 0$ (parameters for sampling); $w_i : \sum_{i=1}^{\mu} w_i = 1, w_1 \geq w_2 \geq ... \geq w_\mu > 0$ (weight parameters)

Initialization: $m_0 \in \mathbb{R}^n, C_0 = I$, $\sigma_0 > 0$; $p_{\sigma,0} = 0, p_{c,0} = 0$; $k = 0$

while Not Terminated **do**

    Sampling:

        Eigendecomposition: $C_k = BD^2B^T$

        Sample λ points $z_j \sim \mathcal{N}(0, I)$ for $j = 1, ..., \lambda$

        $y_j = BDz_k \sim \mathcal{N}(0, C_k)$

        $x_j = m_k + \sigma_k y_j \sim \mathcal{N}(m_k, \sigma_k^2 C_k)$

    Rank and Recombination:

        Evaluate $f(x_1), ..., f(x_\lambda)$

        Select the best μ points: $x_{(1)}, ..., x_{(\mu)}$ such that $f(x_{(1)}) \leq f(x_{(2)}) \leq ... \leq f(x_{(\lambda)})$

        Let $\bar{y}_w = \sum_{i=1}^{\mu} w_i y_{(i)}$ where $y_{(i)}$ corresponds to $x_{(i)}$

    Update parameters

        Update m:

            $m_{k+1} = m_k + \sum_{i=1}^{\lambda} w_i (x_{(i)} - m^{(k)})$

        Update stepsize:

            $p_{\sigma,k+1} = (1 - c_\sigma) p_{\sigma,k} + \sqrt{c_\sigma (2 - c_\sigma) \mu_{\text{eff}}} (C_k)^{\frac{1}{2}} \bar{y}_w$

            $\sigma_{k+1} = \sigma_k \exp\{\frac{c_\sigma}{d_\sigma} (\frac{\|p_{\sigma,k+1}\|}{\mathbb{E}[\|\mathcal{N}(0,I)\|]} - 1)\}$

        Update Covariance Matrix:

            $p_{c,k+1} = (1 - c_c) p_{\sigma,k} + h_\sigma \sqrt{c_c (2 - c_c) \mu_{\text{eff}}} \bar{y}_w$

            $C_{k+1} = (1 - c_1 - c_\mu) C_k + c_1 p_{c,k+1} p_{c,k+1}^T + c_\mu \sum_{i=1}^{\mu} w_i y_{(i)} y_{(i)}^T$

    $k = k + 1$

end while

The covariance matrix C_k is designed to learn second order information of the objective, although no analysis has been established [45]. This method has been widely applied to practical applications such as Reinforcement Learning [98] and hyperparameter tuning [64]. The method does not involve any approximations to the gradient of the objective function and relies solely on rankings to perform updates. CMA-ES enjoys the benefit of having few algorithm parameters and is believed to work well on difficult problems that are ill-conditioned, require many function evaluations to solve and have relatively high dimensionality. It has been found to outperform 31 DFO methods on a collection of black-box functions [47], including IBO methods. However, when only a small number of function evaluations are available, `CMA-ES` can be inferior to other methods such as IBO. Convergence analysis has been established on a modified version of `CMA-ES` [36].

1.4. Bayesian Optimization

Bayesian Optimization (BO) methods have been proposed to solve DFO problems with expensive objective evaluations and simple constraints. It has been popular for hyperparameter tuning in the machine learning community. See [39] for a complete overview for BO methods.

The basic idea of BO consists of two steps: i) modeling the objective function and ii) optimizing the acquisition function to obtain the next iterate. BO methods employ Gaussian Process (GP) regression to build a surrogate model for the objective function and quantify the uncertainty in the function. Given a set of evaluated points, GP assumes that the objective values are drawn from some multivariate Gaussian distribution with a mean vector and covariance matrix. Utilizing the properties of Gaussian distribution, one can

compute a conditional distribution, or in the nomenclature of BO, posterior probability distribution, based on all available data for a new point that hasn't been evaluated. The mean vector and kernel for the covariance matrix are hyperparameters for BO and remains flexible as long as several properties are satisfied; see [39]. The acquisition function is then defined based on the poster distribution, with careful balance between exploration and exploitation. Common choices of acquisition functions include expected improvement, upper confidence bound and knowledge gradient [39]. The general idea indicates that the next iterate should either have a better expected objective (Exploitation) or high variance (Exploration). This is important for BO as the method is designed for global optimization.

When the objective contains stochastic Gaussian noise, BO can be easily adapted by incorporating such information in the kernel function. However, BO methods are only efficient for problems with small number of variables (up to 20) as it scales poorly with the dimension due to the GP component. The performance of the methods also relies on proper choice of the GP component, e.g., picking a good kernel, and can be volatile.

1.5. An Application: Prompting Large Language Models

DFO problems arise in many applications from engineering to machine learning, especially when the objective results from black-box simulations or model evaluation. Examples include hyperparameter tuning [58], adversarial attacks [19], traffic model calibration [4], parameter estimation [103] to name a few.

One recent application comes from the field of Natural Language Processing, where a general pre-trained large language model (LLM) needs to be adapted to address individual language tasks. Given the massive size of the LLMs, one approach that has gained

increasing attention is to append a prompt, e.g., task description, to every input sequence, while keeping the language model parameters frozen. This approach is appealing as it eliminates the need for storing separate copies of the model parameters for each task. Unfortunately, this prompt design process is often done in a manual trial-and-error manner, yielding results that may not be as effective as model tuning. To address this, one strategy is to append a continuous prompt p to every input sequences x_i and perform optimization on this continuous vector without changing parameters of the massive language model.

Suppose we wish to classify the correct label y_i for each input sequence x_i using loss function $\mathcal{L}$. The prompt optimization problem can thus be formulated as

$$\min_{p \in \mathbb{R}^D} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(p; x_i), y_i) \tag{1.5.1}$$

where f is the pre-trained language model and N the number of training samples. With the massive size of the LLMs, computing the derivatives of the objective function via backpropagation can be prohibitively memory intensive. In addition, some LLMs are released as API services that only allows input and output manipulations, the users cannot perform backpropagation to obtain derivatives of the objective function in (1.5.1). As a result, this problem must be considered as a DFO problem that has no access to the derivatives of the functions. Additionally, the problem is a noisy DFO problem with deterministic errors as LLMs are often implemented in half or single precision. In Chapter 5, we will present our numerical experiments to improve the performance of LLMs on solving a collection of language tasks in a DFO setting.

CHAPTER 2

On the Numerical Performance of Finite-Difference Based Methods for Derivative-Free Optimization

2.1. Introduction

The problem of minimizing a nonlinear objective function when gradient information is not available has received much attention in the literature; see e.g., [29, 59] and the references therein. A variety of methods have been developed for unconstrained optimization, and some of these methods have been extended to deal with constraints. The important benchmarking paper by Moré and Wild [69] showed that traditional methods, such as the Nelder-Mead simplex method [72] and a leading pattern-search method [44], are not competitive with the interpolation-based trust-region approach pioneered by Powell [78] and developed concurrently by several other authors [29]. The advantages of Powell's approach reported in [69] were observed for both smooth and nonsmooth problems, as well as noisy and noiseless objective functions. Rios and Sahinidis [85] confirmed the findings of Moré and Wild concerning the inefficiency of traditional methods based on Nelder-Mead or pattern searches. Based on these studies, we regard the interpolation-based approach of Powell as a leading method for derivative-free optimization (DFO).

There is, however, an alternative approach for DFO that has been largely neglected in the nonlinear optimization literature. It consists of approximating derivatives using finite differences, and using them within standard derivative-based nonlinear optimization

algorithms. Many of the papers in the DFO literature dismiss this approach at the outset as being too expensive in terms of functions evaluations or as ineffective in the presence of noise. As a result of this prevalent view, the vast majority of papers on DFO methods do not present comparisons against a finite-difference approach. We believe that if such comparisons had been made, particularly in the noiseless setting, research in the field would have followed a different trajectory. Ironically, as pointed out by Berahas et al. [7], the finite-difference approach is widely used by practitioners for noiseless problems, often unwittingly, as many established optimization codes invoke a finite-difference option when derivatives are not provided. A disconnect occurred between research and practice, and no systematic effort was made to bridge this gap.

This paper builds upon Berahas et al. [7] and Nesterov and Spokoiny [73], and aims to bring the *gradient approximation approach* to the forefront of DFO research by illustrating its performance on a variety of unconstrained and constrained problems, with and without noise. Gradient approximations can be computed using finite differences or random sampling techniques [73]. In this paper, we study only the former approach.

As is well known, the use of finite differences is delicate in the presence of noise, and unless carefully implemented, can lead to inefficiencies or outright failure. Nevertheless, when the standard deviation of the noise is known or can be estimated, the approximation of derivatives can be placed on a solid theoretical footing. One can view this task as the computation of derivatives of a smoothed function [73], or as the computation of an estimator that minimizes a mean squared error [71].

Finite-difference-based methods for DFO enjoy two appealing features. They can easily exploit parallelism in the evaluation of functions during finite differencing, and they can be

built using existing software for constrained and unconstrained optimization, sometimes rather easily. This obviates the need to redesign existing unconstrained DFO methods to handle more general problems, an effort that has taken two decades for interpolation-based trust-region methods [25–29, 76, 79, 80], and is yet incomplete. The strategy often suggested of simply applying an unconstrained DFO method to a penalty or augmented Lagrangian reformulation will not yield an effective general purpose method, as is known from modern research in nonlinear programming; see e.g. [75, chapter 17].

To gauge the efficiency of the finite-difference approach for DFO, we implemented it within three derivative-based codes: L-BFGS [75] for unconstrained optimization; LMDER [68] for nonlinear least squares; and KNITRO [16] for inequality constrained optimization. In L-BFGS we employ a relaxed line search for which convergence results in the presence of (errors) or noise are established in [7, 53, 104]. LMDER and KNITRO were not modified in any way, and thus our tests provide baseline performance as noise-tolerant variations of these methods should yield improved performance. (Convergence results have not been established for LMDER and KNITRO in the noisy setting, but [99] analyzes a modification of trust region methods that is relevant to LMDER.) We tested the aforementioned codes against three established codes designed specifically for DFO: NEWUOA [78] for general unconstrained optimization; DFO-LS [18] for nonlinear least squares; and COBYLA [77] for inequality constrained optimization. A large number of experiments were performed in this study. In sections 2.2, 2.3, and 2.4, we display graphs or tables that attempt to summarize the findings as well as possible. The complete set of results, as well as fine details of implementation, are presented in a companion technical report [94].

2.1.1. Literature Review

Gill et al. [40] proposed an adaptive approach for computing the difference interval h, assuming that a bound on the errors in the objective function is known. This work, inspired by earlier research by Lyness [65], pays careful attention to the estimation of bounds on the second (or third) derivatives, since these bounds are needed to obtain an accurate estimate of h. Examples are presented illustrating cases when the approach may fail. To our knowledge, [40] is the first in-depth study (in the context of optimization) of finite-difference gradient approximations in the presence of errors. There was, however, no follow-up on the application of these techniques for derivative-free optimization.

The paper that influenced our work the most is by Moré and Wild [71], who discuss how to choose the differencing interval as a function of the noise level and a bound on the second (or third) derivative, so as to obtain nearly optimal gradient estimates. The noise level, defined as the standard deviation of the noise, can be estimated by sampling (in the case of stochastic noise) or using a table of differences [70] (in the case of computational noise). The authors propose `ECnoise`, a practical procedure for estimating stochastic or computational noise [70]. They also give some attention to the practical estimation of the second derivative.

Nesterov and Spokoiny [73] analyze first-order methods with randomized gradient approximations. They establish worst case complexity bounds for nonsmooth and smooth problems. Numerical experiments with these types of methods are reported in [8, 10]. Kelley et al. [21, 55] propose a finite-difference BFGS method, called implicit filtering, designed for the case when noise can be diminished at any iteration, as needed. In that approach, the finite-difference interval decreases monotonically.

The book by Conn, Scheinberg and Vicente [29] gives a thorough treatment of the interpolation-based trust-region approach for DFO. It presents foundational theoretical results as well as detailed algorithmic descriptions. Larson, Menickelly and Wild [59] give a comprehensive review of DFO methods as of 2018. Their survey covers deterministic and randomized methods, noisy and noiseless objective functions, problem structures such as least squares and empirical risk minimization, and various types of constraints.

Audet and Hare [2] review direct-search and pattern-search methods, and describe a variety of practical applications solved with MADS, and NOMAD. Neumaier [74] reviews methods endowed with convergence guarantees, with an emphasis on global optimization. Kimiaei [57] proposes a randomized method, called VSBBON that implements a noisy line search and employs quadratic models in subspaces determined adaptively.

2.1.2. Contributions of this Paper

To our knowledge, this is the first systematic investigation into the practical performance of finite-difference-based DFO methods relative to established techniques, across a range of problems, with and without noise in the functions. The main contributions of this paper can be summarized as follows. We use the acronym "FD-DFO method" for a method that employs some form of finite differences to approximate the gradient of the objective function and (possibly) the constraints.

(1) For *noiseless functions*, we found that the FD-DFO methods are at least as efficient, if not superior, to established methods, across all three categories of problems, without the use of sophisticated procedures for determining the finite-difference interval.

(2) For *noisy functions*, we observed that NEWUOA is more efficient and accurate than the finite-difference L-BFGS method for unconstrained optimization, but not by a wide margin. DFO-LS is comparable to the finite-difference version of LMDER, for least-squares problems. A simple finite-difference version of KNITRO has comparable performance with COBYLA, for inequality-constrained problems.

(3) The differencing formulas used in our experiments performed well on most, but not all noisy problems. More sophisticated techniques for estimating bounds on the second (or third) derivatives will improve the accuracy of the differencing interval, as well as the overall performance of FD-DFO methods, on noisy problems.

Conclusions (1) and (2) are based on sequential execution. If function evaluations can be parallelized, we expect FD-based methods to have a clear advantage over existing DFO methods, for all classes of problems. On the other hand, we should note that the interpolation-based trust-region methods we tested (NEWUOA and DFO-LS) proved to be more robust in the presence of noise than we expected. These methods do not require knowledge of the noise level in the objective function and yet performed reliably for most levels of noise, suggesting that the internal logic of the algorithm normally reacts correctly to the noise inherent in the problem (although rare failures were observed).

We also comment on some limitations of this work. For each problem class, we employed only one established DFO code to benchmark the efficiency of the corresponding FD-DFO method. We found it essential to work with a small number of codes that we could understand well, but benchmarking other software is desirable. Another limitation of this study is that it considers only one model of noise: additive uniformly-distributed bounded noise. Although we experimented with normal noise and observed that the FD-based codes

worked well, we do not know how robust these methods are for heavy tail distributions or anisotropic noise. We focused on just one simple model of noise because this already raised some important algorithmic questions that need to be resolved to make FD-DFO methods highly reliable in practice.

As our focus was on scalable local optimization methods, we did not consider global optimization methods, such as Bayesian optimization, surrogate optimization, evolutionary methods [37, 39, 45], or other global optimization methods that employ restarts, grid searches or surrogate models [74, 87].

2.2. Unconstrained Optimization

Let us consider the solution of unconstrained optimization problems of the form

$$\min_{x \in \mathbb{R}^n} \phi(x), \tag{2.2.1}$$

given only noisy evaluations,

$$f(x) = \phi(x) + \epsilon(x). \tag{2.2.2}$$

Here, $\epsilon(x)$ denotes (deterministic) computational error or the realization of a random variable at x representing noise. The function ϕ is assumed to be smooth. We compare the performance of NEWUOA and L-BFGS with finite-difference gradients on 73 unconstrained `CUTEst` problems [42], varying the dimension of each problem up to $n \leq 300$ whenever possible. All experiments were run in double precision. The methods tested in our experiments are as follows.

- `NEWUOA`: A model-based trust-region derivative-free algorithm that forms quadratic models using interpolation of function values, combined with a minimum Frobenius-norm update of the Hessian approximation; see Powell [78]. We called the code in Python 3.7 through PDFO developed by Ragonneau and Zhang [84]. We used the default settings in NEWUOA, which in particular, set the number of interpolation points to $2n + 1$.
- `FD-L-BFGS`: A finite-difference implementation of the limited-memory BFGS algorithm [75] with a bisection Armijo-Wolfe line search. We use a memory of size 10. Forward- and central-difference options are tested. At intermediate trial points generated during the line search, the directional derivative is computed via forward- or central-differences along the direction of interest.

Instead of NEWUOA we could have used DFOTR [5], which in our experience is often competitive with NEWUOA in terms of function evaluations, but has much higher per-iteration cost.

We first consider the case when noise is not present and later study the effect of noise on the performance of the algorithms.

2.2.1. Experiments on Noiseless Functions

In the first set of experiments, we let $\epsilon(x) \equiv 0$, so that the only errors in the function evaluations are due to machine roundoff, and we let ϵ_M denote unit roundoff. The approximate gradient $g(x) \in \mathbb{R}^n$ of the objective function $\phi(x)$, computed by finite

differencing, is given as:

(2.2.3)
$$[g(x)]_i = \frac{f(x + h_i e_i) - f(x)}{h_i}, \qquad h_i = \max\{1, |[x]_i|\}\sqrt{\epsilon_M}; \quad \text{(forward differencing)}$$

(2.2.4)
$$[g(x)]_i = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i}, \quad h_i = \max\{1, |[x]_i|\}\sqrt[3]{\epsilon_M}. \quad \text{(central differencing)}$$

The directional derivative $D_p\phi(x) = \nabla\phi(x)^T p$ of ϕ along a direction p is required within the Armijo-Wolfe line search employed by L-BFGS. It is approximated as:

(2.2.5)
$$g(x; p) = \frac{f(x + hp_u) - f(x)}{h}\|p\|, \qquad h = \sqrt{\epsilon_M}; \quad \text{(forward differencing)}$$

(2.2.6)
$$g(x; p) = \frac{f(x + hp_u) - f(x - hp_u)}{2h}\|p\|, \quad h = \sqrt[3]{\epsilon_M}; \quad \text{(central differencing)}$$

where $p_u = p/\|p\|$ is the normalized direction.

These choices of h can be improved by including contributions of the second and third derivatives, respectively, as discussed in the next subsection. However, we found that in the noiseless setting, and for our test functions, such a refinement is not needed to make the finite-difference L-BFGS approach competitive.

The algorithms are terminated when either

(2.2.7)
$$\phi(x_k) - \phi^* \leq \tau \cdot \max\{1, |\phi^*|\},$$

with $\tau = 10^{-6}$, or when the limit of $500 \times n$ function evaluations is reached. The optimal value ϕ^* is determined by running BFGS with exact gradients (provided by `CUTEst` [42])

until no more progress can be made on the function. Complete numerical results are given in the technical report [94].

In Figure 2.1, we summarize the results using *log-ratio profiles* proposed by Morales [67], which in this case report the quantity

$$\log_2 \left(\frac{\text{evals}_{\text{LBFGS}}}{\text{evals}_{\text{NEWUOA}}} \right), \tag{2.2.8}$$

where $\text{evals}_{\text{LBFGS}}$ and $\text{evals}_{\text{NEWUOA}}$ denote the total number of function evaluations for FD-L-BFGS and NEWUOA to satisfy (2.2.7) or reach the maximum number of function evaluations. In the figures, the ratios (2.2.8) are plotted in increasing order. Thus, the larger the shaded region, the more successful a method.

Overall, we observe in these tests (with $\tau = 10^{-6}$) that forward-difference L-BFGS outperforms NEWUOA on the majority of problems in terms of function evaluations. This is perhaps surprising because NEWUOA was designed to be parsimonious in terms of function evaluations, whereas finite-difference L-BFGS requires n function evaluations per iteration. It is also notable that this is achieved without any additional information (for example, the Lipschitz constant of the gradient for forward differencing) to squeeze out the best possible accuracy of the gradient in the finite-difference L-BFGS approach. As expected, central-difference L-BFGS requires significantly more function evaluations than the forward-difference option, and does not provide significant benefit in terms of solution accuracy for the majority of the problems. In particular, when run in double precision, forward-difference L-BFGS is able to converge to the same tolerance that one would expect with analytical gradients.

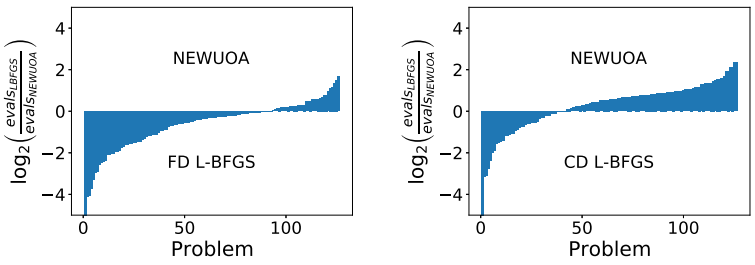

Figure 2.1. *Efficiency, Noiseless Case.* Log-ratio profiles for the total number of function evaluations to achieve (2.2.7) with $\epsilon(x) = 0$. The left figure compares forward-difference L-BFGS with NEWUOA, and the right figure compares central-difference L-BFGS with NEWUOA.

In terms of CPU time, NEWUOA's execution time grows much faster with the problem dimension than L-BFGS' because one iteration of NEWUOA requires $O(n^2)$ flops, whereas the iteration cost of L-BFGS is $O(n)$ flops and all test functions are inexpensive to evaluate. Across all the problems, we observe that when $n \approx 100$, NEWUOA can take at least 5-10 times longer than L-BFGS in terms of wall-clock time.

While NEWUOA is an inherently sequential algorithm, finite-difference L-BFGS offers ample opportunities for parallelism when computing the finite-difference approximation to the gradient. This is an often overlooked benefit of finite-difference-based methods in the DFO literature, as we are not aware of implementations of model-based trust-region methods that benefit significantly from parallelism.

These results also indicate that finite-difference L-BFGS is likely to be much more efficient than direct search methods (Nelder-Mead or pattern search) on noiseless problems since the latter are not competitive with the interpolation-based trust region approach, as mentioned above.

2.2.2. Experiments on Noisy Functions

In this set of experiments, we synthetically inject uniform stochastic noise into the objective function. In particular, we sample $\epsilon(x) \sim \sigma_f U(-\sqrt{3}, \sqrt{3})$ i.i.d. independent of x, where $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$. By construction of $\epsilon(x)$, we have that $\sigma_f^2 = \mathbb{E}[\epsilon(x)^2]$. We refer to standard deviation of the noise σ_f as the *noise level*.

We employ a more precise formula for the finite-difference interval than in the noiseless setting — one that depends both on the noise level σ_f and on the curvature of the function. Specifically, we compute a different h_i for each coordinate direction based on the well known results in [71]. We define

$$[g(x)]_i = \frac{f(x + h_i e_i) - f(x)}{h_i}, \qquad h_i = \sqrt[4]{8}\sqrt{\frac{\sigma_f}{L_i}}; \quad \text{(forward differencing)} \tag{2.2.9}$$

$$[g(x)]_i = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i}, \quad h_i = \sqrt[3]{\frac{3\sigma_f}{M_i}}, \qquad \text{(central differencing)} \tag{2.2.10}$$

where L_i and M_i are bounds on the second and third derivative along the i-th coordinate direction e_i.

We estimate L_i using a second-order difference. Given a direction $p \in \mathbb{R}^n$ where $\|p\| = 1$, we define

$$\Delta(t) = f(x + tp) - 2f(x) + f(x - tp),$$

where t is the second-order differencing interval. The Lipschitz constant L along the direction p can thus be approximated as $L \approx \Delta(t)/t^2$. However, as with the choice of h, we need to be careful in the selection of t, and for this purpose we employ an iterative

technique proposed by Moré and Wild [71], whose goal is to find an interval t that satisfies

$$|\Delta(t)| \geq \tau_1 \epsilon_f, \qquad \tau_1 \gg 1 \tag{2.2.11}$$

$$|f(x \pm tp) - f(x)| \leq \tau_2 \max\{|f(x)|, |f(x \pm tp)|\}, \qquad \tau_2 \in (0,1). \tag{2.2.12}$$

These heuristic conditions aim to ensure that t is neither too small nor too large; see [71] for a full description of the Moré-Wild (MW) technique. This technique cannot, however, be guaranteed to find a t that satisfies (2.2.11), (2.2.12). To account for this, we developed the following procedure for estimating the constants L_i for forward differencing.

Procedure I. *Adaptive Estimation of L*

1. At the first iteration of the L-BFGS method, invoke the MW procedure to compute t_i, for $i = 1, ..., n$. If such a t_i can be found to satisfy (2.2.11), (2.2.12), set $L_i = \max\{10^{-1}, |\Delta(t_i)|/t_i^2\}$ in (2.2.9). Otherwise, set it to $L_i = 10^{-1}$. Store the L_i in a vector $\mathbf{L}$. To calculate the directional derivative in the Armijo-Wolfe line search, set the Lipschitz constant to $L = \|\mathbf{L}\|/\sqrt{n}$.

2. If at any iteration of the L-BFGS algorithm the line search returns a steplength $\alpha_k < 0.5$, then re-estimate the vector $\mathbf{L}$ by calling the MW procedure as in step 1 at the current iterate x_k.

We modify the line search in FD-L-BFGS to better handle noise; see Shi et al. [91] for details. No other changes were made to the L-BFGS method. It is well known, that when the gradient is sufficiently accurate, L-BFGS generates well-scaled directions so that $\alpha_k = 1$ is acceptable. Thus, the occurrence of $\alpha_k < 0.5$ is viewed as an indication that the

curvature of the problem may have changed, and should be re-estimated. The function evaluations performed in the estimation of $\mathbf{L}$ will be accounted for in the numerical results presented below.

The technique for computing differencing interval for central differences is described in [94].

2.2.2.1. Accuracy. We first compare the accuracy achieved by each algorithm, as measured by the optimality gap $\phi(x_k) - \phi^*$. We do so by running NEWUOA until $\rho = \rho_{\text{end}} = 10^{-6}$, and running the FD-L-BFGS method until the objective function could not be improved over 5 consecutive iterations. In Figure 2.2, we report the log-ratio profile

$$\log_2\left(\frac{\phi_{\text{LBFGS}} - \phi^*}{\phi_{\text{NEWUOA}} - \phi^*}\right) \tag{2.2.13}$$

for $\sigma_f = 10^{-5}$, where ϕ_{LBFGS}, ϕ_{NEWUOA} denote the lowest objective achieved by each method. (The results are representative of those obtained for $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-7}\}$.) Since we include only runs where both solvers converged to the same local minimizer, the differences in both the numerator and denominator are nonnegative.

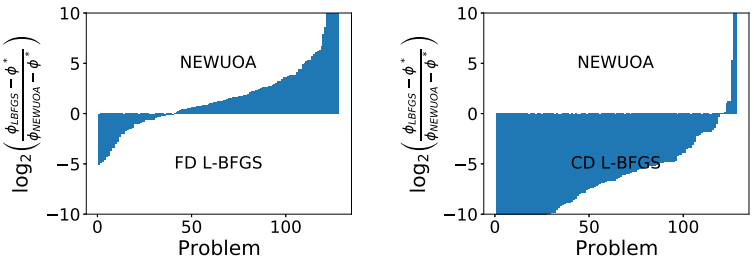

Figure 2.2. *Accuracy, Noisy Case for* $\sigma_f = 10^{-5}$. Log-ratio optimality gap profiles comparing NEWUOA against forward-difference L-BFGS (left) and central-difference L-BFGS (right).

As seen in Figure 2.2, NEWUOA achieves higher accuracy in the solution than forward-difference L-BFGS, while central-difference L-BFGS yields far better accuracy than both. It is not surprising that central differencing yields much higher accuracy than forward differencing since the noise levels of their gradient approximations are, respectively, $O(\sigma_f^{2/3})$ and $O(\sigma_f^{1/2})$. On the other hand, it is not straightforward to analyze the error contained in the gradient approximation constructed by NEWUOA, and in turn its final accuracy. To try to shed some light into this question, we tested NEWUOA using only $n+1$ interpolation points and observed that it now lags behind FD-L-BFGS in terms of accuracy. More generally, our tests suggest that the choice $p = 2n + 1$ recommended by Powell strikes the right balance between accuracy in the solution and the speed of algorithm.

2.2.2.2. Efficiency. We now report the number of function evaluations required to achieve

$$\phi(x_k) - \tilde{\phi}^* \leq \tau \cdot (\phi(x_0) - \tilde{\phi}^*), \tag{2.2.14}$$

for varied τ. Here, $\tilde{\phi}^*$ denotes the best solution obtained by NEWUOA. In Figure 2.3, we report log-ratio profiles based on

$$\log_2 \left(\frac{\text{evals}_{\text{LBFGS}}}{\text{evals}_{\text{NEWUOA}}} \right). \tag{2.2.15}$$

We observe from this figure that NEWUOA is more efficient than forward-difference L-BFGS. The advantage is less significant for $\tau = 10^{-2}$ but becomes pronounced for $\tau = 10^{-6}$, which is consistent with our earlier observation about the ability of NEWUOA to achieve higher accuracy. Central-difference L-BFGS is also less efficient overall than NEWUOA, but becomes more competitive as τ is decreased. It is notable that NEWUOA is able to deliver such

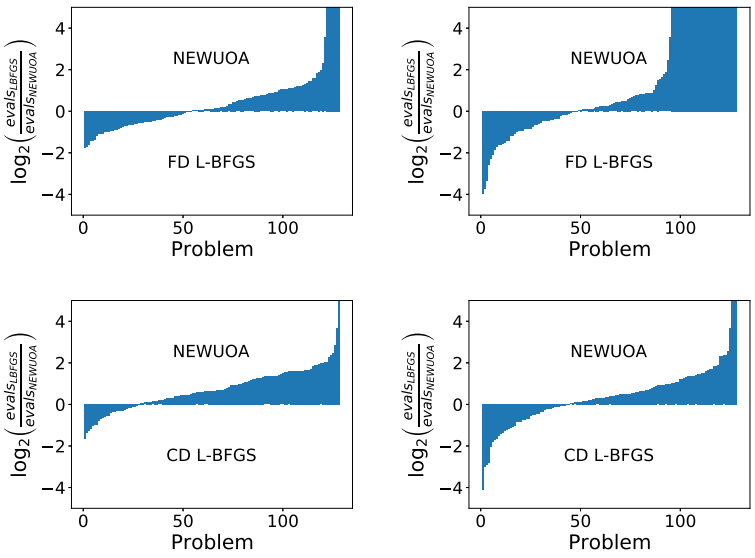

Figure 2.3. *Efficiency, Noisy Case for* $\sigma_f = 10^{-5}$. Log-ratio profiles for the number of function evaluations to achieve (2.2.14) for $\tau = 10^{-2}$ (left) and 10^{-6} (right), comparing NEWUOA against forward-difference L-BFGS (top) and central-difference L-BFGS (bottom). These plots are representative of the other noise levels $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$.

strong performance without knowledge of the noise level of the function. The adjustment of the two trust-region radii in NEWUOA seems to be quite effective: shrinking the radii fast enough to ensure steady progress, but not so fast as to create models dominated by noise. To our knowledge, there has been no in-depth study of the *practical* behavior of NEWUOA (or similar codes) in the presence of noise; we regard this as an interesting research topic.

The results reported in this section highlight the importance of developing more effective finite difference estimation techniques than those employed in our experiments. Recent work along these lines is described in [92].

2.3. Nonlinear Least Squares

In many unconstrained optimization problems, the objective function has a nonlinear least-squares form. Therefore, it is important to pay particular attention to this problem structure in the derivative-free setting. We write the problem as

$$\min_{x\in\mathbb{R}^n} \phi(x) = \tfrac{1}{2}\|\gamma(x)\|^2 = \tfrac{1}{2}\sum_{i=1}^{m} \gamma_i^2(x),$$

where $\gamma : \mathbb{R}^n \to \mathbb{R}^m$ is a smooth function. We assume that the Jacobian matrix $[J(x)]_{ij} = \frac{\partial \gamma_i(x)}{\partial x_j}$ is not available but that the individual residual functions $\gamma_i(x)$ can be computed. More generally, the evaluation of the γ_i may contain noise so that the observed residuals are given by

$$r_i(x) = \gamma_i(x) + \epsilon_i(x), \quad i = 1, \ldots, m,$$

where $\epsilon_i(x)$ models noise as in (2.2.1). Thus, the minimization of the true objective function ϕ must be performed based on noisy observations $r_i(x)$ that define the observed objective function

$$f(x) = \tfrac{1}{2}\|r(x)\|^2 = \tfrac{1}{2}\sum_{i=1}^{m} r_i^2(x). \tag{2.3.1}$$

Since noise is incorporated into each residual function, the model of noise is different from the additive noise model in the general unconstrained case discussed in the previous section. In particular, the function evaluation $f(x) = \frac{1}{2}\sum_{i=1}^{m} \gamma_i^2(x) + 2\epsilon_i(x)\gamma_i(x) + \epsilon_i^2(x)$ contains both multiplicative and additive components of noise.

Our goal is to study the viability of methods based on finite-difference approximations to the Jacobian. To this end, we employ a classical Levenberg-Marquardt trust-region

method where the Jacobian is approximated by differencing, and perform tests comparing it against a state-of-the-art DFO code designed for nonlinear least-squares problems. The rationale behind the selection of codes used in our experiments is discussed next.

Interpolation Based Trust Region Methods.

Cartis and Roberts [86] proposed a Gauss-Newton type approach, referred to as DFO-GN, in which an approximation of the Jacobian is computed by linear interpolation using $n + 1$ function values at recently generated points. An improved version of DFO-GN is DFO-LS [18], which provides a variety of options and heuristics to accelerate convergence and promote a more accurate solution. The numerical results reported by Roberts et al. [18] indicate that DFO-LS is a state-of-the-art code for DFO least squares, and therefore will be used in our benchmarking.

Finite-Difference Gauss-Newton Method.

One can employ finite differencing to estimate the Jacobian matrix $J(x)$ within any method for nonlinear least squares, and since this is a mature area, there are a number established solvers. We chose LMDER for our experiments, which is part of the `MINPACK` package [68] and is also available in the `scipy` library. We did not employ LMDIF, the finite-difference version of LMDER, because it does not allow the use of different differencing intervals for each of the residual functions $r_i(x)$; we elaborate on this point below. Another code available in `scipy` is TRF [13], but our tests show that LMDER is slightly more efficient in terms of function evaluations, and tends to give higher accuracy in the solution. The code NLS, recently added to the `Galahad` library [43] would provide an interesting alternative. That method, however, includes a tensor to enhance the Gauss-Newton model,

and since this may give it an advantage over DFO-LS, we decided to employ the more traditional code LMDER.

In summary, the solvers used in our tests are:

- `LMDER`: A derivative-based Levenberg-Marquardt trust-region algorithm from the `MINPACK` software library [68], where the finite-difference module is supplied by us. We call the code in Python 3.7 through `scipy` version 1.5.3, using the default parameter settings.
- `DFO-LS`: The most recent DFO software developed by Cartis et al. [18] for nonlinear least squares. This method uses linear interpolation to construct an approximation to the Jacobian matrix, which is then used in a Gauss-Newton-type method. We used version 1.0.2 in our experiments, with default settings except that the `model.abs_tol` parameter is set to 0 to avoid early termination.

The test problems in our experiments are those used by Moré and Wild [69], which have also been employed by Roberts and Cartis [86] and Zhang et al. [105]. The 53 unconstrained problems in this test set include both zero and nonzero residual problems, with various starting points, and are all small dimensional, with $n \leq 12$. To measure efficiency, we regard m evaluations of individual residual components $r_i(\cdot)$ as one function evaluation. These m evaluations of the individual residual components are not necessarily performed at the same point. We terminate the algorithms when either: i) the maximum number of function evaluations ($500 \times n$) is reached, ii) an optimality gap stopping condition, specified below, is triggered; or iii) the default termination criterion of the two codes is satisfied with tolerance of 10^{-8} (this controls the minimum allowed trust region radius).

2.3.1. Experiments on Noiseless Functions

We first consider the noiseless case corresponding to $\epsilon_i(x) \equiv 0$, $\forall i$. For LMDER, we estimate the Jacobian $J(x)$ using forward differences. As before, let ϵ_M denote machine precision. We first evaluate $\{r(x + h_j e_j)\}_{j=1}^n$, where h_j is defined as in (2.2.3), and compute the Jacobian estimate as

$$[\hat{J}(x)]_{ij} = \frac{r_i(x + h_j e_j) - r_i(x)}{h_j}. \tag{2.3.2}$$

As in the previous section, we consider log-ratio profiles to compare the efficiency and accuracy of LMDER and DFO-LS. We record

$$\log_2\left(\frac{\text{evals}_{\text{LMDER}}}{\text{evals}_{\text{DFOLS}}}\right) \quad \text{and} \quad \log_2\left(\frac{\phi_{\text{LMDER}} - \phi^*}{\phi_{\text{DFOLS}} - \phi^*}\right),$$

where ϕ^* is obtained from [86], ϕ_{DFOLS}, ϕ_{LMDER} denote the best function values achieved by the respective methods, and $\text{evals}_{\text{DFOLS}}$, $\text{evals}_{\text{LMDER}}$ denote the number of function evaluations needed to satisfy the termination test (2.2.7) for various values of τ. If either of the solvers reach the maximum number of function evaluations before finding a solution that satisfies (2.2.7), we will set the ratio to a very large number (or negated). If both solvers fail, then the log-ratio is zero. The results comparing DFO-LS and LMDER are summarized in Figures 2.4 and 2.5. The complete table of results is given in the technical report [94].

The performance of the two methods appears to be comparable since the area of the shaded regions is similar. This may be surprising since the Gauss-Newton approach seems to be a particularly effective way of designing an interpolation-based trust-region method, requiring only linear interpolation to yield useful second-order information. Cartis

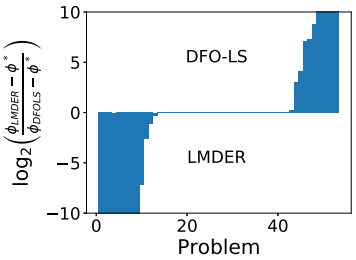

Figure 2.4. *Accuracy, Noiseless Case.* Log-ratio optimality gap profiles comparing DFO-LS and LMDER for $\epsilon(x) = 0$.

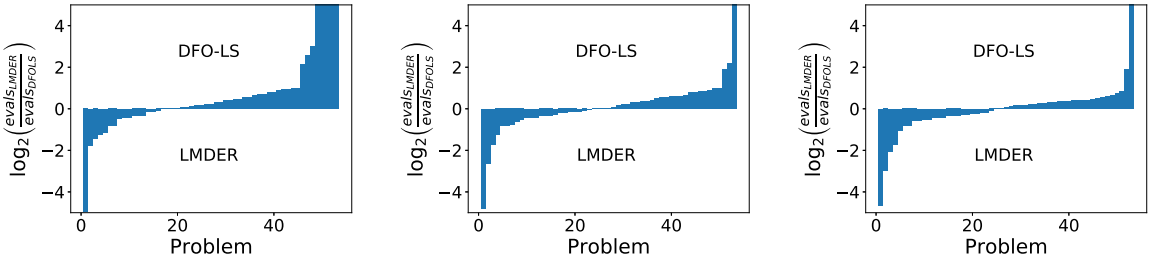

Figure 2.5. *Efficiency, Noiseless Case.* Log-ratio profiles comparing DFO-LS and LMDER for $\epsilon(x) = 0$. The figures measure number of function evaluations to satisfy (2.2.7) for $\tau = 10^{-1}$ (left), 10^{-3} (middle), and 10^{-6} (right).

and Roberts [86] dismiss finite-difference methods at the outset, and do not provide numerical comparisons with them. However, the tradeoffs of the two methods merit careful consideration. DFO-LS requires only one function evaluation per iteration, but its gradient approximation, $\hat{J}(x_k)^T r(x_k)$, is inaccurate until the iterates approach to the solution and the trust region has shrunk. In contrast, the finite-difference Levenberg-Marquardt method in LMDER computes quite accurate gradients in this noiseless setting, requiring a much smaller number of iterations, but at a much higher cost per iteration in terms of function evaluations. The tradeoffs of the two methods appear to yield, in the end, similar performance, but we should note that DFO-LS is typically more efficient in the early stages of the optimization, as illustrated in Figure 2.5 for the low tolerance level $\tau = 10^{-1}$. On

the other hand, the finite-difference approach is more amenable to parallel execution, as mentioned in the previous section.

Let us now consider the linear algebra cost of the two methods. A typical iteration of DFO-LS solves the interpolation system with LU factorization, for n different right hand sides, at a per-iteration cost of $O(mn^2)$ flops, assuming that $m > n$. (DFO-LS offers a number of options, such as regression, which may involve a higher cost, but we did not invoke those options.) The linear algebra cost of the finite-difference version of LMDER described above is $O(mn)$ flops. Therefore, in terms of CPU time, LMDER is faster than DFO-LS whenever the cost of function evaluations does not dominate the iteration cost.

2.3.2. Experiments on Noisy Functions

Let us assume that the noise model is the same across all residual functions, i.e., $\epsilon_i(x)$ are i.i.d. for all i. As in Section 2.2.2, we generate noise independently of x, following a uniform distribution, $\epsilon_i(x) \sim \sigma_f U(-\sqrt{3}, \sqrt{3})$, with noise levels $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$.

Differencing will be performed more precisely than in the noiseless case. Following [70], the forward-difference approximation of the Jacobian is defined as

$$[\hat{J}(x_k)]_{ij} = \frac{r_i(x + h_{ij}e_j) - r_i(x)}{h_{ij}}, \quad \text{where } h_{ij} = 8^{1/4}\left(\frac{\sigma_f}{L_{ij}}\right)^{1/2}, \tag{2.3.3}$$

where L_{ij} is a bound on $|e_j^T \nabla^2 \gamma_i(x) e_j|$ within the interval $[x, x + h_{ij}e_j]$. To estimate L_{ij} for every pair (i, j), and at every iteration, would be impractical, and normally unnecessary. Several strategies can be designed to provide useful information at an acceptable cost. For concreteness, we estimate L_{ij} once at the beginning of the run of LMDER.

More concretely, we compute a different h_{ij} for every residual function γ_i and each coordinate direction e_j at the starting point, and keep h_{ij} constant throughout the run of LMDER. The $\{L_{ij}\}$ for $i = 1, ..., m$, $j = 1, ..., n$ are obtained by applying the Moré-Wild (MW) procedure [71] described in the previous section to estimate the Lipschitz constant for function γ_i along coordinate directions e_j. If the MW procedure fails, we set $L_{ij} = 1$. The cost of computing the L_{ij}, in terms of function evaluations, is accounted for in the results reported below.

In DFO-LS, we did not employ restarts, and set the `obj_has_noise` option to its default value `False`, which also changes some trust-region-related parameters. We did so for two reasons. First, restarts introduce randomness, and as Cartis et al. [18] observed, can lead the algorithm to a different minimizer, making comparisons difficult. In addition, restarts are designed to allow the algorithm to make further progress as it reaches the noise level of the function.

Accuracy. We compare the best optimality gap achieved by LMDER and DFO-LS. We run the algorithms using their default parameters (except that we set `model.abs_tol = 0` for DFO-LS) until no further progress could be made. In Figure 2.6, we plot the log-ratio profiles

$$\log_2 \left(\frac{\phi_{\text{LMDER}} - \phi^*}{\phi_{\text{DFOLS}} - \phi^*} \right), \tag{2.3.4}$$

for noise levels $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$.

We observe that DFO-LS is more accurate than LMDER, which points to the strengths of DFO-LS, since it does not require knowledge of the noise level of the function in its

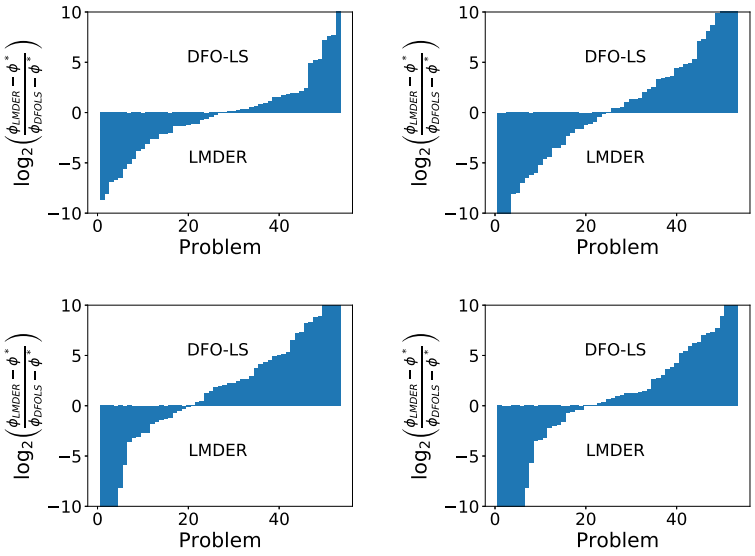

Figure 2.6. *Accuracy, Noisy Case.* Log-ratio optimality gap profiles comparing DFO-LS and LMDER for $\sigma_f = 10^{-1}$ (upper left), 10^{-3} (upper right), 10^{-5} (bottom left), 10^{-7} (bottom right). The bounds on the second derivatives are kept constant over the optimization process.

internal logic. However, our implementation of LMDER is not sophisticated, as fixing the Lipschitz constant at the start of the finite-difference method is not always a good strategy. An interesting open question is whether the adaptive strategies recently proposed in [92] would close the accuracy gap.

Efficiency. To measure the efficiency of the algorithms in the noisy case, we record the number of function evaluations required to satisfy the termination condition (2.2.14), where $\tilde{\phi}^*$ denotes the best objective value achieved by the two solvers, for a given noise level. To do so, both solvers were run until they could not make more progress. The differencing interval in LMDER was computed as in the experiments measuring accuracy for the noisy setting, i.e., by employing only the Moré-Wild procedure at the first iteration.

In Figure 2.7, we plot

$$\log_2\left(\frac{\text{evals}_{\text{LMDER}}}{\text{evals}_{\text{DFOLS}}}\right) \tag{2.3.5}$$

for two values of the tolerance parameter τ and for two levels of noise. We omit the plots for other values of τ and σ, as they demonstrate similar performance. (When (2.2.14) cannot be satisfied for a solver within the given budget of $500 \times n$ function evaluations, we set the corresponding value in (2.3.5) to a very large number.)

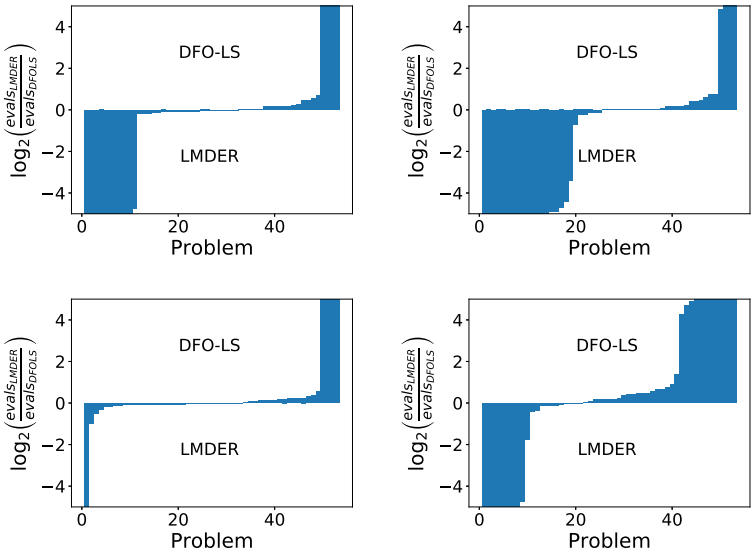

Figure 2.7. *Efficiency, Noisy Case.* Log-ratio profiles comparing DFO-LS and LMDER for $\sigma_f = 10^{-1}$ (top row) , and 10^{-3} (bottom row). The figure measures the number of function evaluations to satisfy (2.2.14) for $\tau = 10^{-1}$ (left column), and 10^{-6} (right column). Lipschitz constants were estimated only at the start of the LMDER run.

There is no clear winner among the two codes used in the experiments reported in Figure 2.7. For low accuracy ($\tau = 0.1$), LMDER appears to be more efficient, whereas the opposite is true for high accuracy ($\tau = 10^{-6}$). We note again that DFO-LS is able to

handle different noise levels efficiently and reliably, without knowledge of the noise level or Lipschitz constants. On the other hand, the finite-difference approach is competitive even with a fairly coarse Lipschitz estimation procedure, and perhaps more important, it can be incorporated into existing codes (doing so in LMDER required little effort). In other words, in the finite-difference approach to derivative-free optimization, algorithms do not need to be constructed from scratch but can be built as adaptations of existing codes.

2.3.2.1. Commentary. Our finite-difference approach is tailored to each individual residual component and each coordinate direction. However, it is natural to question the necessity of estimating the Lipschitz constant for each component of each individual residual, as we did in our experiments, since this is affordable only if one can evaluate residual functions individually. One can envision problems for which a much simpler Lipschitz estimation suffices. For example, in data-fitting applications all individual residual functions γ_i may be similar in nature. In this setting, one could use a single Lipschitz constant, say L, across all components and residual functions, especially when the variables are scaled prior to optimization. L could be updated a few times in the course of the optimization process. On the other hand, if the scale of the variables varies significantly, one can compute Lipschitz constants L_i for each component across all residual functions, requiring the estimation of n Lipschitz constants.

2.4. Constrained Optimization

We now consider inequality-constrained nonlinear optimization problems of the form

$$\min_{x\in\mathbb{R}^n} \phi(x) \qquad \text{s.t. } \psi(x) \leq 0, \quad l \leq x \leq u, \tag{2.4.1}$$

where $\psi : \mathbb{R}^n \to \mathbb{R}^m$ represents a set of m linear or nonlinear constraints, $l, u \in \mathbb{R}^n$, and ϕ and ψ are twice continuously differentiable. We assume that the derivatives of ϕ and ψ are not available, and more generally that we have access only to noisy function evaluations:

$$f(x) = \phi(x) + \epsilon(x), \qquad c_j(x) = \psi_j(x) + \epsilon_j(x). \tag{2.4.2}$$

We assume the same noise model for the objective and each of the constraint functions, for simplicity. We do not present comparisons for general problems involving both equality and inequality constraints because we were not able to find an established DFO code of such generality that was sufficiently robust in our experiments (COBYLA accepts only inequality constraints).

To our knowledge, there have been very few comparative studies of DFO methods for constrained optimization as judged by the very few references in the comprehensive review by Larson et al. [59]. The best known interpolation-based DFO software available for solving problem (2.4.1) is COBYLA, developed by Powell [77]. The method implemented in that code constructs linear approximations to ϕ and ψ at every iteration using function interpolation at points placed on a simplex in $\mathbb{R}^n$; it is designed to handle only inequality constraints. A more recent method by Powell, in the spirit of NEWUOA, is LINCOA [76]. It implements an interpolation-based trust-region approach but it can handle only linear constraints.

There are many production-quality software packages for deterministic constrained optimization where we could implement the finite difference DFO approach. We chose KNITRO because one of the algorithms it offers is a simple sequential quadratic programming (SQP) method that is close in spirit to COBYLA. We did not to employ the interior-point

methods offered by KNITRO, which are known to be very powerful techniques for handling inequality constraints, because they may put COBYLA at an algorithmic disadvantage. In the same vein, we did not employ SNOPT because it implements a sophisticated SQP method with many advanced features to improve efficiency and reliability. In short, we selected a simple nonlinear optimization method to more easily identify the strengths and weaknesses of the finite difference approach.

The two codes are tested under the following settings.

- `COBYLA`. We ran the version of COBYLA maintained in the PDFO package [84]. We set the final trust region radius to 10^{-8} (`rhoend=1e-8`) to observe its asymptotic behavior, particularly in the noiseless case. We ran PDFO version 1.0, and called COBYLA via its Python interface (Python 3.7.7).
- `KNITRO`. We ran Artelys Knitro 12.2 with `alg=4` (an SQP algorithm), `gradopt=2` (forward differencing), and `hessopt=6` (L-BFGS). The choice of the finite difference interval h is described below. In order to make the algorithm as close as possible to COBYLA, we set the memory size of L-BFGS updating to its minimum value, $t = 1$ (`lmsize=1`). For consistency with COBYLA, we disabled the termination test based on the optimality error by setting `opttol=1e-16`, and `findiff_terminate=0`, and instead terminate when the computed step is less than 10^{-8} (by setting `xtol=1e-8` and `xtol_iters=1`). We called KNITRO via its Python interface.

As in the previous sections, we used test problems from the `CUTEst` set [42], which were called through the Python interface, `PyCUTEst` version 1.0. We recall from (2.4.1) that n and m refer to the number of variables and constraints, respectively (excluding bound constraints). We first selected fixed-size problems that have at least one general

nonlinear inequality and have no equality constraints, and for which $n \leq 100$ and $m \leq 100$. The characteristics of the resulting 49 problems are listed in the technical report [94].

2.4.1. Experiments on Noiseless Functions

In the first set of experiments, we applied COBYLA and KNITRO to solve the 49 small-scale, fixed-size `CUTEst` problems with exact function evaluations, i.e., with $\epsilon(x) = \epsilon_j(x) \equiv 0$ for all i. As in prior sections, the approximate gradient $g(x) \in \mathbb{R}^n$ of the objective function and Jacobian $\hat{J}(x) \in \mathbb{R}^{m \times n}$ of the constraints are evaluated by simple forward differences, where h_i is set as in (2.2.3). Both algorithms were stopped when the number of function evaluations exceed $500 \max(n, m)$, or when the trust-region radius or steplength reaches its lower bound for COBYLA or KNITRO, respectively. Both solvers report feasibility error as the max-norm of constraint violations.

We sorted the results into four groups, according to the following outcomes:

(i) Both solvers converged to a feasible point with approximately the same objective function values.

(ii) The solvers converged to feasible points with different objective function values.

(iii) One of the solvers terminated at an infeasible point.

(iv) Both solvers terminated at infeasible points.

There were 32 problems associated with outcome (i). We can use them to safely compare the performance of the two solvers in terms of accuracy and efficiency. (We comment on outcomes (ii)-(iv) in the companion technical report [94].)

Given that the two codes achieved feasibility in these 32 problems, we measure accuracy by comparing the best objective function value obtained by each solver with the value ϕ^* obtained by running KNITRO with exact gradients until it could not make further progress. (In all the runs for determining ϕ^*, feasibility error was less than 10^{-10}.) To measure accuracy, we report the ratios

$$\log_2\left(\frac{\max\{\phi_{\text{KNITRO}} - \phi^*, 10^{-8}\}}{\max\{\phi_{\text{COBYLA}} - \phi^*, 10^{-8}\}}\right). \tag{2.4.3}$$

We compare accuracy up to eight digits because reporting, say, the ratio $10^{-12}/10^{-15}$ would be misleading, given that the accuracy in the constraint violation could have the inverse ratio. The results are presented in Figure 2.8, which shows that for most problems both solvers were able to achieve eight digits of accuracy; for the remaining problems, KNITRO gave higher accuracy.

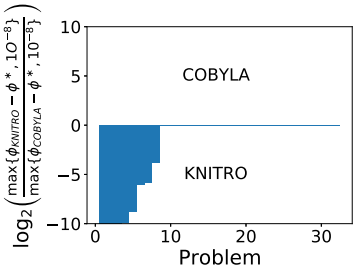

Figure 2.8. *Accuracy, Noiseless Case.* Log-ratio profiles comparing KNITRO and COBYLA for $\epsilon(x) = \epsilon_i(x) = 0$. The figure plots the ratios (2.4.3) for problems for which the two solvers yielded the same solution.

To measure efficiency, we conducted a series of experiments using the same subset of 32 problems corresponding to outcome (i). We record the number of function evaluations, $\text{evals}_{\text{KNITRO}}$ and $\text{evals}_{\text{COBYLA}}$, required by the two codes to satisfy condition (2.2.7) for various values of τ. (If a solver fails to satisfy this test, we set the number of evaluations

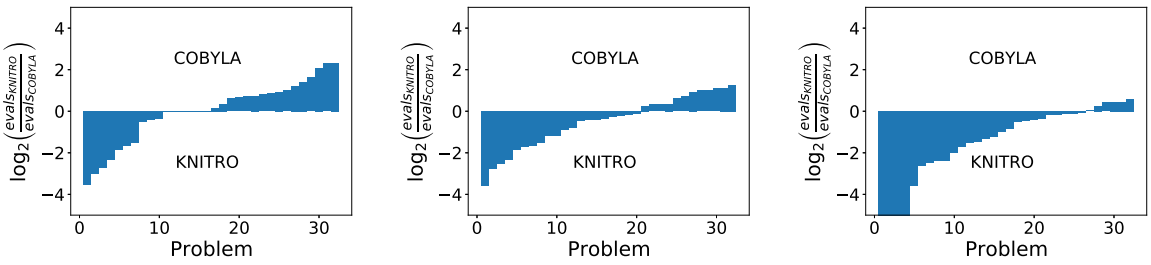

Figure 2.9. *Efficiency, Noiseless Case.* Log-ratio profiles comparing KNITRO and COBYLA for $\epsilon(x) = \epsilon_i(x) = 0$. The figures measure number of function evaluations to satisfy (2.2.7) for $\tau = 10^{-1}$ (left), 10^{-3} (middle), and 10^{-6} (right).

to a large value.) Figure 2.9 plots the ratios

$$\log_2\left(\frac{\text{evals}_{\text{KNITRO}}}{\text{evals}_{\text{COBYLA}}}\right) \tag{2.4.4}$$

for $\tau \in \{10^{-1}, 10^{-3}, 10^{-6}\}$. The technical report [94] contains the complete set of results. We observe that for low accuracy, COBYLA is slightly more efficient, while KNITRO becomes significantly more efficient when high accuracy is required.

We should note that employing memory size $t = 1$ in L-BFGS updating yields a very weak quadratic model in the SQP method of KNITRO. We experimented with a memory of size $t = 10$ and observed that the performance of KNITRO improved, particularly in the early iterations of the runs, but not dramatically, which was surprising.

We also conducted experiments with `CUTEst` test problems of variable dimension and noted that the advantage of the FD approach becomes greater as the dimension of the problem increases; see [94]. This stands in stark contrast with a common perception in the DFO literature that finite differences require too many function evaluations compared to methods specifically designed for DFO problems.

2.4.2. Experiments on Noisy Functions

We now inject artificial noise in the evaluation of the objective and constraint functions, using the same noise model as in the unconstrained setting. We employ forward differences to approximate the gradient and Jacobian, with finite-difference interval

$$h_i = \max\{1, |[x]_i|\}\sqrt{\sigma_f}, \quad i = 1, ..., n.$$

We do not include Lipschitz constant estimates because this simple formula suffices for our purposes.

In the first experiment, we ran the two codes with their least stringent termination tests to observe the quality of the final solutions. As in the noiseless case, we set `rhoend=1e-8` for COBYLA and `xtol=1e-8` for KNITRO, and impose a limit of $500 \max\{n, m\}$ function evaluations. We tested the 32 `CUTEst` problems for which both solvers converged to the same feasible solution in the noiseless case (outcome (i) above). In Figure 2.10, we compare the accuracy, $\phi(x_k) - \phi^*$, in the true objective given by the code codes, up to eight digits, as in (2.4.3). If the feasibility violation is large compared to the noise level, i.e. $\| \max\{\psi(x_k), 0\}\|_\infty \geq \sqrt{3}\sqrt{\sigma_f}$, we mark the corresponding run as a failure. We observe from Figure 2.10 that the performance of the two solvers is comparable, with KNITRO slightly more efficient for low accuracy. Interestingly, for the highest accuracy, $\sigma_f = 10^{-7}$, the performance of the codes is remarkably close (note that the log ratio is nearly zero for about half of the problems).

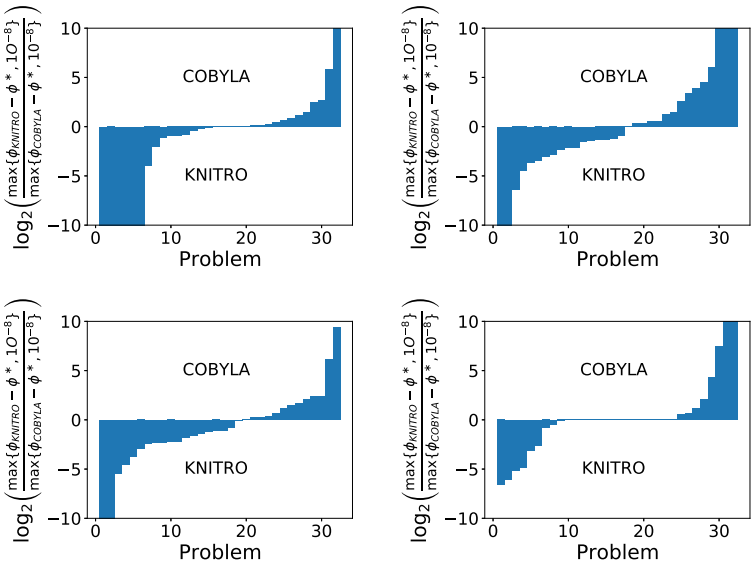

Figure 2.10. *Accuracy, Noisy Case.* Log-ratio profiles comparing accuracy in the objective by KNITRO and COBYLA, for $\sigma_f = 10^{-1}$ (upper left), 10^{-3} (upper right), 10^{-5} (bottom left), and 10^{-7} (bottom right).

Next, we compare the efficiency of the two solvers for two levels of accuracy in the objective. Specifically, we record the number of function evaluations required to satisfy

$$\phi(x_k) - \tilde{\phi}_* \leq \tau(\phi(x_0) - \tilde{\phi}_*) \quad \text{and} \quad \|\max\{\psi(x_k), 0\}\|_\infty \leq \sqrt{3}\sqrt{\sigma_f}, \tag{2.4.5}$$

for $\tau \in \{10^{-2}, 10^{-6}\}$, where $\tilde{\phi}_*$ is the minimum objective value obtained by the two codes. Figure 2.11 plots the ratios (2.4.4), and suggests that it is difficult to choose between the two codes. Therefore, in our tests on noisy problems, an unsophisticated code (KNITRO/SQP) for deterministic nonlinear optimization, using a simple strategy for choosing the finite difference interval, is competitive with COBYLA, a method specifically designed for derivative-free optimization.

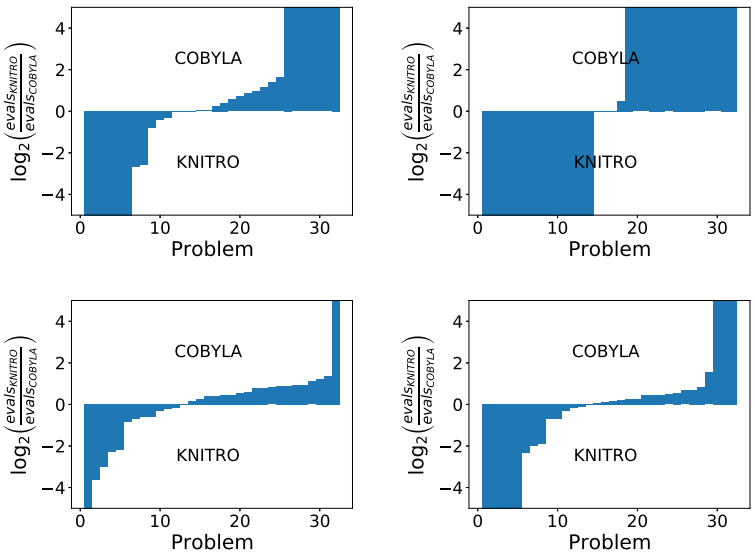

Figure 2.11. *Efficiency, Noisy Case.* Log-ratio profiles (2.4.4) comparing KNITRO and COBYLA for $\sigma_f = 10^{-3}$ (top row) and 10^{-7} (bottom row). The figure measures the number of function evaluations to satisfy (2.4.5) for $\tau = 10^{-2}$ (left) and 10^{-6} (right).

2.5. Final Remarks

Finite-difference approximations are widely employed in numerical analysis, particularly for solving differential equations. The limitations of finite-difference methods are well-documented, particularly in the noisy case. However, optimization provides a more benign setting as errors do not necessarily accumulate; if a poor gradient estimation yields a bad step, it may be corrected at a later step. Two attractive features of finite-difference methods for derivative-free optimization are the simplicity of building them upon existing nonlinear (gradient-based) optimization solvers, and their ease of parallelization.

Our numerical study indicates that finite-difference-based optimization methods can be made competitive, in most cases, against state-of-the-art methods designed specifically

for DFO. Results to that effect were reported by Berahas et al. [7] in the context of unconstrained optimization. Our study tests a simpler version of L-BFGS than that explored in [7], and more important, considers also least squares and constrained optimization—settings for which comparative studies do not exist, to the best of our knowledge.

Our experiments show that the finite-difference approach can be further improved by employing more sophisticated (adaptive) procedures for computing the finite-difference interval. The key is in better automatic estimation of local Lipschitz constants. Investigation along this important line of inquiry should lead to more robust methods for derivative-free optimization.

CHAPTER 3

Adaptive Finite-Difference Interval Estimation for Noisy Derivative-Free Optimization

3.1. Introduction

A powerful approach for derivative-free optimization is to utilize finite differences. This is done by computing a finite-difference approximation to the gradient, and substituting the exact gradient with the approximation within a known nonlinear optimization method; see [93]. These methods operate by spending at least $n + 1$ function evaluations at each iteration to take a meaningful step, where n is the total number of variables. This lies in contrast to interpolation-based methods, which utilize prior function evaluations with only one new evaluation at each iteration; see [29, 59]. Therefore, in order for the finite-difference approach to be effective, one must ensure that the quality of the gradient is satisfactory and significant progress is being made at each iteration of the algorithm (as opposed to n steps of the interpolation-based approach).

Often, black-box functions to be optimized are contaminated by stochastic or computational noise. This noise could arise naturally from modeling randomness within a simulation, or as a bi-product of an adaptive computation, for example through the early termination of an iterative solver. The presence of noise has largely prevented finite-difference methods from gaining more popularity within the derivative-free optimization

community, as the precise choice of the finite-difference interval becomes increasingly critical as the noise level increases.

In particular, the finite-difference interval requires knowledge of both the noise level and higher-order derivative of the function. While the former may be known *a priori* or can be estimated by sampling or computing difference tables [70], the latter quantity is not normally available to the user. In order to make finite-difference methods a viable alternative in the presence of noise, a robust procedure is needed for estimating higher-order derivatives, either implicitly or explicitly.

To put this more precisely, let us consider the problem of estimating the d-th order derivative of a smooth univariate function $\phi : \mathbb{R} \to \mathbb{R}$. We will assume that we are only provided noisy function evaluations of the form

$$f(t) = \phi(t) + \epsilon(t) \tag{3.1.1}$$

where $\epsilon : \mathbb{R} \to \mathbb{R}$ models the error, and that the error is bounded, i.e., there exists $\epsilon_f \geq 0$ such that $|\epsilon(t)| \leq \epsilon_f$. We call ϵ_f the *noise level* of the function. We focus on the univariate case, although this can be easily extended to the multivariate setting for computing the gradient by applying the procedure to each component.

The simplest and cheapest finite-difference approximation to the first derivative is the forward-difference approximation. If $\phi^{(d)}$ denotes the d-th order derivative of ϕ, then the forward-difference approximation is computed by

$$\phi^{(1)}(t) \approx \frac{f(t+h) - f(t)}{h} \triangleq f^{(1)}(t; h) \tag{3.1.2}$$

where $h > 0$ is the finite-difference interval. Note the slight abuse of notation by denoting $f^{(d)}$ as the finite-difference approximation to the d-th order derivative. With no noise, excluding round-off error, one would ideally choose h as small as possible, the common practical choice being $h = \max\{1, |x|\}\sqrt{\epsilon_M}$, where ϵ_M is machine precision, to handle rounding errors. However, this choice of the finite-difference interval may be poor under the presence of large errors, as is well-known.

To see this, consider the following decomposition of the error in the forward-difference approximation:

$$\left|f^{(1)}(t;h) - \phi^{(1)}(t)\right| \leq \left|\frac{\phi(t+h) - \phi(t)}{h} - \phi^{(1)}(t)\right| + \left|\frac{\epsilon(t+h) - \epsilon(t)}{h}\right|. \tag{3.1.3}$$

We will call the error induced by the first term *truncation error* since it arises from truncation of the Taylor series, and the error induced by the second term *measurement error* due to error in the function evaluations.

Note that if h is small, then the truncation error is small but the measurement error may be large. On the other hand, if h is too large, the measurement error may be small but the truncation error may be too high. Therefore, the optimal h trades off these two terms by making the error from each of these two sources equal. In this paper, we propose an adaptive procedure for estimating the finite-difference interval in the presence of noise that properly balances these two different sources of error. The procedure must be: (1) reliable, that is, applicable to most, if not all, practical problems of interest; (2) accurate, producing near-optimal estimates of the finite-difference interval; and (3) efficient, employing the least number of function evaluations possible. We argue that our procedure achieves these goals in many practical situations.

This paper is organized into five sections. We present the notation and literature review in the rest of this section. In Section 2, we introduce our finite-difference interval estimation procedure for the forward-difference case. In Section 3, we present the generalized procedure for arbitrary finite-difference schemes and provide theoretical guarantees for the termination of our procedure. Extensive numerical results on synthetic problems with injected noise are provided in Section 4, and concluding remarks are made in Section 5.

3.1.1. Literature Review

The problem of estimating derivatives, particularly in the presence of rounding errors, is a fundamental question within numerical analysis and scientific computing. Fornberg proposed a stable algorithm for generating finite-difference formulas on arbitrarily spaced grids [38]. Lyness and Moler observed that the Cauchy integral theorem allows one to evaluate the d-th derivative of a complex function as a closed complex integral via numerical integration techniques [66]. This was simplified and extended by Squire and Trapp who observed that one could avoid cancellation error by using complex perturbations in the Taylor expansion, called complex step differentiation [96]. This has more recently led to extensions of the complex step to evaluating the Hessian by Hare and Srivastava [50]. Brekelmans, et al. compared design of experiments schemes against standard finite-difference schemes within the stochastic noise regime [14].

To handle rounding errors, Curtis and Reid describe a heuristic that estimates the truncation and rounding errors using central and forward-difference estimates. The ratio between the two estimates of these errors are used to determine the finite-difference interval [30]. Stepleman and Winarsky use a set of decreasing central-difference intervals. The

optimal interval is obtained by the smallest interval that does not violate monotonic decrease in the absolute difference between consecutive central-difference estimates [97]. Gill, Murray, Saunders, and Wright introduced an adaptive procedure for computing forward-difference intervals by utilizing a ratio to determine the second derivative [40, 41]. Their procedure has some similarities with our approach, which we discuss in Section 3.2.1. Barton proposed an adaptive procedure for handling rounding or multiplicative errors by interpreting the function values as correct up to a fixed number of significant digits and ensuring that at least a certain number of significant digits change from the resulting difference interval [6]. Most recently, Moré and Wild proposed a heuristic for estimating the second derivative for determining the forward-difference interval that checks two conditions: (1) if the noise dominates the second-order derivative; and (2) if the forward and backward difference is too large relative to the function values [71]. A comparison of the resulting errors between finite-difference and simplex gradients were analyzed in [9, 49].

Incorporating finite differences into optimization methods have also had a long history. Kiefer and Wolfowitz first applied finite differences to stochastic approximation [56]. Kelley developed an implicit filtering BFGS method that utilizes finite differences in the case where noise decays as the iterates converge to the solution [21, 55]. Berahas, et al. proposed a finite-difference L-BFGS method that incorporates `ECNoise` and a heuristic for estimating the second derivative by Moré and Wild into L-BFGS [7, 70, 71]. Most recently, Shi, et al. tested finite-difference methods within the unconstrained, least squares, and constrained settings assuming knowledge of the noise level [93].

3.1.2. Notation

In the following sections, we will use Bachmann-Landau notation liberally. Suppose $f, g : \mathbb{R} \to \mathbb{R}^{\geq 0}$. We will write $g(h) = \mathcal{O}(f(h))$ if there exists a $C \in \mathbb{R}$ such that $g(h) = Cf(h)$. If $g(h) = o(f(h))$, then for every $\epsilon > 0$ there exists a constant N such that $|g(h)| \leq \epsilon |f(h)|$ for all $h \leq N$. Similarly, if $g(h) = O(f(h))$, then there exists constants $\epsilon > 0$ and N such that $|g(h)| \leq \epsilon |f(h)|$ for all $h \leq N$.

We will use $\phi^{(d)} : \mathbb{R} \to \mathbb{R}$ to denote the d-th order derivative of ϕ. For a given vector $x \in \mathbb{R}^n$, $[x]_i$ denotes the i-th component of x. Similarly, for a given matrix $A \in \mathbb{R}^n$, $[A]_{ij}$ denotes the (i, j)-th entry of A. We will use $\| \cdot \|$ to denote the standard Euclidean norm unless otherwise specified.

3.2. An Adaptive Forward-Difference Interval Estimation Procedure

Suppose we are interested in determining the finite-difference interval for the forward-difference approximation of the first derivative of ϕ. Since the Taylor expansion of the function ϕ is given by

$$\phi(t + h) = \phi(t) + \phi^{(1)}(t)h + \frac{\phi^{(2)}(t)}{2}h^2 + o(h^2),$$

the total error can be bounded by

$$|\phi^{(1)}(t) - f^{(1)}(t; h)| \leq \underbrace{\frac{|\phi^{(2)}(t)|h}{2}}_{T_1} + \underbrace{\frac{2\epsilon_f}{h}}_{T_2} + o(h).$$

By ignoring the higher-order term, this yields an optimal interval (with respect to the upper bound) of

$$h^* \approx 2\sqrt{\frac{\epsilon_f}{|\phi^{(2)}(t)|}}. \tag{3.2.1}$$

This formula requires an estimate of the second derivative $|\phi^{(2)}(t)|$. We now propose a procedure that yields an interval $h = \mathcal{O}\left(\sqrt{\frac{\epsilon_f}{|\phi^{(2)}(t)|}}\right)$ without estimating $|\phi^{(2)}(t)|$ separately.

Our procedure balances the truncation T_1 and measurement error T_2. To do so, it estimates the ratio between these two errors directly and attempts to find an interval h for which this ratio is close to some constant value. We claim that the ratio T_1/T_2 of the truncation over measurement error can be approximated, for example, by the *testing ratio*

$$r(h; f, t, \epsilon_f) = \frac{|f(t+4h) - 4f(t+h) + 3f(t)|}{8\epsilon_f}. \tag{3.2.2}$$

Given $r_l > 1$ and $r_u > r_l + 2$, we perform a bisection search to find an interval $h > 0$ that satisfies

$$r(h; f, t, \epsilon_f) \in [r_l, r_u]. \tag{3.2.3}$$

In particular, if $r(h; f, t, \epsilon_f) < r_l$, then the numerator is dominated by noise, indicating that h is too small. On the other hand, if $r(h; f, t, \epsilon_f) > r_u$, then the numerator significantly dominates the noise, which implies that h is too large. Our procedure for forward differences is summarized in Algorithm 3.1.

Algorithm 3.1. Adaptive Forward-Difference Interval Estimation

Input: One-dimensional noisy function $f : \mathbb{R} \to \mathbb{R}$; noise level $\epsilon_f > 0$; lower- and upper-bound $(r_l, r_u) = (1.5, 6)$;

Output: Finite-difference interval h such that (3.2.3) holds.

$h \leftarrow \frac{2}{\sqrt{3}}\sqrt{\epsilon_f}$;

$l \leftarrow 0$, $u \leftarrow +\infty$;

while True do

    Evaluate $r(h; f, t, \epsilon_f) = \frac{|f(t+4h)-4f(t+h)+3f(t)|}{8\epsilon_f}$;

    if $r(h; f, t, \epsilon_f) < r_l$ **then**

        $l \leftarrow h$;

    else if $r(h; f, t, \epsilon_f) > r_u$ **then**

        $u \leftarrow h$;

    else

        break;

    end if

    if $u = +\infty$ **then**

        $h \leftarrow 4h$;

    else if $l = 0$ **then**

        $h \leftarrow h/4$;

    else

        $h \leftarrow (l + u)/2$;

    end if

end while

return h

To see why this procedure works to give us a near-optimal h, note that

$$\phi(t+4h) - 4\phi(t+h) + 3\phi(t) = 6\phi^{(2)}(t)h^2 + o(h^2). \tag{3.2.4}$$

Therefore, if we expand (3.2.2), we obtain:

$$r(h; f, t, \epsilon_f) = \left| \frac{3\phi^{(2)}(t)h^2}{4\epsilon_f} + \frac{\epsilon(t+4h) - 4\epsilon(t+h) + 3\epsilon(t)}{8\epsilon_f} + o(h^2) \right|. \tag{3.2.5}$$

Since $\left| \frac{\epsilon(t+4h)-4\epsilon(t+h)+3\epsilon(t)|}{8\epsilon_f} \right| \leq 1$ by the fact that $|\epsilon(t)| \leq \epsilon_f$ for all $t \in \mathbb{R}$, by imposing (3.2.3) and ignoring the $o(h^2)$ term, we approximately have

$$\frac{3|\phi^{(2)}(t)|h^2}{4\epsilon_f} \in [r_l - 1, r_u + 1] \iff h \in \frac{2}{\sqrt{3}} \sqrt{\frac{\epsilon_f}{|\phi^{(2)}(t)|}} \cdot [\sqrt{r_l - 1}, \sqrt{r_u + 1}]. \tag{3.2.6}$$

Therefore, if r_l and r_u are chosen properly, such as $r_l = 1.5$ and $r_u = 6$, we obtain $h \in [\sqrt{0.5}, \sqrt{7}] \cdot \sqrt{\frac{\epsilon_f}{|\phi^{(2)}(t)|}}$, which is the same order as the optimal finite-difference interval (3.2.1), differing only by a small constant factor.

By scaling h by a factor of 4 in Algorithm 3.1 when $u = \infty$ or $l = 0$, the new trial h only requires a single new function evaluation to check the testing ratio when h is updated monotonically.

In addition, the testing ratio is affine-invariant with respect to the function of interest in the sense that $r(h; f, t, \epsilon_f)$ remains unchanged if applied to a modified function $\tilde{f}(t) = af(t) + b$ for $a \neq 0$ and $b \in \mathbb{R}$ with noise level $|a|\epsilon_f$, i.e. $r(h; \tilde{f}, t, |a|\epsilon_f) = r(h; f, t, \epsilon_f)$. Therefore, the finite-difference interval h will correctly remain unchanged under this transformation.

3.2.1. Comparison to Prior Methods

Although the definition of the testing ratio appears similar to the ratio in Gill, et al. [40], which is defined as the inverse ratio

$$\frac{4\epsilon_f}{|f(t+\tilde{h}) - 2f(t) + f(t-\tilde{h})|}, \tag{3.2.7}$$

our approach markedly differs from theirs in three aspects: (1) we utilize the h derived from Algorithm 3.1 directly as the chosen difference interval, whereas Gill, et al. use the difference interval $\tilde{h}$ to estimate the second derivative; (2) our approach utilizes a bisection search to find h rather than monotonically increasing or decreasing h; and (3) it relies on second-order forward-difference instead of central-difference estimates.

The first aspect follows from the observation made in (3.2.6) that $h = \mathcal{O}\left(\sqrt{\frac{\epsilon_f}{|\phi^{(2)}(t)|}}\right)$. Hence, the difference interval h obtained by our bisection procedure is near-optimal in that it is only a small constant factor away from the optimal interval (except for certain cases discussed below). Two observations can be made from this derivation. The first is that since $f(t+h)$ and $f(t)$ are used in the evaluation of the testing ratio, one can reuse prior function evaluations from the bisection procedure to estimate $f^{(1)}(t;h)$. The second observation is that the derivation motivates an initial choice of $h = \mathcal{O}(\sqrt{\epsilon_f})$ as opposed to $h = \mathcal{O}(\sqrt[4]{\epsilon_f})$, as used in Moré and Wild [71]. This is corroborated by our experiments in Section 3.4.

The second aspect follows from different tradeoffs between cost and accuracy in the finite-difference interval estimate. In particular, whereas Gill, et al.'s procedure is cheaper

but yields a less accurate estimate, our procedure is able to guarantee a sufficiently accurate estimate at higher cost.

Lastly, the third aspect is designed to avoid cancellation due to symmetry in the numerator of the testing ratio. In particular, Gill, et al. [40] rely on a testing ratio using the central difference:

$$\frac{4\epsilon_f}{|f(t+\tilde{h}) - 2f(t) + f(t-\tilde{h})|} \in [0.001, 0.1]. \tag{3.2.8}$$

However, we can obtain poor estimates of the derivative due to cancellation under symmetry of the function, for example, on a quartic function $\phi(t) = t^4$ near (but not at) $t = 0$ with sufficiently large noise.

Our procedure also differs from Moré and Wild's procedure [71]. Their procedure estimates the second derivative by

$$\phi^{(2)}(t) \approx \frac{f(t+\tilde{h}) - 2f(t) + f(t-\tilde{h})}{\tilde{h}^2} = f^{(2)}(t;\tilde{h}), \tag{3.2.9}$$

with interval $\tilde{h} > 0$, then inserts this estimate into the optimal formula (3.2.6). The difference interval $\tilde{h}$ is required to satisfy

$$|f(t+\tilde{h}) - 2f(t) + f(t-\tilde{h})| \geq \tau_1 \epsilon_f \tag{3.2.10}$$

$$|f(t \pm \tilde{h}) - f(t)| \leq \tau_2 \max\{|f(t)|, |f(t \pm \tilde{h})|\} \tag{3.2.11}$$

with $\tau_1 \gg 1$ and $\tau_2 \in (0, 1)$. Their method attempts to satisfy this within two trials as follows:

(1) Set $\tilde{h}_1 = \sqrt[4]{\epsilon_f}$ and compute $\mu_1 = |f^{(2)}(t; \tilde{h}_1)|$. If conditions (3.2.10) and (3.2.11) are satisfied for $\tilde{h}_1$, return μ_1.

(2) Set $\tilde{h}_2 = \sqrt[4]{\epsilon_f/\mu_1}$ and compute $\mu_1 = |f^{(2)}(t; \tilde{h}_2)|$. If conditions (3.2.10) and (3.2.11) are satisfied for $\tilde{h}_2$, return μ_2.

(3) If $|\mu_1 - \mu_2| \leq \frac{1}{2}\mu_2$, return μ_2.

If the heuristic is unable to return an estimate μ after two trials, this is considered as a failure.

Similar to Gill's procedure, this method differs from ours in that it estimates the second derivative rather than utilizing the h derived from the testing ratio directly. As a result, it initializes $\tilde{h} = \mathcal{O}(\sqrt[4]{\epsilon_f})$ rather than $\mathcal{O}(\sqrt{\epsilon_f})$. We have found this to be a poor initial choice of $\tilde{h}$, particularly when ϵ_f is large, as we will see in Section 3.4.

While (3.2.10) appears similar to the testing ratio, it is better interpreted as ensuring that noise does not dominate the second-derivative estimation due to the large choice of $\tau_1 = 100$. The second condition (3.2.11) is not affine-invariant in the sense that adding a sufficiently large b can force the condition to be satisfied. This is undesirable as perturbations of the function should not change the overall behavior of the method.

3.3. Generalized Finite-Difference Interval Estimation

Typically, finite-difference interval estimation procedures for numerical optimization focus on forward differences [6, 40, 71]. However, in the noisy regime, higher-order finite-difference approximations, such as central differences, can yield more accurate approximations; see [93]. As a result, in order to attain the highest possible accuracy, one must design methods that efficiently find a near-optimal difference interval for more

general finite-difference schemes. To handle this, we propose a generalization of the forward-difference case, Algorithm 3.1, for d-th order derivatives.

Consider a finite-difference approximation scheme $S = (w, s)$ defined over m points, where we approximate $\phi^{(d)}(t)$ using the equation

$$f_S^{(d)}(t; h) = \frac{\sum_{j=1}^{m} w_j \cdot f(t + hs_j)}{h^d} \approx \phi^{(d)}(t) \tag{3.3.1}$$

where $w \in \mathbb{R}^m$ and $s \in \mathbb{R}^m$ are the associated weights and shifts of the finite-difference scheme. As in the forward-difference setting, we will use a slight abuse of notation by denoting the finite-difference approximation as $f_S^{(d)}(t; h)$. *Forward-difference* and *central-difference* schemes for approximating the first derivative (i.e., $d = 1$) are obtained by defining s and w as $s = (0, 1)^T$ and $w = (-1, 1)^T$ and $s = (-1, 1)^T$ and $w = (-\frac{1}{2}, \frac{1}{2})^T$, respectively. The standard *second-order central-difference* scheme is defined as $s = (-1, 0, 1)^T$ and $w = (1, -2, 1)^T$.

In order for the finite-difference scheme to be valid, the coefficients w and shifts s must be chosen such that the Taylor expansion of the finite-difference approximation over the function ϕ satisfies

$$\sum_{j=1}^{m} w_j \cdot \phi(t + hs_j) = \phi^{(d)}(t)h^d + c_q\phi^{(q)}(t)h^q + o\left(h^q\right), \tag{3.3.2}$$

where $q \geq d+1$ denotes the order of the remainder term[1]. This ensures that $f_S^{(d)}(t;h) \approx \phi^{(d)}(t)$. In order to guarantee this, the finite-difference scheme S must satisfy

$$\frac{1}{l!}\sum_{j=1}^{m} w_j s_j^l = \begin{cases} 1 & \text{if } l = d \\ 0 & \text{if } l < q,\ l \neq d \end{cases}$$

and as a result

$$c_q = \frac{1}{q!}\sum_{j=1}^{m} w_j s_j^q.$$

See Appendix B.1 for more detail on how generic finite-difference schemes are derived.

Therefore, in the presence of noise, the worst-case error for the finite-difference scheme of interest can be bounded by

$$|f^{(d)}(t;h) - \phi^{(d)}(t)| \leq |c_q| \left|\phi^{(q)}(t)\right| h^{q-d} + \|w\|_1 \epsilon_f h^{-d} + o\left(h^{q-d}\right).$$

One can define an approximately optimal choice of h:

$$h^* \approx \left| \frac{d}{q-d} \cdot \frac{\|w\|_1 \epsilon_f}{c_q \phi^{(q)}(t)} \right|^{1/q}. \tag{3.3.3}$$

While ϵ_f is assumed to be known and d, q, w and c_q are available, the q-th order derivative $\phi^{(q)}(x)$ is unknown and often difficult to estimate. Following the idea from the forward-difference case, we propose a procedure for estimating (3.3.3) directly. We first construct a testing ratio r_S associated with scheme S:

$$r_S(h; f, t, \epsilon_f) = \frac{\left|\sum_{j=1}^{\tilde{m}} \tilde{w}_j \cdot f(t + h\tilde{s}_j)\right|}{\epsilon_f}$$

[1]Note that the order of accuracy can be higher than $d+1$ for certain schemes, such as central-difference approximations.

where $\tilde{w}, \tilde{s} \in \mathbb{R}^{\tilde{m}}$ where $\tilde{m} \geq q+1$, $\tilde{s}_j \neq \tilde{s}_k$ for all $j \neq k$, and $\tilde{w}$ and $\tilde{s}$ satisfies

$$\sum_{j=1}^{\tilde{m}} \tilde{w}_j \cdot \phi(t + h\tilde{s}_j) = c_r \phi^{(q)}(t) h^q + o\left(h^q\right), \; c_r = \frac{1}{q!} \sum_{j=1}^{\tilde{m}} \tilde{w}_j \tilde{s}_j^q \neq 0. \tag{3.3.4}$$

Without loss of generality, we require $\tilde{w}$ to satisfy

$$\|\tilde{w}\|_1 = 1,$$

the reason for which will be evident below. We then perform a bisection search to find an interval h that satisfies

$$r_S(h; f, t, \epsilon_f) \in [r_l, r_u] \tag{3.3.5}$$

for some $r_l > 1$ and $r_u > r_l + 2$. The procedure is summarized in Algorithm 3.2.

Algorithm 3.2. Adaptive Finite-Difference Interval Estimation

$h \leftarrow h_0$;
$l \leftarrow 0$, $u \leftarrow +\infty$;
while True do
    Evaluate $r_S(h; f, t, \epsilon_f)$;
    if $r_S(h; f, t, \epsilon_f) < r_l$ **then**
        $l \leftarrow h$;
    else if $r_S(h; f, t, \epsilon_f) > r_u$ **then**
        $u \leftarrow h$;
    else
        break;
    end if
    if $u = +\infty$ **then**
        $h \leftarrow \eta h$;
    else if $l = 0$ **then**
        $h \leftarrow h/\eta$;
    else
        $h \leftarrow (l + u)/2$;
    end if
end while
return h

By (3.3.4), we can see that

$$r_S(h; f, t, \epsilon_f) = \left| \frac{c_r \phi^{(q)}(t) h^q}{\epsilon_f} + \frac{\sum_{j=1}^{\tilde{m}} \tilde{w}_j \cdot \epsilon(t + h\tilde{s}_j)}{\epsilon_f} + o\left(h^q\right) \right| \tag{3.3.6}$$

$$= \left| \frac{c_r \phi^{(q)}(t) h^q}{\epsilon_f} + \Delta + o\left(h^q\right) \right|, \ \Delta \triangleq \frac{\sum_{j=1}^{\tilde{m}} \tilde{w}_j \cdot \epsilon(t + h\tilde{s}_j)}{\epsilon_f}. \tag{3.3.7}$$

Note that by definition of Δ, $|\Delta| \leq 1$. This is a consequence of the requirement that $\|\tilde{w}\|_1 = 1$ and that $|\epsilon(t)| \leq \epsilon_f$ for all $t \in \mathbb{R}$. Therefore, if we have $r_S(h; f, t, \epsilon_f) \in [r_l, r_u]$ and if we the ignore $o\left(h^q\right)$ term, then we (approximately) have

$$\left| \frac{c_r \phi^{(q)}(t) h^q}{\epsilon_f} \right| \in [r_l - 1, r_u + 1], \tag{3.3.8}$$

i.e.,

$$h \in \left[\left(\frac{r_l - 1}{|c_r|} \frac{\epsilon_f}{|\phi^{(q)}(t)|} \right)^{1/q}, \left(\frac{r_u + 1}{|c_r|} \frac{\epsilon_f}{|\phi^{(q)}(t)|} \right)^{1/q} \right]. \tag{3.3.9}$$

Note from (3.3.3) that h has the same dependence on ϵ_f and $\phi^{(q)}(t)$ as h^*. As in the forward-difference case, our algorithm is invariant to affine transformations with respect to the function.

Example 1 (First-Order Central Difference). Consider the first-order central-difference scheme for approximating the first derivative:

$$f_S^{(1)}(t; h) = \frac{f(t + h) - f(t - h)}{2h}, \tag{3.3.10}$$

where $s = (-1, 1)^T$ and $w = (-\frac{1}{2}, \frac{1}{2})^T$. The Taylor expansion of the numerator is given as:

$$\frac{\phi(t+h) - \phi(t-h)}{2h} = \phi^{(1)}(t) + \frac{\phi^{(3)}(t)h^2}{6} + o(h^2).$$

The full error of the derivative approximation and the approximate optimal choice of h are:

$$\left| f_S^{(1)}(t;h) - \phi^{(1)}(t) \right| \leq \frac{\left|\phi^{(3)}(t)\right| h^2}{6} + \frac{\epsilon_f}{h} + o(h^2),\ h^* \approx \sqrt[3]{\frac{3\epsilon_f}{|\phi^{(3)}(t)|}}.$$

One example of a valid testing ratio is:

$$r_S(h; f, t, \epsilon_f) = \frac{|f(t+3h) - 3f(t+h) + 3f(t-h) - f(t-3h)|}{8\epsilon_f}. \tag{3.3.11}$$

Example 2 (Second-Order Central Difference). Consider the second-order central-difference scheme for approximating the second derivative:

$$f_S^{(2)}(t;h) = \frac{f(t+h) - 2f(t) + f(t-h)}{h^2}, \tag{3.3.12}$$

where $s = (-1, 0, 1)^T$ and $w = (-\frac{1}{2}, \frac{1}{2})^T$. The Taylor expansion of the numerator is given as:

$$\frac{\phi(t+h) - 2\phi(t) + \phi(t-h)}{h^2} = \phi^{(2)}(t) + \frac{\phi^{(4)}(t)h^2}{24} + o(h^2).$$

The full error of the derivative approximation and the approximate optimal choice of h are:

$$\left| f_S^{(2)}(t;h) - \phi^{(2)}(t) \right| \leq \frac{\left|\phi^{(4)}(t)\right| h^2}{24} + \frac{4\epsilon_f}{h^2} + o(h^2),\ h^* \approx 2\sqrt[4]{\frac{6\epsilon_f}{|\phi^{(4)}(t)|}}.$$

One example of a valid testing ratio is:

$$r_S(h; f, t, \epsilon_f) = \frac{|f(t+2h) - 4f(t+h) + 6f(t) - 4f(t-h) + f(t-2h)|}{16\epsilon_f}. \tag{3.3.13}$$

3.3.1. Practical Considerations

In order to make the procedure both efficient and robust, we discuss a number of practical considerations below.

I. Generation of Testing Ratio. Although many choices of r_S are possible for any finite-difference scheme S, it would be useful to have a method for automatically generating valid testing ratios that efficiently utilize function values. A simple yet useful way to construct r_S is through the formula

$$r_S^\alpha(h; f_S, t, \epsilon_f) = \frac{\left|\left(f_S^{(d)}(t; h) - \alpha^{-d} f_S^{(d)}(t; \alpha h)\right) h^d\right|}{A\epsilon_f}, \tag{3.3.14}$$

where $\alpha \neq 1$ and A is computed by normalizing the coefficients such that $\|\tilde{w}\|_1 = 1$ is satisfied.

This approach is guaranteed to generate a valid testing ratio r_S for any $\alpha \neq 1$ since it cancels out the $\phi^{(d)}(t)$ term in the Taylor expansion, leaving only the relevant higher-order term of order q of interest. In particular, since

$$\sum_{j=1}^{m} w_j \cdot \phi(t + hs_j) = \phi^{(d)}(t)h^d + c_q\phi^{(q)}(t)h^q + o(h^q)$$

$$\sum_{j=1}^{m} w_j \cdot \phi(t + \alpha hs_j) = \phi^{(d)}(t)(\alpha h)^d + c_q\phi^{(q)}(t)(\alpha h)^q + o(h^q),$$

we obtain that

$$\left(f_S^{(d)}(t;h) - \alpha^{-d} f_S^{(d)}(t;\alpha h)\right) h^d = \sum_{j=1}^{m} \left(w_j \cdot \phi(t + hs_j) - \alpha^{-d} w_j \cdot \phi(t + \alpha h s_j)\right)$$
$$= c_q(1 - \alpha^{q-d})\phi^{(q)}(t)h^q + o(h^q),$$

which satisfies (3.3.4) with an effective $c_r = c_q(1 - \alpha^{q-d})/A$ as desired.

For finite-difference schemes with equidistant points, a small modification to the bisection search allows us to reuse 1 – 2 function evaluations at each iteration, depending on the original scheme S. To do this, we can multiply or divide by the same factor $\eta = \alpha$ when monotonically increasing or decreasing h within the bisection search, as done with $\eta = 4$ in the forward-difference algorithm (Algorithm 3.1).

In addition, with this choice of the testing ratio, the function evaluations needed within the finite-difference scheme is implicitly evaluated within the testing ratio. We can therefore obtain the finite-difference approximation using previously computed function values at no additional cost.

II. Choice of r_l and r_u. Ideally, one should choose r_l and r_u such that they are close to the optimal ratio

$$r^* = \frac{d}{q-d} \cdot \left|\frac{c_r}{c_q}\right| \cdot \|w\|_1$$

in order to yield an h that is close to h^* in (3.3.3). However, this is not directly possible in the presence of noise, which requires that $1 < r_l < r_u - 2$ in order to ensure finite-termination; see Section 3.3.2. We therefore select (r_l, r_u) sufficiently large such that $1 < r_l < r_u - 2$ and, if possible, such that r^* is logarithmically centered within the interval

$[r_l, r_u]$:

$$r_l = \max\left\{1+\eta, \frac{r^*}{\beta}\right\}, \; r_u = \max\left\{3(1+\eta), \beta r^*\right\} \tag{3.3.15}$$

for some $\eta > 0$ and $\beta > 1$. (In our experiments, we set $\eta = 0.1$ and $\beta = 2$.)

Note that when $r_l, r_u > r^*$, the algorithm may overestimate $|\phi^{(q)}(t)|$ and hence underestimate h. In order to avoid this in practice, we have found that it is preferable to choose a testing ratio such that the optimal ratio $r^* \geq \beta(1+\eta)$. This could be done by choosing a different testing ratio, such as by choosing a larger α in (3.3.14).

III. Initialization of h_0. Since the difference interval h that satisfies the procedure is approximately of the form (3.3.9), it is preferable to initialize $h_0 = \mathcal{O}(\epsilon_f^{1/q})$. Two possible choices are $h_0 = \epsilon_f^{1/q}$ or $\left(\frac{d}{q-d} \cdot \frac{\|w\|_1}{|c_q|} \cdot \epsilon_f\right)^{1/q}$. The latter is based on the assumption that $|\phi^{(q)}(t)| \approx 1$. If instead the finite-difference interval is re-estimated within an optimization algorithm, we can initialize h_0 as the difference interval h used at the prior outer iteration of the algorithm.

We observe that on rare occasions a poor initial choice of h_0 can result in large error in the derivative approximation. This occurs when the initial choice of h_0 is too large to capture the local behavior of the function. Reducing the initial interval h_0 resolves this issue.

IV. Handling of Special Cases. The Taylor expansion analysis elucidates two possible failure cases for our procedure. In particular, observe that

$$r_S(h; f, t, \epsilon_f) = \left| \frac{c_r \phi^{(q)}(t) h^q}{\epsilon_f} + \Delta + o(h^q) \right|.$$

If h is large (for example, when the noise level ϵ_f is high), the higher-order terms $o(h^q)$ can dominate the other terms in the Taylor series expansion. This can yield poor estimates of h even if the condition $r_S(h; f, t, \epsilon_f) \in [r_l, r_u]$ is satisfied. In practice, we have not found this to be a common issue.

The more common case is when $\phi^{(q)}(t) \approx 0$. In this case, r_S will be dominated by Δ. In this case, $r_S(h; f, t, \epsilon_f) < r_l$ for all h and h will thus monotonically increase until the maximum number of iterations is reached (which we set `max_iter` to 20). This occurs, for example, with any $(q-1)$-th degree polynomial. In this case, the method provides a warning but does not flag this as a failure. Note that in this case, h is a good choice because sending $h^* \to \infty$ would allow for indefinite reduction in the noise.

V. Extension to Standard Deviation. In some settings, we only have access to the standard deviation of the noise, where $\epsilon(x)$ is modeled as a random variable. Assuming $\mathbb{E}[\epsilon(x)] = 0$, one can extend this procedure to the stochastic setting by replacing ϵ_f with $\sigma_f = \sqrt{\mathbb{E}[\epsilon(x)^2]}$. While finite termination (see next subsection) is not guaranteed, if the procedure succeeds, it will yield an h that has the same dependence on σ_f and $|\phi^{(2)}(t)|$ as the optimal finite-difference interval with respect to its mean-squared error.

VI. Error Bound Estimate on Gradient. Using the h we obtain from our procedure, we can approximate the error bound. Ignoring the $o(h^{q-d})$ term, the error is approximately given by:

$$\epsilon_g(t; h) \approx |c_q| \left|\phi^{(q)}(t)\right| h^{q-d} + \|w\|_1 \epsilon_f h^{-d}.$$

As we have shown in (3.3.8), we can bound $\left|\phi^{(q)}(t)\right|$ by

$$\left|\phi^{(q)}(t)\right| \leq \frac{\epsilon_f(r_u+1)}{|c_r|\, h^q}.$$

Therefore, we can approximately bound the error by

$$\epsilon_g(t;h) \lesssim \left(\frac{|c_q|}{|c_r|}(r_u+1) + \|w\|_1\right)\epsilon_f h^{-d}.$$

In practice, we have found that this error bound is able to obtain an order-of-magnitude of the actual error, but may underestimate the error due to the $o(h^{q-d})$ term.

3.3.2. Finite Termination

Next, we prove a finite termination theorem for Algorithm 3.2. We start by making the following assumptions:

Assumption 3.3.1. *The testing ratio r_S satisfies:*

$$|r_S(h;\phi,t,\epsilon_f) - r_S(h;f,t,\epsilon_f)| \leq 1,\ \forall t \in \mathbb{R},\ h > 0$$

This assumption is satisfied by our requirement that $\|\tilde{w}\|_1 = 1$ and that $|\epsilon(t)| \leq \epsilon_f$. Note that this can be easily satisfied by simply rescaling the numerator to ensure that the total noise accumulated in the numerator is bounded by ϵ_f.

Assumption 3.3.2. *As a function of h, $r_S(h;\phi,t,\epsilon_f)$ is continuous with $r_S(0;\phi,t,\epsilon_f) = 0$, and there exists an integer $K \in \mathbb{N}$ such that*

$$r_S(2^K h_0;\phi,t,\epsilon_f) \geq r_u - 1$$

Assuming that $\epsilon_f > 0$, the requirement that $r_S(0; \phi, t, \epsilon_f) = 0$ is automatically satisfied by validity of the testing ratio (3.3.4). The second part of Assumption 3.3.2, while technical, is satisfied, for example, when $\left|\phi^{(q)}(\xi)\right| \geq \eta > 0$ for all $\xi \in [\min_j\{t + h\tilde{s}_j\}, \max_j\{t + h\tilde{s}_j\}]$. With these assumptions, we can now show finite termination.

Theorem 3.3.3. *Suppose Assumptions 3.3.1 and 3.3.2 are satisfied. In addition, suppose r_u and r_l are chosen such that $0 < r_l < r_u - 2$. Then, Algorithm 3.2 will terminate successfully in a finite number of iterations.*

Proof. We assume that $h \geq 0$. Assume by contradiction that Algorithm 3.2 does not terminate finitely. We denote the variables l, u, h used *at the beginning of* the k-th iteration of Algorithm 3.2 as l_k, u_k, h_k, respectively. Obviously, we have

$$0 \leq l_k \leq h_k \leq u_k, \ \forall k \in \mathbb{N},$$

and

$$l_k \leq l_{k+1} < u_{k+1} \leq u_k, \ \forall k \in \mathbb{N}.$$

First, we show that $r_S(l_k; \phi, t, \epsilon_f) < r_l + 1$ for all $k \in \mathbb{N}$, by induction on k. Clearly this is true for $k = 0$ since $l_0 = 0$, and we have $r_S(0; \phi, t, \epsilon_f) = 0$ by Assumption 3.3.2. Suppose the statement holds for $k \leq K$. We have two cases: (1) $r_S(h_K; f, t, \epsilon_f) < r_l$, which by Assumption 3.3.1 implies $r_S(h_K; \phi, t, \epsilon_f) \leq r_S(h_K; f, t, \epsilon_f) + 1 < r_l + 1$. In this case $l_{K+1} = h_K$, so $r_S(l_{K+1}; \phi, t, \epsilon_f) = r_S(h_K; \phi, t, \epsilon_f) < r_l + 1$. (2) $r_S(h_K; f, t, \epsilon_f) > r_u$, in which case $l_{K+1} = l_K$ so by the induction hypothesis $r_S(l_{K+1}; \phi, t, \epsilon_f) = r_S(l_K; \phi, t, \epsilon_f) < r_l + 1$. Therefore the induction hypothesis holds for $(K + 1)$-th iteration.

By a similar argument, we can show that either $u_k = +\infty$, or $u_k < +\infty$ and $r_S(u_k; \phi, t, \epsilon_f) > r_u - 1$ for all $k \in \mathbb{N}$.

In summary, we can show that for all $k \in \mathbb{N}$, we have

$$\text{either } r_S(l_k; \phi, t, \epsilon_f) < r_l + 1 < r_u - 1 < r_S(u_k; \phi, t, \epsilon_f), \tag{3.3.16}$$

$$\text{or } r_S(l_k; \phi, t, \epsilon_f) < r_l + 1 \text{ and } u_k = +\infty. \tag{3.3.17}$$

Next, we claim that there exists $K_1 \in \mathbb{N}$ such that $u_k < +\infty$ for $k \geq K_1$. Suppose this is not the case, then we have $r_S(h_k; f, t, \epsilon_f) < r_l, \ \forall k \in \mathbb{N}$. In this case, we have $h_{k+1} = 2l_{k+1} = 2h_k$, so $h_k = 2^k h_0$ for all $k \in \mathbb{N}$. By Assumption 3.3.2, there exists $K \in \mathbb{N}$ such that $r_S(h_K; \phi, t, \epsilon_f) \geq r_u - 1$, and since $r_S(h_K; f, t, \epsilon_f) \geq r_S(h_K; \phi, t, \epsilon_f) - 1$, we have $r_S(h_K; f, t, \epsilon_f) \geq r_u - 2 > r_l$, contradicting the inequality $r_S(h_k; f, t, \epsilon_f) < r_l, \ \forall k \in \mathbb{N}$. This proves the existence of K_1.

We are now ready to present the contradiction. For $k \geq K_1$, since $u_k < \infty$, we have

$$u_{k+1} - l_{k+1} = \frac{1}{2}(u_k - l_k)$$

This implies that $u_k - l_k \to 0$. Since $r_S(h; \phi, t, \epsilon_f)$ (as a function of h) is continuous and $u_{K_1} < +\infty$, $[0, u_{K_1}]$ is compact so $r_S(h; \phi, t, \epsilon_f)$ (as a function of h) is *uniformly continuous* on $[0, u_{K_1}]$. Note that $l_k, u_k \in [0, u_{K_1}]$ for $k \geq K_1$, therefore we have

$$r_S(u_k; \phi, t, \epsilon_f) - r_S(l_k; \phi, t, \epsilon_f) \to 0$$

This contradicts the fact that

$$r_S(l_k; \phi, t, \epsilon_f) < r_l + 1 < r_u - 1 < r_S(u_k; \phi, t, \epsilon_f), \ \forall k \in \mathbb{N}, \ k \geq K_1$$

Therefore, Algorithm 3.2 must terminate finitely. Clearly, whenever it terminates, the output h_R must satisfy

$$r_S(h_R; f, t, \epsilon_f) \in [r_l, r_u].$$

□

3.4. Numerical Experiments

In this section, we present numerical results demonstrating the reliability of our finite-difference interval estimation procedure. We first utilize the method for computing first and second derivatives of commonly tested functions, with added noise. We then insert our procedure into a standard L-BFGS implementation and demonstrate its usefulness on a subset of synthetic noisy `CUTEst` problems [42]. All methods were implemented in Python 3.

3.4.1. Finite-Difference Interval Estimation

We first test our proposed procedure on several univariate functions. We focus on the case where $d = 1$ and 2 as this is most relevant to optimization. We test Algorithm 3.2 using 6 different estimating schemes, shown in Table 3.1. The testing ratios are generated using formula (3.3.14) with different choices of α. The α for each scheme is chosen as the smallest integer such that $r^* > \beta = 2$.

label	d	s	w	q	α	r^*	Comment
`FD`	1	$(0,1)$	$(-1,1)$	2	4	3	forward difference
`CD`	1	$(-1,1)$	$(-1/2,1/2)$	3	3	3	central difference
`FD_3P`	1	$(0,1,2)$	$(-3/2,2,-1/2)$	3	3	3.69	forward difference w/ 3 points
`FD_4P`	1	$(0,1,2,3)$	$(-11/6,3,-3/2,1/1)$	4	3	8.25	forward difference w/ 4 points
`CD_4P`	1	$(-2,-1,1,2)$	$(1/12,-2/3,2/3,-1/12)$	5	2	2.5	central difference w/ 4 points
`L2_CD`	2	$(-1,0,1)$	$(1,-2,1)$	4	2	3	2nd-order central difference

Table 3.1. Schemes for approximating the d-th order derivative used in the experiments. The scheme is defined by $S = (w, s)$ as in (3.3.1); q is defined in (3.3.2).

For a specific testing function ϕ at point t with noise ϵ_f and scheme $S = (w, s)$, we plot the *worst case relative error*, as a function of differencing interval h, defined as:

$$\delta_S(h;\phi,t,\epsilon_f) = \frac{1}{|\phi^{(d)}(t)|}\left[\left|\frac{\sum_{j=1}^{p} w_j\phi(t+s_jh)}{h^d} - \phi^{(d)}(t)\right| + \|w\|_1\frac{\epsilon_f}{h^d}\right].$$

This function captures the relative error of the estimation scheme S on the noisy function f at t, in the worst case. The differencing interval h that minimizes $\delta_S(h;\phi,t,\epsilon_f)$ is the optimal h. Notice that $\delta_S(h;\phi,t,\epsilon_f)$ is a deterministic function that does not rely on the realization of actual noise in $f(t)$.

We manually inject uniformly distributed, stochastic noise into ϕ,

$$f(t) = \phi(t) + \epsilon(t),\ \epsilon(t) \sim \text{Uniform}(-\epsilon_f, \epsilon_f),$$

independent of all other quantities. We then apply Algorithm 3.2 to obtain $h_\dagger$. We plot $h_\dagger$ and observe how far it is from the minimizer of $\delta_S(h;\phi,t,\epsilon_f)$. In Appendix B.2, we report the minimizer of the function $\delta_S(h;\phi,t,\epsilon_f)$ obtained by `scipy.optimize.minimize_scalar`.

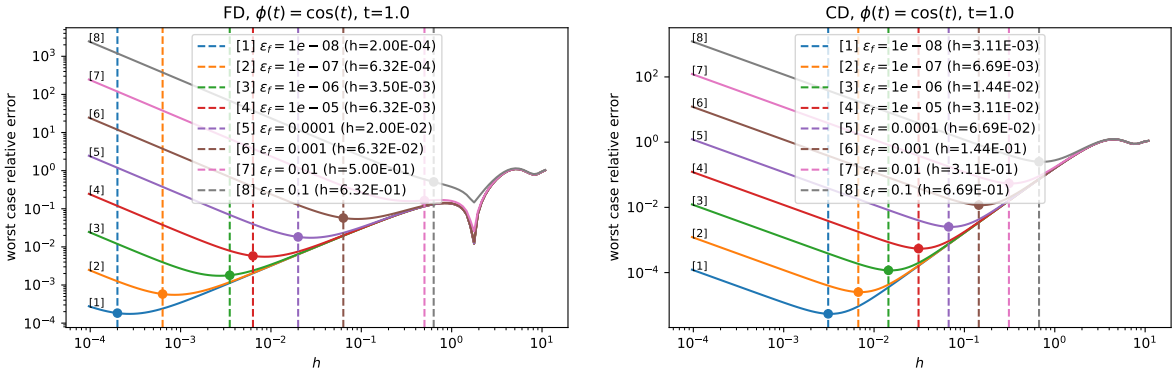

Figure 3.1. Worst case relative error $\delta_S(h; \phi, t, \epsilon_f)$ for forward and central differences against h on function $\phi(t) = \cos(t)$ with different noise levels; the vertical dashed line represents the $h_\dagger$ output by Algorithm 3.2.

3.4.1.1. Robustness to Different Noise Levels. To demonstrate that our method is reliable across a range of noise levels, we test our adaptive procedure for both forward and central differences (`FD, CD`) on the simple function $\phi(t) = \cos(t)$ for different noise levels. The plot of the worst case relative error and obtained interval $h_\dagger$ are illustrated in Figure 3.1. Our results demonstrate that our method performs consistently well across a range of noise levels. For all figures on all finite-difference schemes listed in Table 3.1 and complete numerical results, see Appendix B.2.

3.4.1.2. Difficult and Special Examples. In this subsection, we consider examples of difficult functions given in [40] and [93]. These examples include:

(1) $\phi(t) = (e^t - 1)^2$, at $t = -8$. This function has extremely small first and second-order derivative at $t = -8$, but quickly increases as t increases beyond $t = 0$; a naive choice of $h = \sqrt{\epsilon_f / |\phi^{(2)}(t)|}$ for forward differences can result in an extremely large h and lead to huge error.

(2) $\phi(t) = e^{100t}$, at $t = 0.01$.

(3) $\phi(t) = t^4 + 3t^2 - 10t$, at $t = 0.99999$. This function is considered difficult because $\phi'(1) = 0$, and represents a case where the estimated derivative is very close to 0. In addition, this function is a fourth-order polynomial, so the optimal h for `CD_4P` is $+\infty$.

(4) $\phi(t) = 10000t^3 + 0.01t^2 + 5t$, at $t = 10^{-9}$. This example is difficult in that it has approximate central symmetry at $t = 0$, which can lead to issues for adaptive procedures such as those proposed in [40].

For each example, we again fix $\epsilon_f = 10^{-3}$, and perform our estimation procedure for different schemes, and plot the worst case relative error. The results can be found in Figure 3.2.

For the first two examples, we see that our procedure is able to estimate the derivative well even when the function increases rapidly. These are cases where using our adaptive procedure is significantly more effective than computing an interval based on higher-order derivative information at the point of interest, as observed in [93].

It is also interesting to observe the results for the two polynomials. For $\phi(t) = t^4 + 3t^2 - 10t$ and scheme `CD_4P`, our procedure generates a large $h_\dagger$; this is consistent with the fact that scheme `CD_4P` has $q = 5$, and $\phi^{(5)}(\xi) = 0$ for all ξ on this example, which implies that we should choose h to be as large as possible. This similarly holds true for the schemes `FD_4P` and `CD_4P` on the function $\phi(t) = 10000t^3 + 0.01t^2 + 5t$.

While theoretically speaking we should choose $h = \infty$ in such cases, we can observe in Figure 3.2 that this is not the case. When plotting the worst-case relative error, we see that there exists a large h such that $\delta_S(h; \phi, t, \epsilon_f)$ is minimized, beyond which the relative error begins to sharply increase. This phenomenon is due to round-off error. When h becomes

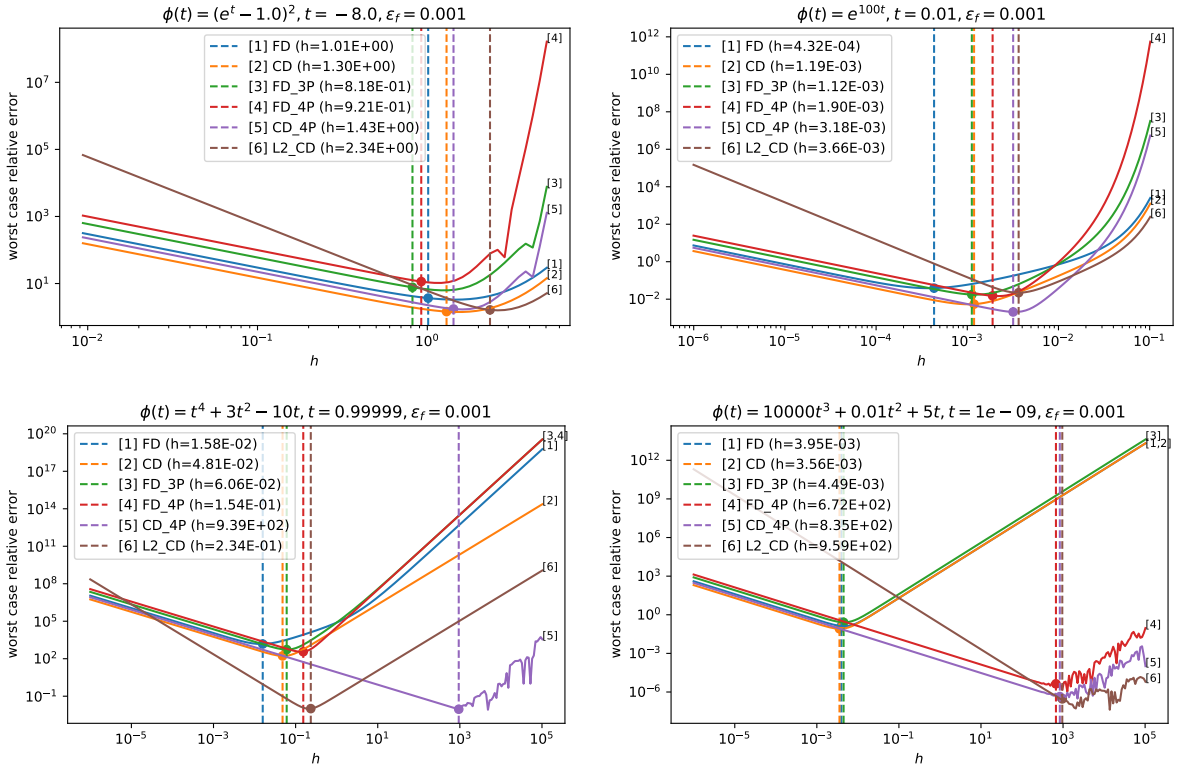

Figure 3.2. Worst case relative error $\delta_S(h; \phi, t, \epsilon_f)$ against h on several special cases; the vertical dashed line represents the $h_\dagger$ output by Algorithm 3.2.

too large, round-off error (which is multiplicative) will dominate ϵ_f; this approximately happens when $\max_j |\phi(t + s_j h)| \, \epsilon_M$ becomes comparable to ϵ_f.

3.4.2. Finite-Difference L-BFGS

In order to show the robustness of our procedure, we apply it within the L-BFGS method. We now let ϕ denote a smooth multivariate function, $\phi : \mathbb{R}^n \to \mathbb{R}$, and consider the problem

$$\min_{x \in \mathbb{R}^n} \phi(x), \tag{3.4.1}$$

while only provided noisy function evaluations of the form

$$f(x) = \phi(x) + \epsilon(x),\ \epsilon(x) \sim \text{Uniform}(-\epsilon_f, \epsilon_f), \tag{3.4.2}$$

where $\epsilon_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$. We perform our tests on a subset of synthetically generated noisy `CUTEst` problems [42] detailed in Table 3.2.

Problem	Dim (n)	Problem	Dim (n)	Problem	Dim (n)	Problem	Dim (n)
AIRCRFTB	5	CRAGGLVY	100	FREUROTH	100	PFIT4LS	3
ALLINITU	4	CUBE	2	GENROSE	100	QUARTC	100
ARWHEAD	100	DENSCHND	3	GULF	3	SINEVAL	2
BARD	3	DENSCHNE	3	HAIRY	2	SINQUAD	100
BDQRTIC	100	DIXMAANH	90	HELIX	3	SISSER	2
BIGGS3	3	DQRTIC	100	NCB20B	100	SPARSQUR	100
BIGGS5	5	EDENSCH	36	NONDIA	100	TOINTGSS	100
BIGGS6	6	EIGENALS	110	NONDQUAR	100	TQUARTIC	100
BOX2	2	EIGENBLS	110	OSBORNEA	5	TRIDIA	100
BOX3	3	EIGENCLS	30	OSBORNEB	11	WATSON	31
BRKMCC	2	ENGVAL1	100	PENALTY1	100	WOODS	100
BROWNAL	100	EXPFIT	2	PFIT1LS	3	ZANGWIL2	2
BROWNDEN	4	FLETCBV3	100	PFIT2LS	3		
CLIFF	2	FLETCHBV	100	PFIT3LS	3		

Table 3.2. Subset of unconstrained `CUTEst` problems and their problem dimensions [42].

The L-BFGS method has the form

$$x_{k+1} = x_k - \alpha_k H_k g(x_k), \tag{3.4.3}$$

where $g(x_k)$ is a finite-difference approximation to the gradient, H_k is the L-BFGS matrix with memory of 10 (see [75]), and α_k is a steplength selected by a relaxed Armijo-Wolfe line search designed to handle noise.

To describe the line search, let α_k^j denote the jth trial steplength at iteration k. Similar to Shi et al.[90], the Armijo condition is relaxed as follows:

(3.4.4)

$$f(x_k + \alpha_k^j p_k) \begin{cases} \leq f(x_k) + c_1 \alpha_k^j g(x_k)^T p_k & \text{if } j = 0,\ g(x_k)^T p_k < -\epsilon_g(x_k)\|p_k\| \\ \leq f(x_k) + c_1 \alpha_k^j g(x_k)^T p_k + 2\epsilon_f & \text{if } j \geq 1,\ g(x_k)^T p_k < -\epsilon_g(x_k)\|p_k\| \\ < f(x_k) & \text{if } g(x_k)^T p_k \geq -\epsilon_g(x_k)\|p_k\|, \end{cases}$$

where $c_1 = 10^{-4}$, $c_2 = 0.9$, and $\epsilon_g(x_k)$ is the estimated gradient error described below. Thus, we relax the line search only when the gradient is reliable; otherwise, we enforce simple decrease. We check the Wolfe condition by evaluating the directional derivative $\nabla\phi(x)^T p$ using finite differences along the direction p, as in [90].

3.4.2.1. Forward Differences. In the first set of experiments, the gradient approximation $g(x_k)$ is obtained by forward differences,

$$[g(x_k)]_i = \frac{f(x + h_i e_i) - f(x)}{h_i}, \quad i = 1, \ldots n,$$

where the differencing interval h_i is determined by one of the following three strategies.

1. `Fixed`. The interval h is fixed across all components i and for the entire iteration. This strategy tries to emulate the common practice of hand-tuning h_i at the start using problem specific information. We simulate this using the formula

$$h = 2\sqrt{\frac{\epsilon_f}{L_2}}, \text{ where } L_2 = \max\left\{10^{-1}, \sqrt{\frac{\sum_{i=1}^n [\nabla^2\phi(x_0)]_{ii}^2}{n}}\right\}, \tag{3.4.5}$$

which assumes that the diagonals of the Hessian are known. The gradient error is approximated assuming L_2 is correct, that is, $\epsilon_g(x) = 2\sqrt{nL_2\epsilon_f}$. We created this option for benchmarking purposes only.

2. `MW`. The Moré-Wild heuristic for estimating and interval h_i for every component i of the gradient. We set $L_2 = \max\{10^{-1}, \hat{L}_2\}$, where $\hat{L}_2$ is the estimate given by the Moré and Wild heuristic. If the heuristic fails, we set $L_2 = 10^{-1}$. The gradient error is estimated similar to `Fixed` but componentwise, i.e., $\epsilon_g(x) = 2\sqrt{\epsilon_f \sum_{i=1}^{n} L_{2,i}}$.

3. `Adaptive` Our adaptive procedure for estimating h_i along each component using Algorithm 3.1.

For the `MW` and `Adaptive` strategies, we re-estimate the second derivative or finite-difference interval whenever a partial derivative needs to be approximated. For example, when computing the full gradient, we estimate the finite-difference interval along each coordinate direction separately. We chose not to compare against Gill et al. [40] as we regard the Moré-Wild heuristic to be an improvement over their approach.

We present results for a few representative problems in Figure 3.3. The L-BFGS method described above is terminated if no further progress is made on the objective function over 5 consecutive iterations. Figure 3.3 plots the optimality gap $\phi(x_k) - \phi^*$ against the number of function evaluations. The optimal value ϕ^* is obtained by solving the original problem to completion without noise with L-BFGS. While we found Moré and Wild's heuristic to work well for $\epsilon_f < 10^{-1}$, their heuristic fails frequently for the case where $\epsilon_f = 10^{-1}$. (This can be seen in the complete results presented in Appendix B.2.) For this reason, we report results for $\epsilon_f = 10^{-1}$ and 10^{-5} to demonstrate the robustness of our algorithm compared

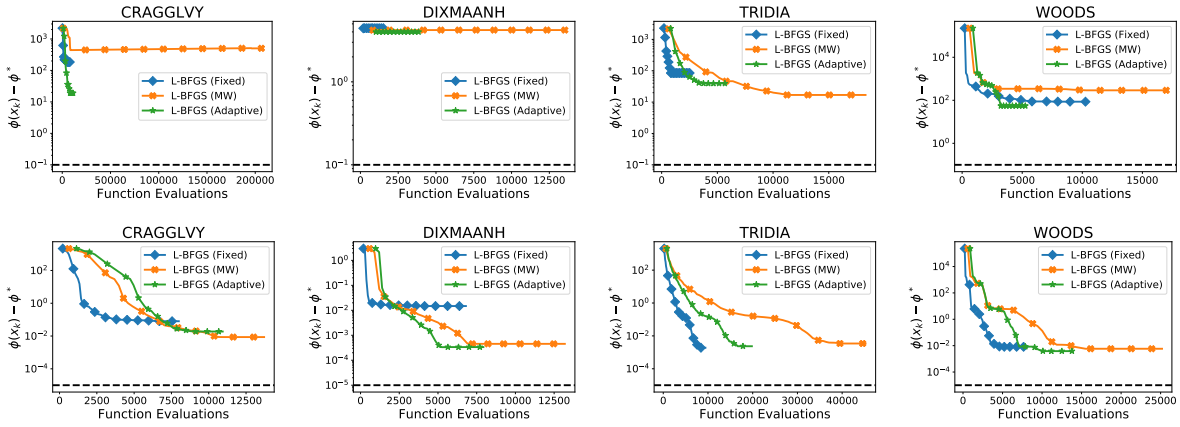

Figure 3.3. Comparison of forward-difference L-BFGS methods with difference intervals determined using a fixed interval, the Moré and Wild heuristic, and our adaptive algorithm. Comparisons are made on representative problems with noise level $\epsilon_f = 10^{-1}$ (top) and 10^{-5} (bottom). The solid line plots the observed function value and the dashed line plots the true function value. The dashed black line shows the noise level ϵ_f of the function.

to Moré and Wild for different noise levels. When Moré and Wild's heuristic succeeds, we observe that our algorithm (`Adaptive`) is able to more efficiently achieve comparable accuracy to the Moré and Wild heuristic, while attaining more accurate solutions than using a fixed interval for some problems. The lack of accuracy in the `Fixed` strategy can be explained by inability for a fixed interval to adapt to changes in the Hessian over the course of the iteration — an exception being the TRIDIA problem, which is very well scaled.

3.4.2.2. Central Differences. In the second set of experiments, we employ central differences,

$$[g(x;h)]_i = \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i}.$$

The differencing interval is determined via a `Fixed` strategy or the `Adaptive` procedure described in Algorithm 3.2. (The Moré and Wild's heuristic does not apply to this case.)

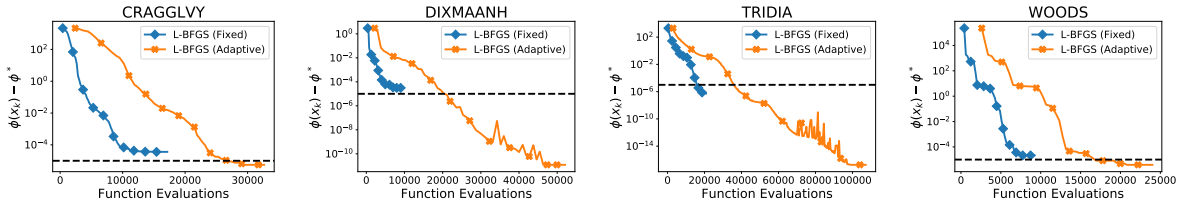

Figure 3.4. Comparison of central-difference L-BFGS methods with difference intervals determined using a fixed interval and our adaptive algorithm. Comparisons are made on representative problems with noise level $\epsilon_f = 10^{-5}$. The solid line plots the observed function value and the dashed line plots the true function value. The dashed black line shows the noise level ϵ_f of the function.

For the `Fixed` strategy, we choose

(3.4.6)
$$h = \sqrt[3]{\frac{3\epsilon_f}{L_3}}, \text{ where } L_3 = \max\left\{10^{-1}, \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{[\nabla^2\phi(x_0+\tilde{h}e_i)]_{ii} - [\nabla^2\phi(x_0)]_{ii}}{\tilde{h}}\right)^2}\right\}$$

and $\tilde{h} = \max\{1, |[x_0]_i|\}\sqrt{\epsilon_M}$. Note that noiseless forward differences are applied to the true Hessian to estimate the third derivative along each coordinate direction at the initial point. This synthetic `Fixed` strategy is presented for benchmarking purposes; it is not generally viable in practice.

Representative results are shown in Figure 3.4. Similar to the forward-difference case, our algorithm is able to obtain higher accuracy in the solution compare to the `Fixed` strategy, but at higher cost as expected. Complete experimental results for all problems and noise levels are presented in Appendix B.2.

3.5. Final Remarks

We have developed a principled and robust procedure for determining the difference interval for estimating gradients in optimization methods, assuming that the noise level is known. Our procedure applies to any finite-difference scheme, including central- and higher-order difference schemes. It performs a bisection search on a ratio that balances the truncation and measurement errors such that one typically attains a near-optimal difference interval. Whereas some methods for estimating the difference interval prioritize efficiency, such as Moré and Wild [71], and others compromise cost and accuracy, such as Gill, et al. [40], our approach is designed to be as robust as possible so that finite-difference gradient approximations can be reliably used in established nonlinear optimization techniques for solving noisy problems.

As demonstrated in our experiments, reusing previous difference intervals from prior iterations allows us to reduce the cost of the estimation procedure. Additional savings can achieved by re-estimating the difference interval periodically; for simple problems only a few times during the course of the optimization will suffice. The ability to exploit parallelism by distributing the computation of the gradient is an advantage that should not be underestimated when comparing the finite-difference approach with other techniques for derivative-free optimization.

CHAPTER 4

A Feasible Nonlinear Programming Approach for Constrained Derivative-Free Optimization

4.1. Introduction

Interpolation-based trust-region optimization (IBO) methods are among the most efficient and reliable techniques for unconstrained derivative-free optimization (DFO) [69]. Yet extending IBO methods to general constrained optimization is not straightforward and has received limited attention in the literature [59]. In this paper, we investigate a method first proposed by Conn et al. [26] designed for problems where the constraints and their derivatives are available analytically. This is a feasible method that constructs a quadratic model of the objective using an IBO approach and computes a step by minimizing this model subject to the original constraints of the problem and a trust region. The step computation is in general a nonlinear optimization sub-problem that is solved using a general purpose nonlinear solver. We refer to this approach as `FIBO`, for feasible interpolation-based optimization method. The numerical results presented in [26], and subsequently [23], are inconclusive and the `FIBO` approach is rarely, if ever, used in practice. In this paper, we present numerical results that suggest that, for an important class of practical applications, our proposed implementation of `FIBO` is competitive with the best methods proposed to date.

The problem under consideration is

$$\min_x \quad f(x) \tag{4.1.1}$$

$$\text{s.t.} \quad c_i(x) = 0, \quad i \in \mathcal{E} \tag{4.1.2}$$

$$c_i(x) \leq 0, \quad i \in \mathcal{I} \tag{4.1.3}$$

for some finite index sets $\mathcal{E}$ and $\mathcal{I}$, and where $f : \mathbb{R}^n \to \mathbb{R}$ and $c_i(x) : \mathbb{R}^n \to \mathbb{R},\ i \in \mathcal{I} \cup \mathcal{E}$, are smooth functions. We assume that the derivatives of f are not available but those of the constraints $c_i(x)$ are.

Applications of this problem setting are often found in practical applications in which the objective is provided by a simulation model and the constraints are given analytically, involving say trigonometric functions, or linear constraints. To cite a concrete example, in constrained reinforcement learning a single reward objective may not be sufficient and various constraints must be enforced to ensure safety requirements or physical limitations of the task [54].

4.1.1. Literature Review

Early methods for analytically constrained derivative-free optimization followed heuristic penalty and augmented Lagrangian approaches in which pattern search methods are employed to solve the sub-problems; see [59] for a survey of these methods. An important drawback of this approach is that derivative information of the constraints is not exploited. SID-PSM requires derivatives of the constraints but is difficult to scale to high dimensions [85]. In addition, as demonstrated by Moré and Wild [69] and Sahinidis [85], pattern

search methods are inefficient compared to IBO methods, both on smooth and nonsmooth DFO problems.

Powell developed efficient IBO methods and software to deal with increasingly more complex sets of constraints. `BOBYQA` [79] is designed for problems with simple bounds; it builds upon the minimum-Frobenius norm update method first introduced in `NEWUOA`. Powell's last code, `LINCOA`, is able to deal with linear constraints [80]. These methods are quite sophisticated as they seek to minimize linear algebra costs. They are accessible through the Python and Matlab interface developed by Zaikun [84]. Powell did not develop an IBO methods to handle general (nonlinear) constraints other than `COBYLA`, which solves general inequality constrained DFO problems by constructing linear models of the objective and constraints. An extension of this approach, named `COBYQA`, was proposed by Ragonneau [83] to handle both equality and inequality constraints via quadratic models.

A variety of IBO methods have been proposed to enforce feasibility of the iterates. When the constraints are analytically available, this set of methods often impose the constraints in the trust region subproblem at every iteration. This line of work was first extended to convex constraints by [24], in which the authors provide convergence analysis assuming that the local model is always fully-linear. [51] extended the global convergence analysis by proposing a generalized fully-linear model in the general convex constrained case, relaxing this fully-linear model assumption. However, [51] only provides numerical results on least-square functions and neither work demonstrate the effectiveness of their approach on general objective functions. Limited work has been done when general constraints are present, as the resulting subproblem from the constraint enforcement is in general computationally expensive to solve. To our knowledge, this approach was first

explored by Conn et. al [26]. Yet the numerical experiments do not provide convincing evidence regarding the effectiveness of the method due to the ill-considered experiment settings. In particular, the other DFO methods being considered do not exploit the availability of constraint derivatives. [23] revisited this approach but under the same reason, does not provide additional insights into this framework. No convergence proof has yet been established for this feasible approach with general constraints. A similar idea is again explored in `CONDOR`, which is designed to solve general inequality constraints by treating bound and linear constraints via active-set methods and the remaining with SQP [11]. For a thorough treatment of IBO methods, see [29].

A recent line of work in unconstrained DFO focuses on methods that employ finite difference approximations to the gradient [7, 73, 93]. This approach can be effective even for noisy functions, if the finite difference interval is chose appropriately. Given that state-of-the-art nonlinear programming solvers normally include a finite difference option for both functions and constraints, they should serve as a benchmark for testing constrained DFO methods.

4.2. The FIBO Algorithm

Let us begin by sketching a basic `IBO` method for unconstrained derivative-free optimization, and refer the reader to [29, 78] for a discussion of many fine points of implementation. The algorithm starts by evaluating the objective function at a set Y_0 of points on a stencil around the initial point x_0. The algorithm then sets $k \leftarrow 0$ and constructs a quadratic model

$$m_k(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T H_k s. \tag{4.2.1}$$

This model can be defined in two different ways. Assuming that Y_k is *poised* and that $|Y_0| = (n+1)(n+2)/2$, this model is uniquely specified by requiring that it interpolate the function values at the current interpolation set. An alternative IBO approach that greatly reduces the linear algebra costs of the iteration employs only $|Y_k| = 2n+1$ interpolation points and defines a model through a minimum Frobenius norm update of the matrix H_k subject to $|Y_k|$ interpolation conditions.

Regardless of how the model is constructed, the trial step s_k is given by the solution of the trust region subproblem

$$\min_s \quad m_k(x_k + s) \tag{4.2.2a}$$

$$\text{s.t.} \quad \|s\|_2 \leq \Delta_k. \tag{4.2.2b}$$

As in any trust region method, acceptance of the step s_k is determined based on a ratio

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \tag{4.2.3}$$

of actual vs predicted reduction in the objective. If sufficient decrease is obtained ($\rho_k \geq \eta$ for some $\eta \in (0, 1)$), the new iterate is defined as $x_{k+1} = x_k + s_k$; otherwise the step is rejected and one sets $x_{k+1} = x_k$.

The trust region Δ_k is increased for successful steps. Otherwise, the algorithm considers two cases. If the interpolation set is not poised, (i.e., if the interpolation points (nearly) lie in a linear subspace of $\mathbb{R}^n$), Δ_k remains unchanged to avoid premature shrinkage of the trust region and Y_k is improved via a geometry phase. If the step is rejected and Y_k is poised, then Δ_k is decreased.

The set Y_k is normally updated at every iteration by removing one point from Y_k (for example, the furthest from x_{k+1}) and by adding x_{k+1} to this set. This trust region interpolation-based approach has proved to be a very efficient for solving unconstrained DFO problems.

We can now describe `FIBO`, the extension of this method to the constrained optimization problem (4.2.2). We assume—and this is important—that the constraints are both analytically available and inexpensive to evaluate compared to the objective function. Then, it is realistic to compute the step by

$$\min_s \; m_k(x_k + s) = f(x_k) + g_k^T s + \tfrac{1}{2} s^T H_k s \tag{4.2.4a}$$

$$\text{s.t.} \; c_i(x_k + s) = 0, \quad i \in \mathcal{E} \tag{4.2.4b}$$

$$c_i(x_k + s) \leq 0, \quad i \in \mathcal{I} \tag{4.2.4c}$$

$$\|s\|_2 \leq \Delta_k. \tag{4.2.4d}$$

The model m_k in (4.2.4a) is constructed by interpolating function values at a set of interpolation points, and as in the unconstrained case, models the objective function f. Note that (4.2.4) includes the original constraints of the problem and is, in general, a nonlinear programming problem that must be solved using any general-purpose nonlinear programming solver (`KNITRO` in our experiments).

Let us assume for now that (4.2.4) can be solved to optimality. If the initial iterate is feasible, then all subsequent iterates will be feasible and it is therefore sufficient for m_k to model only the objective and disregard the contribution of the constraints. All other details of the algorithm, such as the update of the interpolation set and trust region, are

defined as in the unconstrained IBO method. In fact, to develop the `FIBO` method one can start with any IBO method for unconstrained optimization and simply replace the step computation (4.2.2) by (4.2.4).

This framework therefore provides substantial flexibility in that any `IBO` method can be used to construct the model and handle the trust region logic, and any nonlinear programming code can be invoked to solve the trust region subproblem (4.2.4). We state the proposed method in Algorithm 4.1.

Algorithm 4.1. FIBO

1: Parameters: $0 \leq \eta < 1$, and $\Delta_0 > 0$.
2: Choose x_0, a **feasible** starting point, and construct an initial poised interpolation set Y_0.
3: Let $k = 0$.
4: **while** no convergence test is satisfied **do**
5: Build a local (quadratic) model (4.2.1) using interpolation set Y_k.
6: Compute a step s_k by solving the trust region subproblem (4.2.4).
7: Compute the ratio defined by (4.2.3).
8: **if** $\rho_k \geq \eta$ **then**
9: Set $x_{k+1} = x_k + s_k$.
10: Choose $\Delta_{k+1} \geq \Delta_k$.
11: Update Y_k to include x_{k+1}.
12: **else if** Y_k needs to be improved **then**
13: Set $x_{k+1} = x_k$.
14: Set $\Delta_{k+1} = \Delta_k$.
15: Improve Y_k using a geometry-improving procedure.
16: **else**
17: Set $x_{k+1} = x_k$.
18: Choose $\Delta_{k+1} < \Delta_k$.
19: Update Y_k to include $x_k + s_k$.
20: **end if**
21: $Y_{k+1} = Y_k$
22: $k = k + 1$
23: **end while**

We now comment on the following salient properties of this approach.

Application to Noisy DFO. The `FIBO` method can be applied to noisy DFO problems without additional modifications. As noted in [93], `IBO` methods are normally efficient and robust in the presence of noise for reasons that are not well understood, and one can expect these benefits in the constrained case as well. An added advantage is that information regarding the noise level in the problem is not required, compared to DFO methods that employ finite differences.

Feasibility. Assuming that the nonlinear programming solver is successful at every iteration, all iterates of the `FIBO` algorithm are feasible. This is a desirable property for DFO problems in which the objective function is not defined outside the feasible set. (The nonlinear programming method used to solve the subproblem (4.2.4) typically generates infeasible iterates, but it is only the quadratic objective m_k of this subproblem that is evaluated at those points and not the original objective function.) Although the success of the nonlinear solver is not guaranteed since the subproblem (4.2.4) is not convex, we did not observe any failures in our experiments—and we should note that other approaches for constrained DFO [59] do not provide convergence guarantees in all situations, either.

Feasible Interpolation. As discussed in [59], sufficiently good model accuracy can be difficult or impossible to obtain when only feasible points are interpolated. A similar situation occurs when the objective cannot be evaluated at infeasible points. The initial interpolation set has to be constructed differently to include only feasible points. We do not explore this possibility in this work.

Per-Iteration Cost. The three main expenses of the `FIBO` iteration are:

(1) Evaluation of the objective f. This cost is problem dependent but there are many applications where the function evaluation is much more costly than all other computations in the algorithm.

(2) Construction of the quadratic model. Computing the model m_k from scratch at every iteration requires $O(n^6)$ flops, which is quite expensive when the number of variables is large Nevertheless, Powell's approach implemented in `NEWUOA` requires only $O(n^2)$ flops since it employs only $O(n)$ interpolation points.

(3) Solution of the Trust Region Problem. When the number of variables n is very small (say less than 100), the cost of solving (4.2.4) is of no concern even if the nonlinear programming algorithm performs many iterations to return an accurate solution, *provided* the constraints are inexpensive to evaluation. But as the number of variables becomes large the total linear algebra cost on the nonlinear programming is of concern. Note, in particular that since the Hessian of m_k is dense, each evaluation of m_k requires $O(n^2)$ flops. Similarly, if the cost of evaluating the constraints and their derivatives is high, then the `FIBO` approach is not viable given the need for multiple evaluations for the solution of a single trust region step. In the next section, we quantify more precisely the class of problems for which `FIBO` is efficient.

In summary, the `FIBO` method is appropriate for problems with

- expensive function evaluation (at least as expensive as $O(n^2)$ operations),
- inexpensive constraint evaluation,

such that the cost associated with solving the subproblem (4.2.4) and solving the interpolation system is dominated by computational overhead of the objective evaluation.

4.3. Numerical Experiments

We adapt the Python code for `DFOTR` to incorporate general constraints. In particular, we replace the original module for solving trust region subproblem by calling `KNITRO` to solve the general constrained subproblem. Note that the derivative information of this subproblem (4.2.4) is available. Since `DFOTR` picks the interpolation point with the minimum objective value as the best iterate, we disable this option as such point may be infeasible.

We will compare `FIBO` with finite-difference based `KNITRO` (denoted by `FD`), which has been demonstrated to be effective for noiseless problems [93]. We do not present numerical results for solving noisy constrained problems as we believe the problems can be solved similarly as in the nonnoisy constrained case. The details of the solvers are described below.

- `FIBO`. We employ the code `DFOTR` to perform trust region updates, with final trust region radius 10^{-8}. The maximum number of function evaluations allowed is set to be $500 \times \max(m, n)$, where n is the number of variables and m the number of constraints. We set the `stop_predict` convergence test to 0 avoid early termination caused by the ratio test. All remaining parameters are set to default. We use `KNITRO` with *exact gradients* to solve the trust region subproblem and the parameters of `KNITRO` are set to default except `alg = 4` (SQP) and `hessopt = 6` (L-BFGS).
- `FD`. We ran `KNITRO` with `alg = 4` (SQP) and `hessopt = 6` (L-BFGS). The memory size is set to be 10 (default value for `lmsize`). We set `xtol = 1e-8,`

`xtol_iters = 1`, and `findiff_terminate=0` to obtain similar convergence criterion as `DFOTR`. Maximum number of function evaluations allowed is $500 \times \max(m, n)$. We perform *forward differencing* to the objective and provide exact derivative information of constraints to the algorithm.

We consider 38 general constrained problems and assume that the initial points x_0 are feasible. Since the initial point provided by `CUTEst` is usually infeasible, we ran `KNITRO` to obtain a feasible point by replacing the objective by 0. The cost to obtain a feasible solution is omitted for now, as we assume the constraints are not expensive to evaluate. Note that the observations remain the same when we include the cost of obtaining a feasible starting point for `FIBO` and run `FD` from an infeasible initial point x_0. See C.2 for detailed numerical results.

We study the performance of `FIBO` by comparing its final objective accuracy and the associated cost. To study the accuracy of the solvers, we measure accuracy with log-ratios:

$$\log_2\left(\frac{\max(f_{\text{FIBO}} - f^*, 10^{-8})}{\max(f_{\text{FD}} - f^*, 10^{-8})}\right) \tag{4.3.1}$$

where f_{FIBO} and f_{FD} are the final accuracy obtained by the corresponding approach, and the optimal value f^* is obtained by running `KNITRO` with exact gradient. We compare the final accuracy of the two methods up to 8 digits and report the ratios using log-ratio plot in Figure 4.1 [67]. The ratios (4.3.1) are plotted in increasing order such that the area of the shaded region provides a rough idea of the relative performance of the methods. `FD` achieves higher accuracy than `FIBO`, but not by a wide margin. For full numerical results, see Appendix C.2.

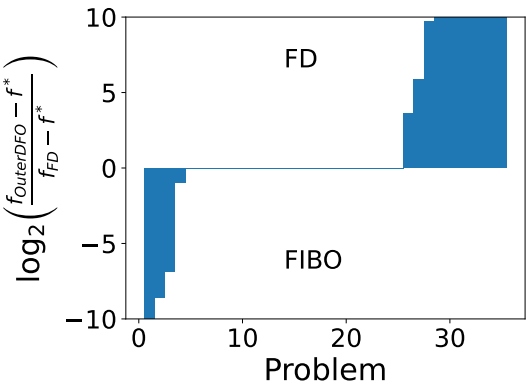

Figure 4.1. Log-ratio Plot for Comparing the Final Accuracy (4.3.1) of `FIBO` and `FD`.

To understand the efficiency of the compared algorithms, we terminate the algorithms when

$$f(x_k) \leq f^* + \tau \cdot \max(1, |f^*|) \tag{4.3.2}$$

where f^* is the optimal solution obtained by running `KNITRO` with exact gradients on the test problems, and τ is the desired accuracy level. In particular, we look at the number of objective evaluations and constraint evaluations to understand the associated cost to reach a given accuracy level. We denote the number of function evaluations required to satisfy (4.3.2) by $\text{evals}_{\text{FIBO}}$ and evals_{FD} respectively for the corresponding method. Similarly, we use $\text{cevals}_{\text{FIBO}}$ and $\text{cevals}_{\text{FD}}$ to denote the number of constraint evaluations required to satisfy (4.3.2). If the condition (4.3.2) cannot be satisfied by an algorithm, we simply set the corresponding quantity as a very large number. We specifically present results for $\tau = 10^{-1}, 10^{-3}, 10^{-5}$ and 10^{-7} to study the efficiency for reaching different accuracy levels. Again, we summarize the results using the log-ratio profile proposed by Morales [67] and

plot the quantities

$$\log_2\left(\frac{\text{evals}_{\text{FIBO}}}{\text{evals}_{\text{FD}}}\right), \quad \log_2\left(\frac{\text{cevals}_{\text{FIBO}}}{\text{cevals}_{\text{FD}}}\right). \tag{4.3.3}$$

The plots for comparing the number of objective evaluations and constraints evaluations are reported in Figure 4.2 and 4.3.

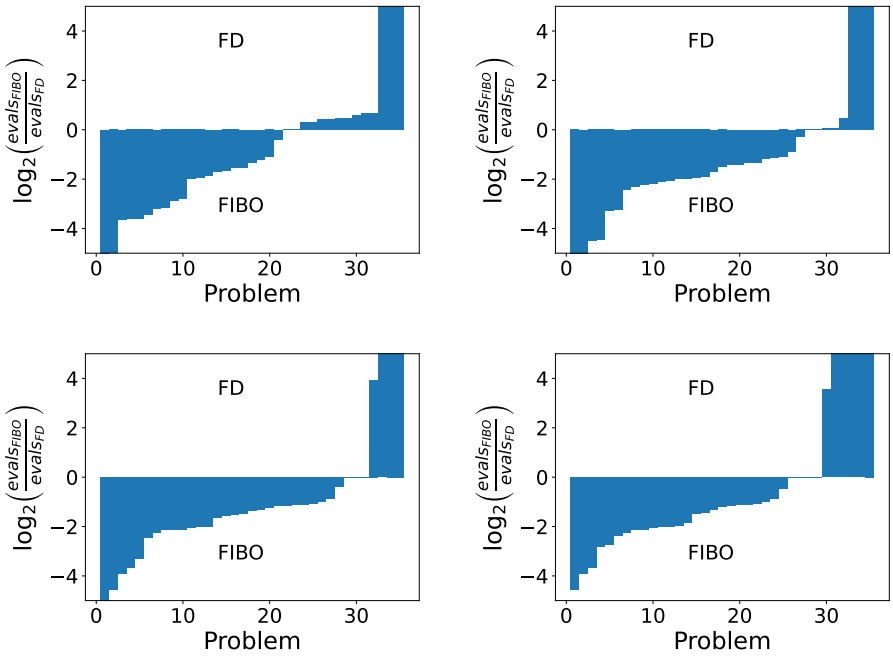

Figure 4.2. Log-ratio plot comparing `FIBO` and `FD` in terms of the number of function evaluations to satisfy (4.3.2) for $\tau = 10^{-1}$ (upper left), 10^{-3} (upper right), 10^{-5} (bottom left), 10^{-7} (bottom right).

As indicated by Figure 4.2, `FIBO` outperforms finite-difference based `KNITRO` in terms of objective evaluations across various accuracy levels. Yet `FD` outperforms `FIBO` in terms of constraint evaluations across all accuracy levels. Since each constraint evaluation is associated with an evaluation of the quadratic model (4.2.1), `FIBO` is very effective when each evaluation of the objective function is at least as expensive as $O(n^2)$ and the cost of

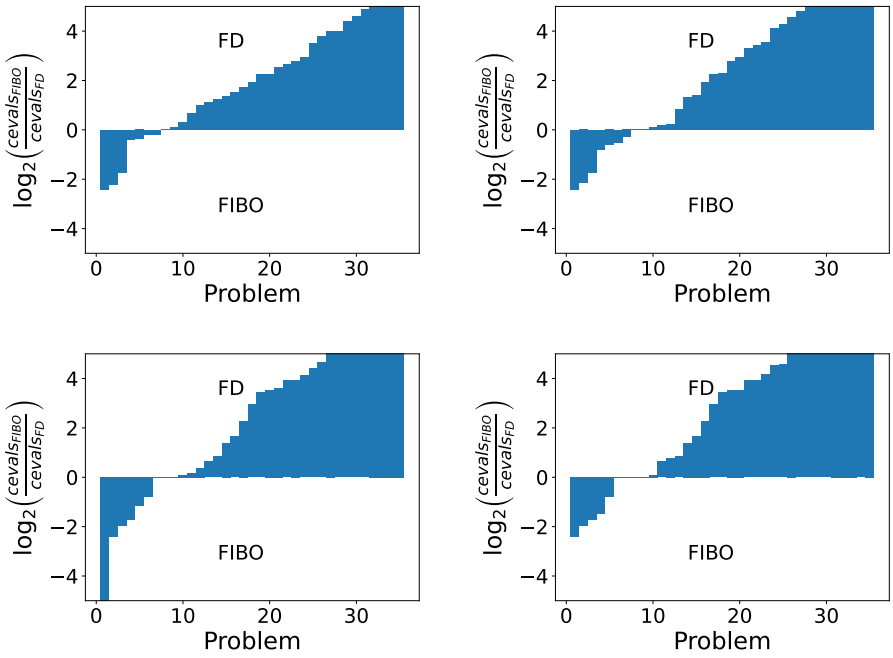

Figure 4.3. Log-ratio plot comparing `FIBO` and `FD` in terms of the number of constraint evaluations to satisfy (4.3.2) for $\tau = 10^{-1}$ (upper left), 10^{-3} (upper right), 10^{-5} (bottom left), 10^{-7} (bottom right).

each constraint evaluation is negligible. Otherwise, when the cost of solving the subproblem (4.2.4) is formidable, `FD` is a viable approach.

4.4. Final Remarks

We have demonstrated the effectiveness of the `FIBO` approach based on empirical investigations. When only analytical constraints are present, our study has revealed that, with minor modifications, existing unconstrained IBO methods can be extended to solve general constrained DFO problems. Our numerical results indicate that `FIBO` is a competitive method for problems where the cost of the objective function dominates that of the constraints. Our framework applies to any IBO methods and gradient-based nonlinear

programming solvers. Yet more sophisticated algorithmic designs are required when the constraints cannot be violated, or black-box constraints are present. It is non-trivial to compute interpolation points that are feasible, when model-improving points are necessary to improve model quality. In addition, when constraints are expensive to evaluate, further algorithmic development for IBO is required to improve efficiency of the proposed method. We have observed fair performance of `FIBO` even if the termination test for (4.2.4) is relaxed, but the relaxation strategy requires more careful investigations.

CHAPTER 5

Prompting Large Language Models with Derivative-Free Optimization

5.1. Introduction

Pre-trained large language models (LLMs) like GPT-4 have been widely successful in various language tasks. However, LLMs are general-purpose models and require additional adaptations, such as fine-tuning or prompting, to address specific language tasks like sentiment analysis. Fine-tuning requires updates of the model parameters on each new language task [52]. This process can be resource-intensive and challenging given the model complexity and the limited sample size of the subsequent task. Additionally, maintaining separate copies of the language model for different tasks poses additional challenge in terms of storage requirements, especially given the massive size of the LLMs. On the other hand, prompting offers a storage-efficient solution by appending a prompt, e.g., task description texts, to the input text while keeping the parameters of the pre-trained model frozen. Prompting has demonstrated its effectiveness on LLMs with the appealing feature of using the same pre-trained model on a variety of tasks [15]. However, effective design of the prompts still remains a challenge: they are often designed in a manual trial-and-error manner and can be sensitive to perturbations.

Several efforts have been proposed to tackle this challenge, including gradient-based approaches that optimize continuous prompts, which are continuous vectors instead of

discrete texts [60, 61]. However, accessing the gradient of the loss function through backpropagation in LLMs like GPT-4 is often not available, as they typically only allow input and output manipulations. Furthermore, the massive size of LLMs makes computing derivatives of the objective function through backpropagation impractical, primarily due to memory constraints. Moreover, LLMs are usually implemented using finite precision arithmetic, with common choices in half and single precision. This choice of precision can introduce higher roundoff errors during model evaluation compared to computations performed in double precision.

In light of this limitation, we approach prompt design as a noisy derivative-free optimization (DFO) problem. Several classes of DFO methods have gained popularity within different research communities, each offering unique advantages. Randomized methods such as evolution strategies have been widely embraced within the machine learning communities. This class of methods employs iterative sampling based on a stochastic distribution, which may be updated to leverage iteration information. Their effectiveness has been demonstrated on applications arising from Reinforcement Learning [98] to Natural Language Processing [100]. Alternatively, interpolation based optimization (IBO) methods and finite-difference (FD) based methods have been recognized as the state-of-the-art DFO algorithms within the optimization community for their robustness and efficiency [93]. These methods rely on utilizing function values to construct approximations to the objective function and/or gradients.

Considering the diverse range of DFO methods available, each with its unique advantages, we examine these approaches in solving the noisy DFO problem arising from prompt design.

5.1.1. Literature Review

Transformer-based Language Models. Transformer-based language models have have been in the spotlight in recent years due to their impressive performance on a wide range of language tasks. The original transformer architecture was proposed in [101], which introduced the self-attention mechanism within an encoder-decoder structure, allowing the model to extract contextual information from the input sequence. The text-to-text transformer (T5) model uses the encoder-decoder structure [82]. Examples of model usage include question-answering. BERT and its variants use an encoder structure in the seq-2-seq architecture to extract general text representations [33]. They can be extended to applications such as sentiment classification. The GPT-2 [81] and GPT-3 [15] are decoder-based transformer models, so that they are trained in an autoregressive manner. The model can be used in, for instance, automatic sentence completion and text generation.

In practice, LLMs often have billions of parameters and thus require significant computational resources and memory, making gradient-based training infeasible for many users and applications. Furthermore, the majority of performant LLMs, such as GPT-4, and BARD, are not open-sourced. They are exclusively accessed through blackbox APIs that typically consume and generate natural language sequences. Due to the limited access to the models, leveraging them to address specific language tasks can be challenging. This calls for effective strategies such as prompt design.

Prompting. Prompts are pieces of information to be prepended to the input X so that the language model can generate a desirable output Y using the extra information. Prompting is desirable for large-size language models, for its advantage of not requiring updates or storage of all model parameters as in the fine-tuning paradigm, which can be

prohibitively expensive. Various prompting approaches have been proposed and yet it still remains an open question to achieve competitive performance against fine-tuning. See [62] for a thorough treatment.

Prompts can be classified into hard prompts and soft prompts. Hard prompts are often composed of task descriptions texts that rely on manual efforts. For instance, "I felt bored watching the movie." may be more likely to be classified as a negative sentence by a language model if we append the prompt "It was __' to the sentence and ask the model to generate the word at the corresponding position. Yet hard prompts can be sub-optimal and sensitive to prompt choices [60]. In contrast, soft prompts are continuous vectors to be injected to the input. They do not necessarily use the same embedding as in the pre-trained language models. The vectors can be tuned or they can be treated as learnable parameters and be optimized via gradient-based methods through backpropagation [60, 61], if the gradient information of the pretrained LLM is available.

However, as mentioned in the previous sections, LLMs may only be accessed as blackboxes so that their gradient information or parameters are not accessible. As a result, [100] proposed Black-Box Tuning, which optimizes continuous prompts by employing the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) method. Other blackbox approaches for prompt tuning include [32, 34], which employ discrete prompt optimization.

5.2. Problem Formulation

We consider a general classification task, which predicts a label $y \in \mathcal{Y}$ for a given input sequence $x = (x_1, ..., x_L)$ of length L using a general-purpose pre-trained LLM. To simplify the setting, we assume access to the embedding matrix E of the LLM and we

denote $X \in \mathbb{R}^{L\times D}$ as the embedded representation of the original input sequence x, where D is the embedding dimension. The embedded input X can be viewed as a numerical representation of the original text sequence via a linear projection using the embedding matrix.

Our goal is to search for a continuous prompt $p \in \mathbb{R}^d$ to be combined with every embedded input X_i in the training samples by solving the following optimization problem:

$$\min_{p\in\mathbb{R}^d} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(p; X_i), y_i), \tag{5.2.1}$$

where $\mathcal{L}$ is the loss function (e.g., cross entropy loss function), f is the black-box pre-trained language model and N is the number of training samples. The size of the prompt p is typically chosen as $d = L_p \times D$, where L_p is a user-provided hyperparameter. Note that the prompt $p \in \mathbb{R}^d$ and the i-th embedded input $X_i \in \mathbb{R}^{L_i\times D}$ may be passed to the rest of the LLM in the form of either $\begin{bmatrix} p \\ X_i \end{bmatrix} \in \mathbb{R}^{(L_i+L_p)\times D}$, or $\hat{p} + X_i \in \mathbb{R}^{L_i\times D}$, where $\hat{p} \in \mathbb{R}^{L_i\times D}$ is usually generated from p using zero padding. We consider the latter form of prompt-input combination in our experiments, as it in general results in shorter input sequences lengths and thus less training time. In addition, $p = 0$ can be used as a natural starting point. Note that the same prompt is appended to every training sample.

The key challenge in solving (5.2.1) lies in the dimensionality of the problem. Since the embedding dimension (D) is often very high, the size of prompt vector p is on a scale of thousands. Yet it has been empirically shown that LLMs have low intrinsic dimensionality. In other words, by tuning only hundreds of parameters and performing random projection onto the full parameter space, one make achieve reasonably good

performance on downstream NLP tasks [1]. Therefore, (5.2.1) can be reformulated as

$$\min_{z \in \mathbb{R}^n} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(Az; X_i), y_i) \tag{5.2.2}$$

where $A \in \mathbb{R}^{d \times n}$ is a random projection matrix such that $n \ll d$.

Recall that the gradient of the function f is not available or very expensive to compute. Thus (5.2.1) and (5.2.2) are unconstrained derivative-free optimization problems.

Furthermore, the problem formulation (5.2.2) is noisy in practice as LLMs are typically implemented in single precision, half precision or mixed precision due to GPU memory constraints and speed concerns. Therefore, DFO algorithms such as finite differences must be applied carefully when such deterministic noise is present in the function evaluations.

We remark that our approach assumes that the continuous projected prompt Az in (5.2.2) can be combined with the embedding representation X of the input sequence x directly. However, there are cases when a given LLM only accepts text inputs and does not provide access to its embeddings. In such scenarios, the prompt must be prepended to the input text sequence before passing through the embedding. This renders the prompt p discrete and (5.2.1) intractable to solve. We argue that it can be beneficial for LLM services to include a feature that enables seamless integration of continuous prompts into the embeddings. This would simplify the handling of hard prompts, ultimately benefiting a wide range of language tasks and workflows.

5.3. Numerical Experiments for Optimizing Prompts

We evaluate the effectiveness of our proposed approach based on transformer-based language models RoBERTa and T5, in addressing several common language classification

tasks, including sentiment classification, natural language inference and topic classification. To tackle these tasks, we examine three state-of-the-art classes of DFO algorithms: IBO methods, finite differences and randomized algorithms for solving the problem (5.2.2). We describe below our experiments details.

5.3.1. Experiment Setup

Datasets. We evaluate on 6 common language classification datasets: SST-2 [95], Yelp P. [106], SNLI [12], RTE [102], DBpedia [3], AG's News [106]. The datasets include tasks such as sentiment analysis, topic classification and natural language inferences.

Few-Shot Learning. It has been demonstrated that LLMs have great power for few-shot learning [15]. To simulate the scarce data setting, we consider a 16-shot learning setting in our experiments. In particular, we randomly choose 16 samples for each class to construct the training and validation set. As in [100], we use the original test set for consistency with existing benchmarks.

Backbone Model. In our experiments, we consider open-source transformer-based models RoBERTa-large [63] and T5-large [82] for solving the language tasks so that the prompts can be directly applied to the embedded representations. The chosen models are representative of bi-directional encoder and encoder-decoder models respectively.

Hyperparameters. We employ cross entropy loss as our loss function $\mathcal{L}$ and use accuracy to measure performance. We consider the decision variable z of dimension 500 ($n = 500$) along with prompt length $L_p = 50$. We employ a projection matrix A that is generated from a Gaussian distribution with mean and covariate matrix computed using the mean and standard deviation of the language model embeddings. Note that

the projection matrix A remains fixed during the experiments. We set a budget of 20,000 function evaluations in our numerical experiments.

DFO Solvers. We describe below the implementation details of the DFO algorithms.

- `CMA-ES`: A stochastic DFO algorithm developed by [45]. We call an Python implementation of `CMA-ES` via the `pycma` module with version 3.3.0. The parameters are set to default and no additional information is provided regarding noise. Specifically, the method samples $4 + 3\sqrt{n}$ points at every iteration.
- `FD L-BFGS`: A finite-difference based implementation of the limited-memory BFGS algorithm with a bisection Armijo-Wolfe linesearch. The memory size is set to 10. We used forward differencing to compute a finite difference approximation of the gradient and the directional derivative along the search direction during linesearch. We employed differencing interval h of size 0.1 based on machine epsilon for single precision and an estimated bound on the second order derivative of the loss function.
- `NEWUOA`: A state-of-the-art unconstrained IBO method that is available through the `PDFO` package [84]. We set `rhoend =` 10^{-8} as a termination criterion for this algorithm. All other parameters are set to default.

We use $z = 0$ as the starting point for all algorithms.

5.3.2. Results

We comment on the training loss of the prompt optimization problem (5.2.2) across different datasets. Our observations reveal interesting insights into the performance of different solvers. Overall we observe that `CMA-ES` quickly reduces the training loss within

n function evaluations. On the other hand, `FD LBFGS` requires $n+1$ function evaluations to approximate a gradient and `NEWUOA` requires $2n+1$ function evaluations for quadratic model construction during the first iteration. As the number of model evaluation increases, we notice that `NEWUOA` becomes more efficient in minimizing the training loss compared to the other solvers. When a sufficiently large number of model evaluations is allowed, `CMA-ES` may obtain the lowest training loss among the solvers we have compared. We also observe that `FD L-BFGS` tends to be the least efficient solvers among the solvers. To illustrate the relative performance across the solvers, we plot representative training loss and validation loss in Figure 5.1 and Figure 5.2.

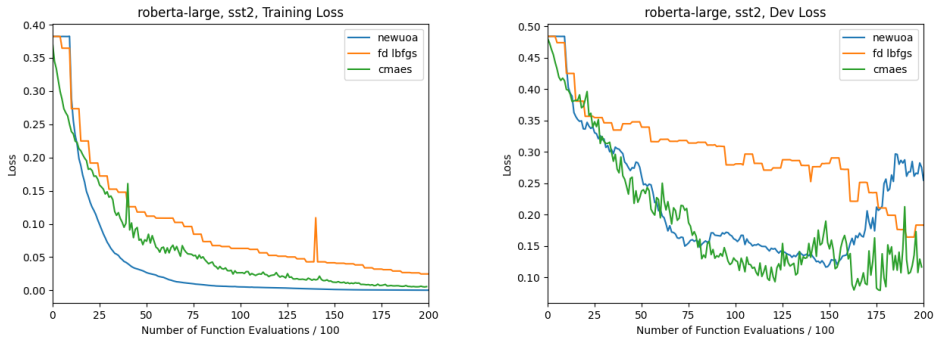

Figure 5.1. Training and Validation Loss for Prompt Optimization of RoBERTa-large for Solving SST-2.

Dataset	SST-2	Yelp P.	AG's News	DBPedia	SNLI	RTE
CMA-ES	**89.79**	85.82	**83.89**	85.65	45.04	42.24
NEWUOA	88.30	91.62	83.11	**95.78**	44.51	42.24
FD L-BFGS	86.93	**91.85**	81.68	82.70	**47.94**	**53.07**
Manual Prompt	83.60	89.64	75.75	31.92	38.82	51.43

Table 5.1. Test accuracy: Prompt Optimization for RoBERTa-large.

Moreover, we present the final test accuracy based on the model that corresponds to the lowest validation loss. The results are summarized in Table 5.1 and Table 5.2

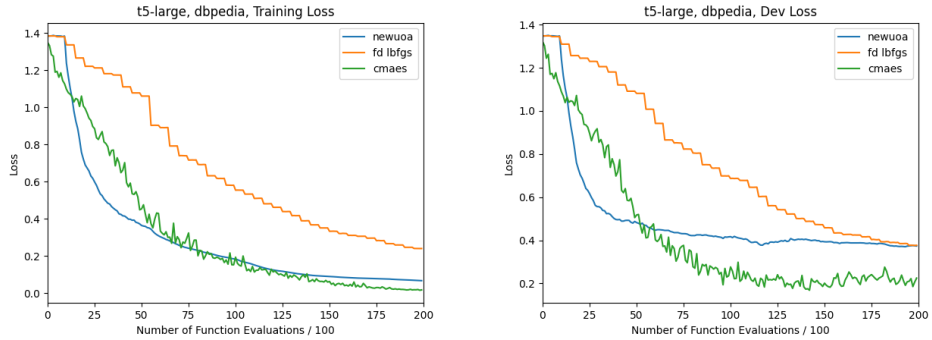

Figure 5.2. Training and Validation Loss for Prompt Optimization of T5-large for Solving DBPedia.

Dataset	SST-2	Yelp P.	AG's News	DBPedia	SNLI	RTE
CMA-ES	86.70	**95.91**	**83.51**	**93.25**	**46.62**	46.93
NEWUOA	86.01	93.97	77.74	89.46	42.59	**56.32**
FD L-BFGS	**91.28**	94.92	77.64	90.26	40.17	53.79
Manual Prompt	69.04	82.24	40.07	59.40	40.18	54.51

Table 5.2. Test accuracy: Prompt Optimization for T5-large.

for RoBERTa-large and T5-large, respectively. We also include results for a manually tuned prompt, which in general yields lower accuracy compared to the final test accuracy obtained by the DFO solvers. Our numerical experiments indicate that `NEWUOA` exhibit comparatively lower test accuracy across the evaluated datasets despite its efficiency in reducing the training loss. For RoBERTa-large, neither `FD L-BFGS` nor `CMA-ES` emerges as a clear winner. In contrast, when T5-large is used as the backbone model, CMA-ES achieves the highest test accuracy across majority of the datasets.

5.4. Final Remarks

We have addressed an important problem of adapting large language models to various language classification tasks by leveraging DFO algorithms. We have demonstrated the

efficacy of several DFO algorithms, including `CMA-ES`, `NEWUOA` and `FD L-BFGS` in solving this DFO problem. Our numerical study has highlighted the efficiency of `NEWUOA` in effectively minimizing the training loss. Additionally, we have observed the potential of `CMA-ES` in achieving the lowest training loss when a sufficiently large number of model evaluations are employed. Nevertheless, further investigation is needed to assess the generalization of DFO methods on the testing set, as well as a more sophisticated formulation of the DFO problem. We also note that the unavailability of embedding information for some existing LLMs poses challenges for this prompt design approach. Additional investigations can be conducted to study the feasibility of utilizing external embeddings for this prompt design problem and to provide insights into the applicability of DFO methods in this setting.

CHAPTER 6

Analyzing the Performance of DFO Methods on a Wider Class of Problems

6.1. Introduction

Motivated by the observations in Chapter 5, we delve deeper into the comparative analysis of DFO methods by exploring more generalized problem settings. We are interested in whether the relative performance of the compared DFO solvers, concerning the objective presented in problem (5.2.2), extends to other functions in general. We aim to gain further insights into the relative merits and performance characteristics of these DFO methods across a broader range of problem scenarios, and identify potential avenues for improving their performance in practice. In particular, we explore strategies for improving the accuracy of IBO methods in the presence of noise.

Formally, we now consider general unconstrained DFO problems of the following form as in (1.0.1) (assuming $\Omega = \mathbb{R}^n$)

$$\min_{x \in \mathbb{R}^n} \phi(x) \tag{6.1.1}$$

where $\phi : \mathbb{R}^n \to \mathbb{R}$ is a smooth function. Recall from Chapter 1 that the derivatives of ϕ are not available and only noisy evaluations of the function can be observed:

$$f(x) = \phi(x) + \epsilon(x), \tag{6.1.2}$$

where $\epsilon(x)$ models the evaluation error at x. We assume that the noise $\epsilon(x)$ is stochastic and independent of x. See Chapter 1 for further discussions on the noise model.

As has been well-studied in [93], `NEWUOA` and `FD-LBFGS` are competitive for noiseless problems, with `NEWUOA` showcasing a slight advantage over `FD L-BFGS` for noisy problems. Consequently, we narrow down our empirical investigations to `NEWUOA` and `CMA-ES` for solving unconstrained DFO problems. Therefore, the solvers tested in this chapter are as follows.

`CMA-ES`. We call an Python implementation of `CMA-ES` via the `pycma` module with version 3.3.0. The parameters are set to default and no additional information is provided with respect to noise. To elaborate, the method samples $4 + 3\sqrt{n}$ points at every iteration.

`NEWUOA`. We use `NEWUOA` through a Python wrapper `PDFO` [84]. We set `rhoend` $= 10^{-8}$ as a termination criterion for this algorithm. All other parameters are set to default, e.g., the number of interpolation point is set to $2n + 1$.

To summarize the per-iteration cost of the methods, `NEWUOA` incurs $O(n^2)$ flops of computation cost at every iteration due to interpolation, while `CMA-ES` requires $O(n^3)$ flops due to eigendecomposition of the covariance matrix. The computation cost of `CMA-ES` can be reduced to $O(n^2)$ when the decomposition is performed at every $n/10$ iterations [48].

We will investigate the above methods on both noiseless and noisy functions in Section 6.2 to study the effect of noise on their performance.

6.2. Numerical Experiments

We selected 73 unconstrained CUTEst problems with varying dimensions up to 200 to test the performance of `NEWUOA` and `CMA-ES`. We compare them on problems with and without noise to simulate different settings of DFO problems.

Several metrics have been proposed in the literature to study the relative performance of different methods, including performance profiles, data profiles and the log-ratio profiles [67, 69]. We choose the log-ratio profile [67] in our study, as in contrast to the other two options, it provides both an overview of the general performance of the methods and their relative performance for each specific problem. Specifically, the height of the plotted bars and the area of the shaded region can offer insights into the performance of the algorithms with relative scale, although it works the best when only two methods are being compared.

We present our empirical findings based on accuracy and efficiency of the algorithms.

6.2.1. Noiseless Functions

We first present the case when the functions do not contain noise, i.e., the true objective is always returned in double precision.

Accuracy. To measure the accuracy of the final solutions obtained by `CMA-ES` and `NEWUOA`, we vary the budget, i.e. total number of function evaluations allowed, from $0.5n$ to $500n$ and compare the optimality gap achieved by them. Formally, we compare the relative accuracy of `CMA-ES` and `NEWUOA` up to 8 digits by computing the log-ratio:

$$\log_2 \frac{\max(\phi_{\text{CMA-ES}} - \phi^*, 10^{-8})}{\max(\phi_{\text{NEWUOA}} - \phi^*, 10^{-8})} \tag{6.2.1}$$

where $\phi_{\text{CMA-ES}}$ and ϕ_{NEWUOA} denote the best objectives achieved by the corresponding algorithms within the given budget, and ϕ^* denotes the optimal objective value. We plot (6.2.1) in an increasing order so that the areas of the shaded regions give a general overview of the relative performance of the compared methods, with each bar demonstrating the relative accuracy for a given problem.

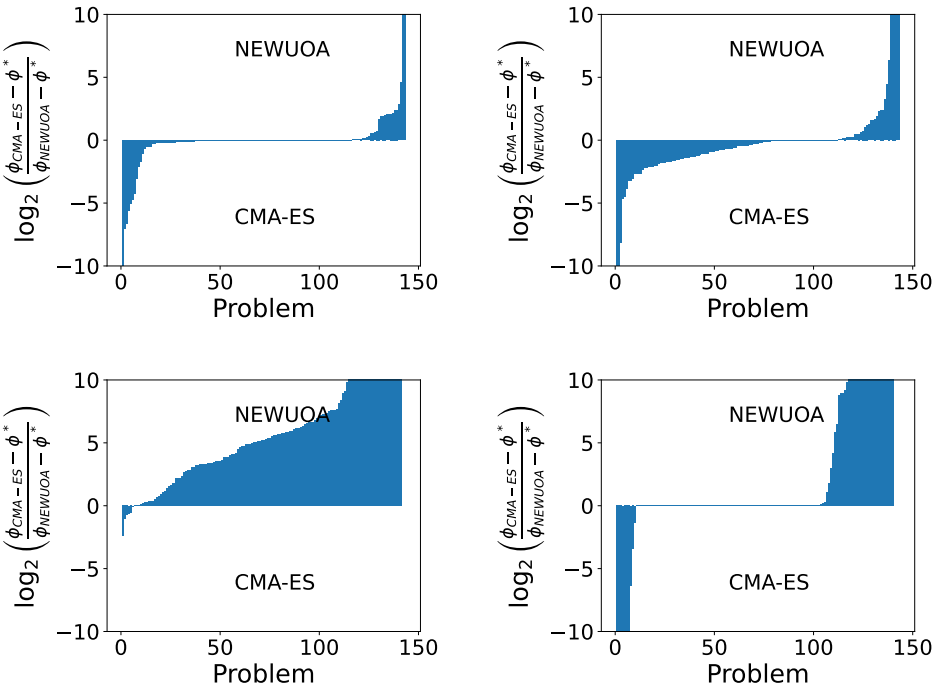

Figure 6.1. *(Selected) Accuracy Log-Ratio Profiles for Noiseless Problems.* Plots of (6.2.1) comparing `CMA-ES` and `NEWUOA` within a budget of $0.5n$ (upper left), $2n$ (upper right), $10n$ (bottom left) and $500n$ (bottom right).

As observed in Figure 6.1, `CMA-ES` can achieve higher accuracy than `NEWUOA` when the allowed number of function evaluations is limited to less than $2n$. This behavior aligns with our expectations since `NEWUOA` requires $2n + 1$ function evaluations to perform the first iteration. However, as the allowed budget for function evaluations increases, `NEWUOA`

outperforms `CMA-ES` on nearly all problems. Nevertheless, it is worth noting that when a significantly larger budget of $500n$ function evaluations is allowed, the performance gap between `NEWUOA` and `CMA-ES` is narrowed. This indicates that `CMA-ES` can require a considerate number of function evaluations to reach the solution.

Efficiency. We can further study the relative performance of `CMA-ES` and `NEWUOA` by checking the minimum number of function evaluations required by each method to satisfy

$$\phi(x_k) - \phi^* \leq \tau \cdot \max(1, |\phi^*|) \tag{6.2.2}$$

where ϕ^* is the optimal objective value and τ is a pre-specified accuracy level. We denote the required number of function evaluations as $\text{eval}_{\text{CMA-ES}}$ and $\text{eval}_{\text{NEWUOA}}$. We fix the budget as $500n$ function evaluations and vary τ from 0.1 to 10^{-8} to study the performance with respect to various accuracy levels and plot the log-ratio

$$\log_2 \frac{\text{eval}_{\text{CMA-ES}}}{\text{eval}_{\text{NEWUOA}}} \tag{6.2.3}$$

in an increasing order in Figure 6.2.

We observe that `NEWUOA` is much more efficient than `CMA-ES` in achieving (6.2.2) across nearly all problems, regardless of the accuracy level. Therefore, when dealing with noiseless DFO problems where attaining a specific accuracy level is deemed sufficient or when the budget is sufficiently large for model construction, `NEWUOA` is more preferable. Otherwise, if only fewer than $2n$ function evaluations are available, `CMA-ES` can be served as an alternative for potential higher accuracy.

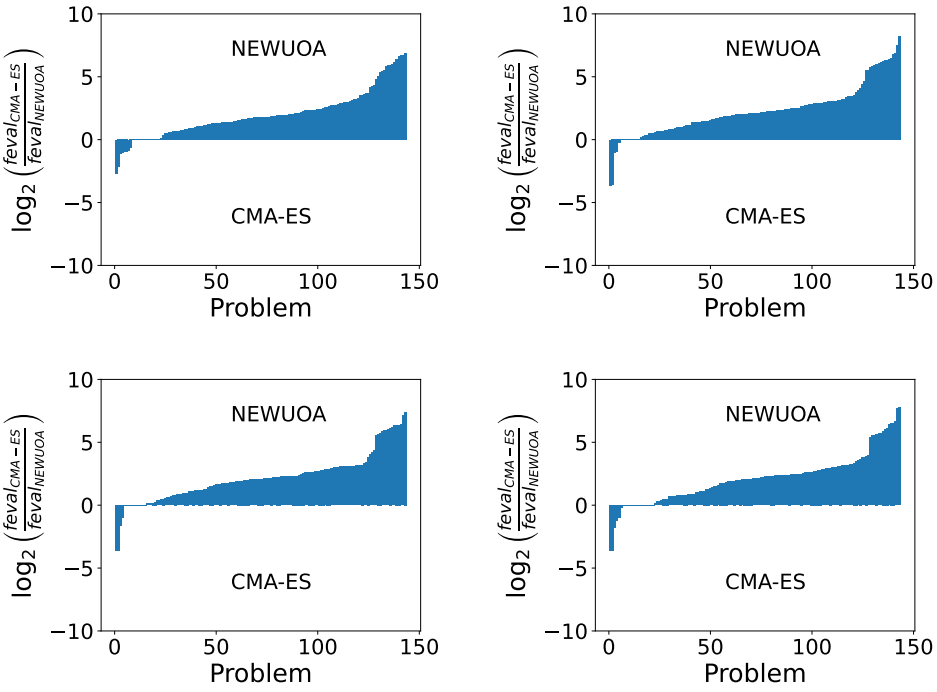

Figure 6.2. *(Selected) Efficiency Log-Ratio Profiles for Noiseless Problems.* Plots of (6.2.3) comparing `CMA-ES` and `NEWUOA` with $\tau = 0.1$ (upper left), 10^{-4} (upper right), 10^{-6} (bottom left) and 10^{-8} (bottom right).

6.2.2. Noisy Functions

We now present results for problems with stochastic noise. To simulate a noisy setting, we inject i.i.d Gaussian noise with mean zero and variance ϵ_f^2 to the 73 noiseless CUTEst functions:

$$f(x;\xi) = \phi(x) + \epsilon(x;\xi), \quad \epsilon(x;\xi) \sim \mathcal{N}(0, \epsilon_f^2), \tag{6.2.4}$$

where ϵ_f is denoted as the noise level of the problem.

Accuracy. To investigate the robustness of both `CMA-ES` and `NEWUOA` against computation errors, we ran the solvers using the same parameter settings as in the noiseless

case. Similar to Section 6.2.1, we measure the best true objective value obtained by both methods within budgets ranging from $0.5n$ to $500n$ function evaluations. We report the log-ratio profiles in Figure 6.3 for a noise level of 10^{-3}, which is representative for other noise levels. The log-ratio profiles reveal that for very small budgets ($\leq 2n$), `CMA-ES` is slightly more competitive than `NEWUOA`. However, as the budget increases to medium level, `NEWUOA` consistently outperforms `CMA-ES` in terms of final accuracy. Intriguingly, when more generous budgets of function evaluations are allowed, `CMA-ES` demonstrates its ability to achieve significantly higher accuracy than `NEWUOA` across a majority of the problems.

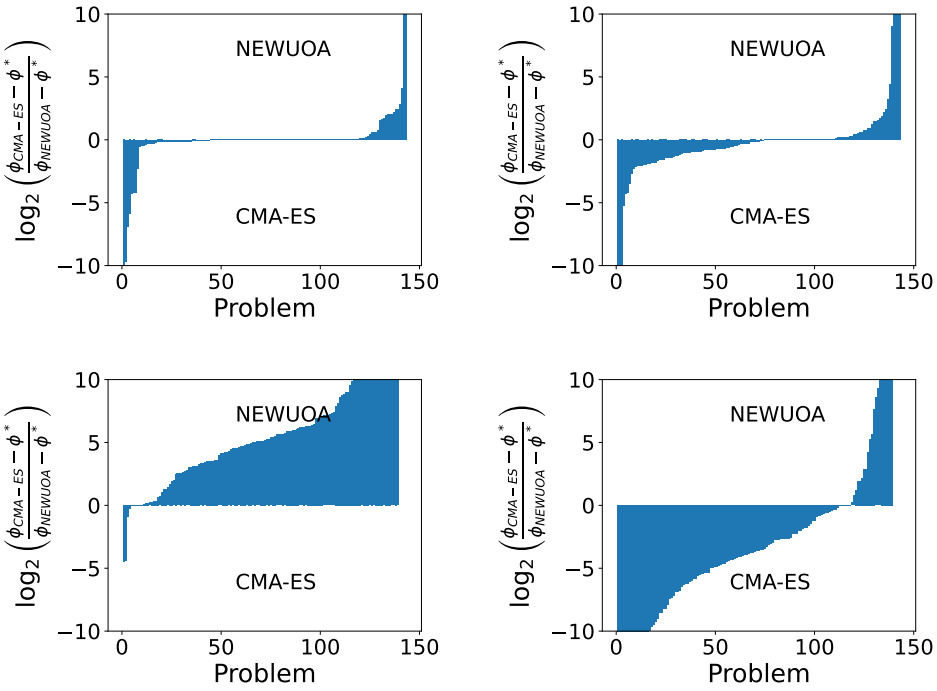

Figure 6.3. *(Selected) Accuracy Log-Ratio Profiles for Noisy Problems ($\epsilon_f = 10^{-3}$).* Plots of (6.2.1) comparing `CMA-ES` and `NEWUOA` within a budget of $0.5n$ (upper left), $2n$ (upper right), $10n$ (bottom left) and $500n$ (bottom right).

Efficiency. We evaluate the efficiency of `CMA-ES` and `NEWUOA` in the noisy setting by comparing the number of function evaluations, denoted as $\text{eval}_{\text{CMA-ES}}$ $\text{eval}_{\text{NEWUOA}}$, that are required to achieve

$$\phi(x_k) - \phi^* \leq \tau \cdot (\phi(x_0) - \phi^*), \tag{6.2.5}$$

where τ varies from 0.1 to 10^{-8} and ϕ^* denotes the optimal objective value. We plot the log-ratio (6.2.3) in Figure 6.4 and 6.5 to report the efficiency of the methods corresponding to noise levels of 10^{-1} and 10^{-7} respectively. Our observations indicate that `NEWUOA` is more efficient than `CMA-ES` in obtaining a given accuracy level when the desired accuracy level is low or the noise level is low. For highly noisy problems, `CMA-ES` and `NEWUOA` are competitive in achieving high accuracy levels. This finding highlights the advantage of `CMA-ES` when an abundant number of function evaluations are available for solving noisy DFO problems, especially when high accuracy is desirable.

We note that, compared to the noiseless case, the efficiency of `NEWUOA` in solving functions with high noise levels and the final accuracy obtained by it under large budgets are both impacted by noise. Although `NEWUOA` is more competitive than `CMA-ES` in solving noiseless problems, with the presence of noise, `CMA-ES` is more capable of achieving high accuracy especially for large noise levels. This implies that the presence of noise poses additional challenges for `NEWUOA`, impacting its effectiveness in solving unconstrained DFO problems. In light of this finding, we briefly discuss in the next section the effect of noise on the performance of an IBO method and potential strategies to mitigate the effects of noise on the performance of `NEWUOA` and other IBO methods.

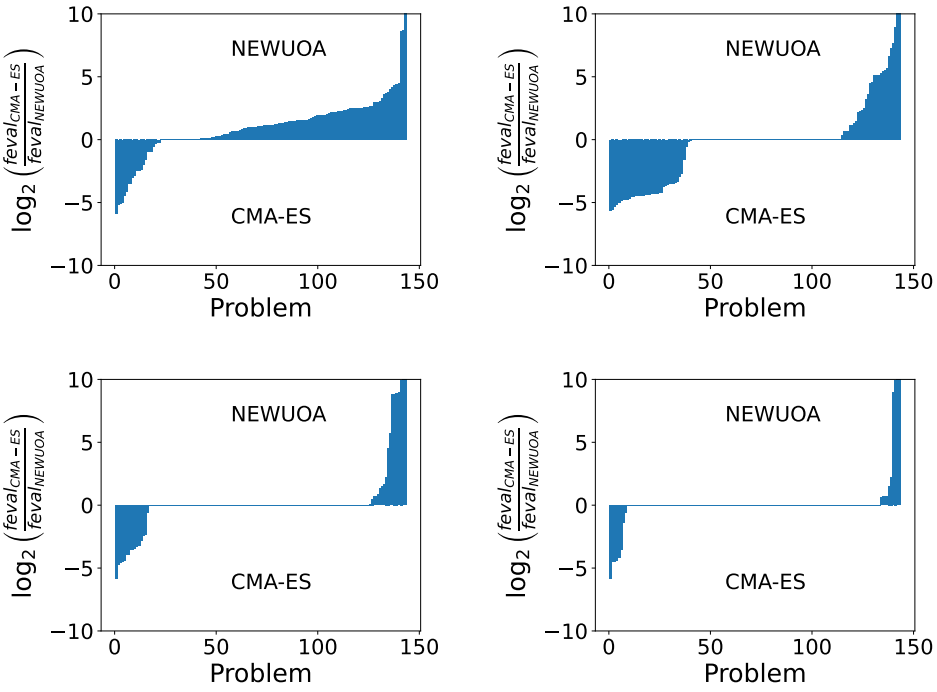

Figure 6.4. *(Selected) Efficiency Log-Ratio Profiles for Noisy Problems ($\epsilon_f = 0.1$).* Plots of (6.2.3) comparing `CMA-ES` and `NEWUOA` with $\tau = 0.1$ (upper left), 10^{-4} (upper right), 10^{-6} (bottom left) and 10^{-8} (bottom right).

6.3. On the Accuracy of IBO methods in the Presence of Noise

Recall from Chapter 1 that IBO methods rely on model construction and ratio tests to perform updates within a trust region framework. With the presence of noise in the objective function, the quadratic model can be inaccurate due to the noisy function values of the interpolation points. Moreover, the noise can cause the ratio test to suggest misleading suggestions for step acceptance or rejection, as the noise can lead to arbitrarily wrong ratio values. Despite these challenges, our numerical results suggest that `NEWUOA` still demonstrates reasonable progress even when noise is present. To shed light on this observation, we present below a result that provides an upper bound for the approximation

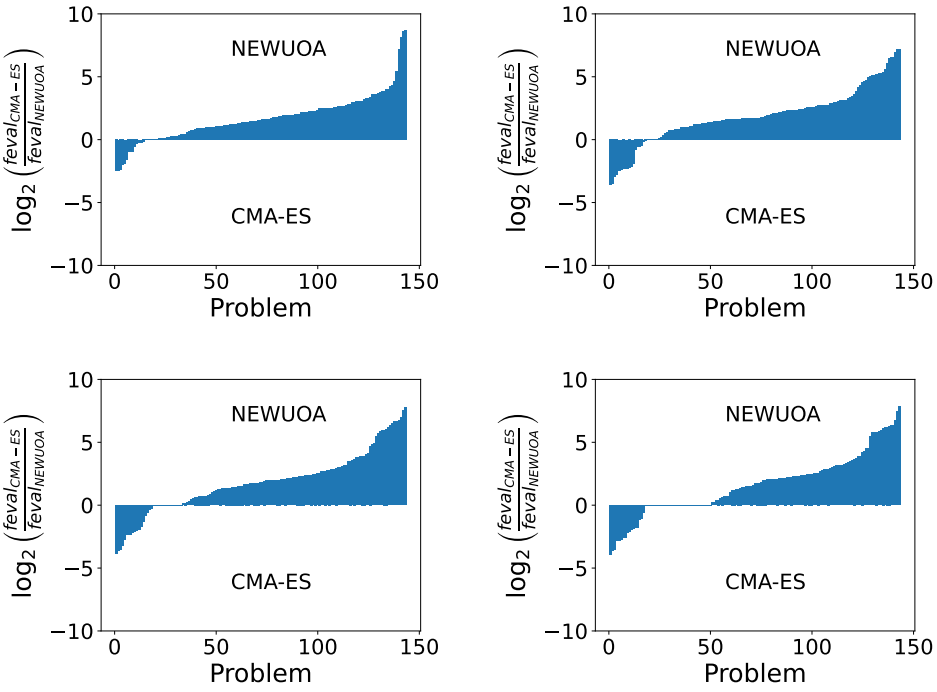

Figure 6.5. *(Selected) Efficiency Log-Ratio Profiles for Noisy Problems ($\epsilon_f = 10^{-7}$).* Plots of (6.2.3) comparing `CMA-ES` and `NEWUOA` with $\tau = 0.1$ (upper left), 10^{-2} (upper right), 10^{-4} (bottom left) and 10^{-6} (bottom right).

error of the underdetermined quadratic model $m(x)$, when bounded noise is present. See Appendix D for a proof based on [29]. A similar bound can be obtained by considering the mean squared approximation error.

Theorem 6.3.1. *Suppose $|\epsilon(x)| \leq \epsilon_f, \forall x \in \mathbb{R}^n$ and $\phi : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable with $\nabla\phi$ being L_2-Lipschitz continuous. We further assume that the interpolation set $Y = \{y^0, y^1, ..., y^p\}$ is poised in $B(y^0; \Delta)$. Then for all $x \in B(y^0; \Delta)$, we have that*

$$\|\nabla\phi(x) - \nabla m(x)\| \leq \frac{5\sqrt{p}}{2}\|\hat{M}^\dagger\|(L_2 + \|H\|)\Delta + \frac{2\sqrt{p}\|\hat{M}^\dagger\|\epsilon_f}{\Delta} \tag{6.3.1}$$

$$|\phi(x) - m(x)| \leq \frac{5\sqrt{p}\|\hat{M}^\dagger\| + 1}{2}(\|H\| + L_2)\Delta^2 + (2\sqrt{p}\|\hat{M}^\dagger\| + 1)\epsilon_f \tag{6.3.2}$$

where $\hat{M}^\dagger = (\hat{M}^T\hat{M})^{-1}\hat{M}^T$ *and*

$$\hat{M} = \begin{bmatrix} (y^1 - y^0)/\Delta & (y^2 - y^0)/\Delta & \dots & (y^p - y^0)/\Delta \end{bmatrix}$$

Note that in contrast to the noiseless case when $\epsilon_f = 0$, for which the upperbound in (6.3.1) and (6.3.2) shrink to zero, instead the upperbound for (6.3.1) tends to infinity due to the additional noise term. Therefore, when the trust region is sufficiently large and the noise term is dominated by the other term, one can expect IBO methods to make progress similarly as in the noiseless case. However, one cannot expect IBO methods to progress effectively once the trust region becomes too small. Indeed, by minimizing the right-hand-side of (6.3.1), the trust region that minimizes the approximation is $\underline{\Delta} = 2\sqrt{\frac{\epsilon_f}{5(\|H\|+L_2)}}$.

We present in Figure 6.6 an example of `NEWUOA` failing to make progress when the initial trust region is too small. We observe that when noise is not present in the function, `NEWUOA` is able to achieve similar accuracy level no matter what value the initial trust region is set to, although efficiency of the algorithm may differ. However, in the presence of noise, `NEWUOA` may stagnate when the trust region is too small with respect to the noise level (and potentially other characteristics of the function and interpolation set), with the algorithm keeps shrinking the trust region.

It has been found in [93] that using more interpolation points can be beneficial in increasing the final accuracy of `NEWUOA`. However, as there is an upper limit in the number of interpolations in quadratic model construction (we do not consider regression model), we briefly discuss three strategies that can improve the accuracy of an IBO method in the presence of noise.

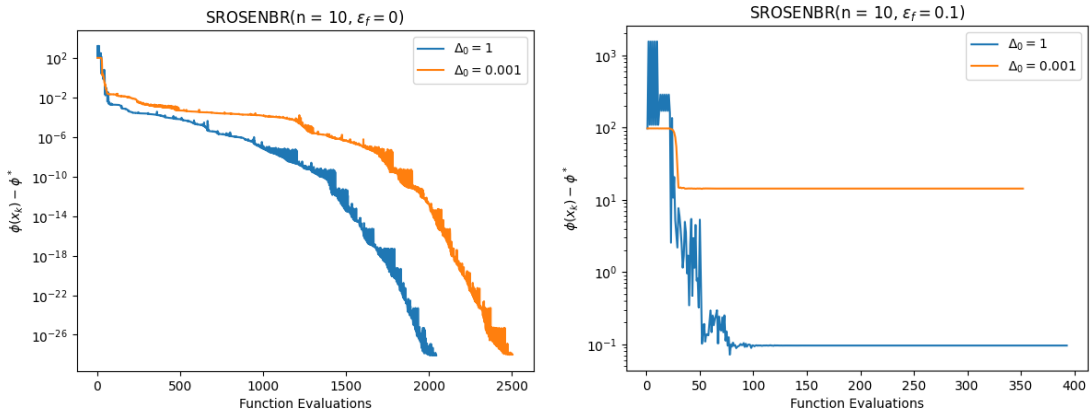

Figure 6.6. Solving noiseless and noisy SROSENBR function with 10 variables using different initial trust region Δ_0. Left: noiseless. Right: noise level = 0.1.

Restarts. A simple yet effective strategy of achieving higher accuracy for `NEWUOA` is to employ restarts. To elaborate, instead of terminate `NEWUOA` when the trust region converges to the minimum size (10^{-8} in our setting), we restart the algorithm at the returned solution by considering it as the new initial point. Therefore, we abandon every point in the interpolation set and reset the trust region radius to the initial value. We repeat this process until we have reached the allowable budget, i.e., the maximum number of function evaluations. We have observed effectiveness of this approach in our empirical studies and we visualize the findings in the log-ratio plots in Figure 6.7 and Figure 6.8.

However, this naive approach can be deemed inefficient as in Figure 6.9. Note that by increasing the trust region to large size, the algorithm evaluates at a great number of points that have high function values and are far from the solution. We speculate that more effective restart procedure can be developed such that the function evaluations are used more efficiently. For instance, by using a smaller initial trust region for later runs or by setting the minimum trust region to be on the order of the noise level allows more

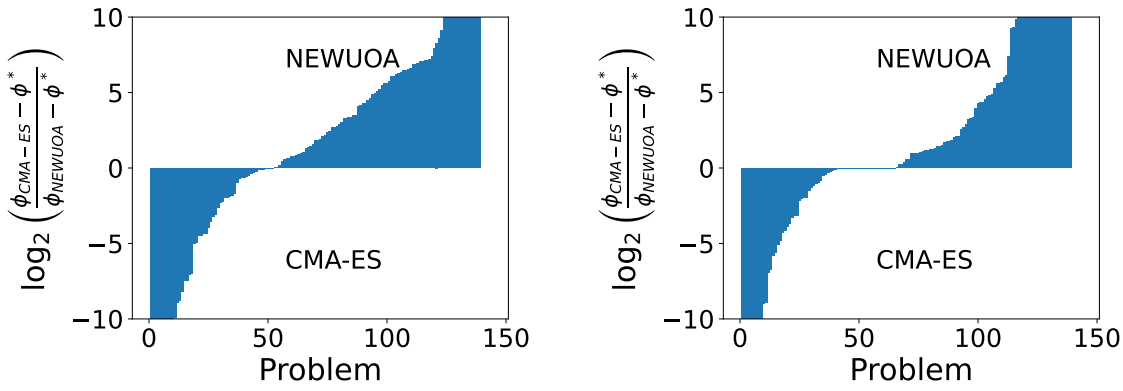

Figure 6.7. *(Selected) Accuracy Log-Ratio Profiles for Noisy Problems with noise level* 0.001 *(Left) and* 10^{-7} *(Right).* Plots of (6.2.1) comparing `CMA-ES` and `NEWUOA` with restarts within a budget of $500n$.

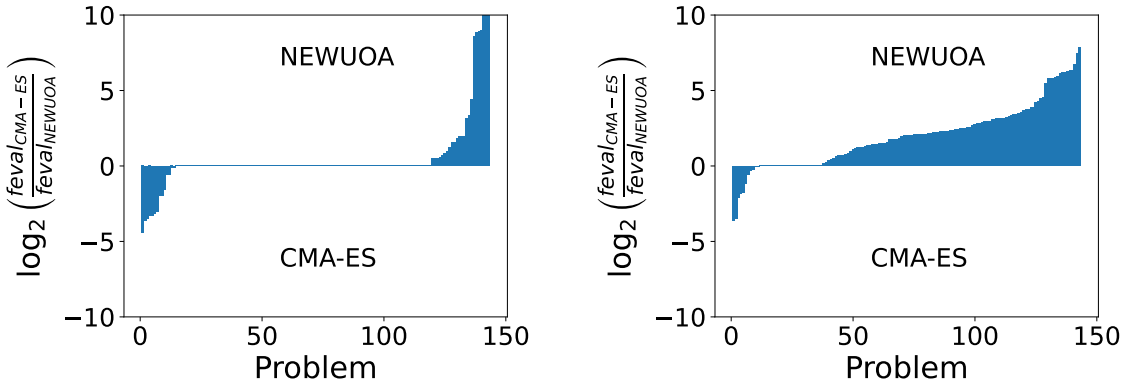

Figure 6.8. *(Selected) Efficiency Log-Ratio Profiles for Noisy Problems with noise level* 0.001 *(Left) and* 10^{-7} *(Right).* Plots of (6.2.3) comparing `CMA-ES` and `NEWUOA` with restarts for $\tau = 10^{-8}$.

restarts and potentially higher accuracy. See [18] for a restart strategy for noisy least square problems.

Ratio Test Relaxation. As suggested in [99], applying relaxation to the ratio test can mitigate the effect of noise in the objective, particularly to prevent the trust region from becoming excessively small. Therefore, we modify `NEWUOA` such that the ratio test is

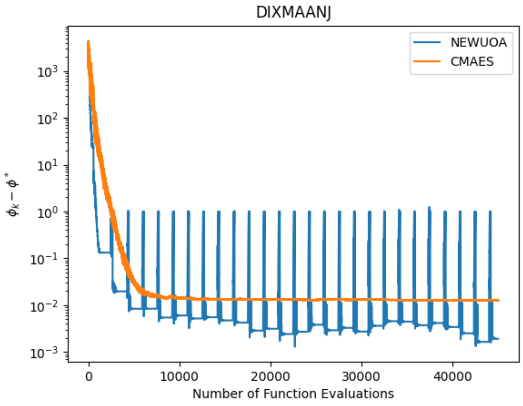

Figure 6.9. Solving `DIXMAANJ` with 90 variables using `NEWUOA` with restarts and `CMA-ES`. The noise level is 10^{-3}.

relaxed as

$$\frac{f(x_k) - f(x_k + s) + 2\epsilon_f}{m_k(x_k) - m_k(x_k + s) + 2\epsilon_f}. \tag{6.3.3}$$

However, contrary to our expectations, our numerical experiments demonstrate that relaxing the ratio test for `NEWUOA` does not improve the final accuracy overall. We present the accuracy log-ratio plots in Figure 6.10. Interestingly, we have observed accuracy improvements in `DFOTR`, a different IBO algorithm, when performing the same ratio relaxation. This outcome may be attributed to the utilization of a trust region lower bound in `NEWUOA`, which prevents the algorithm from accepting steps that are too small with respect to this bounds. Indeed, as shown in Figure 6.11, `NEWUOA` is unable to recover from an initial trust region is too small even though the ratio test has been relaxed. This observation indicates that more sophisticated design may be required to further improve the robustness of `NEWUOA` for solving noisy DFO problems.

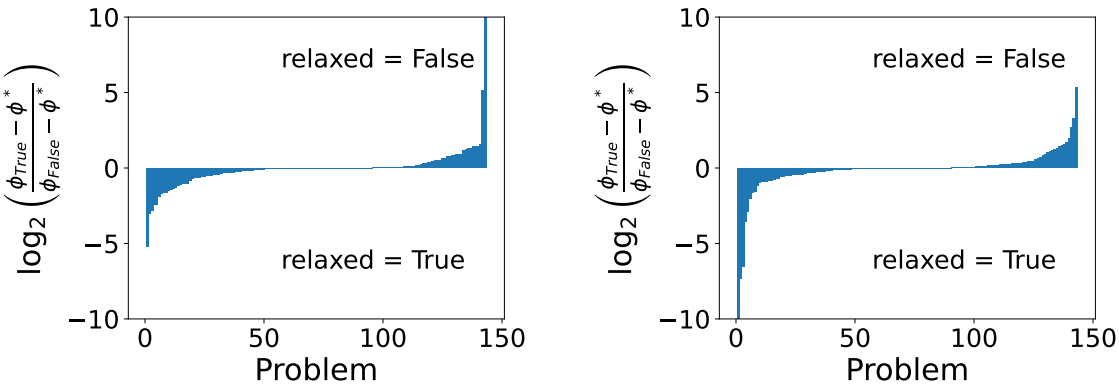

Figure 6.10. *(Selected) Accuracy Log-Ratio Profiles for Noisy Problems with noise level* 0.001 *(Left) and* 10^{-7} *(Right).* Plots of (6.2.1) comparing `NEWUOA` with and without restarts using a budget of $500n$.

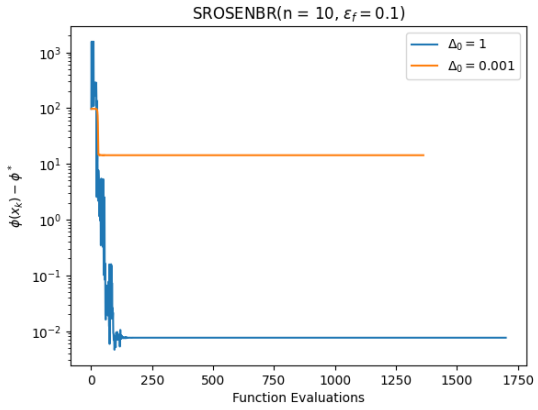

Figure 6.11. Solving noisy SROSENBR function with 10 variables and noise level of 0.1 using different initial trust region Δ_0. The ratio test in `NEWUOA` is relaxed as in (6.3.3).

Adaptive Sampling. When the noise term $\epsilon(x)$ in the problem (6.1.2) is stochastic, another viable approach for obtaining an higher accuracy is to employ sample averaging so that the function value at each point is computed as an average of multiple evaluations at the same point [20, 31, 89]. The noise level of the function is therefore reduced due to variance reduction. An adaptive sample averaging strategy can balance the tradeoff

between the efficiency of the IBO method and the final accuracy. However, theoretical convergence of existing adaptive sample averaging based IBO methods often require a sample size of $O(\Delta_k^{-4})$. This is not desirable even though in practice the sample size is chosen as $O(\Delta_k^{-1})$, as it leads to inefficiency in early iterations. Since IBO methods can often make progress before the trust region become too small, a more sophisticated sampling strategy may be available. For instance, the sample size may only need to be increased when the trust region reaches a given level, say the trust region size that minimizes the upper bound of (6.3.1). Yet in our preliminary studies we have observed difficulties in the algorithm in making progress when the trust region is not adequately enlarged and the interpolation set is not adjusted. We speculate that an optimal adaptive sampling strategy requires careful handling of factors such as the sample size, trust region management and positioning of the interpolation sets.

6.4. Final Remarks

We have performed a comparative study of `NEWUOA` and `CMA-ES` on a broad set of unconstrained problems in this chapter. We have observed that `NEWUOA` outperforms `CMA-ES` in terms of efficiency and accuracy when the function evaluations are exact. Notably, our findings indicate that, in the presence of noise, `NEWUOA` is still more efficient than `CMA-ES` for obtaining desirable accuracy levels within a moderate number of function evaluations. Yet as opposed to the noiseless case, it obtains lower final accuracy than `CMA-ES` when a large budget of function evaluations are available.

This performance gap motivates us to analyze the challenges associated with noisy evaluations for IBO methods and to understand potential strategies for improving their

performance in the presence of noise. In particular, our preliminary empirical results suggest that utilizing naive restarts with `NEWUOA` can significantly narrow down the performance gap between `CMA-ES` in terms of final accuracy. However, relaxing the ratio test does not lead to improved accuracy of `NEWUOA` as opposed to other IBO methods such as `DFO-TR`. In addition, the effectiveness of adaptive sampling is contingent upon further investigation. These findings highlight the importance of designing more efficient restarts for improving the accuracy, the limited impact of relaxing the ratio tests, the need for more meticulous design of adaptive sampling, and the necessity of further redesign of the classical IBO methods in the presence of noise.

References

[1] Aghajanyan, A., Zettlemoyer, L., and Gupta, S. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255* (2020).

[2] Audet, C., and Hare, W. Derivative-free and blackbox optimization.

[3] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. Dbpedia: A nucleus for a web of open data. In *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007+ ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings* (2007), Springer, pp. 722–735.

[4] Balakrishna, R., Antoniou, C., Ben-Akiva, M., Koutsopoulos, H. N., and Wen, Y. Calibration of microscopic traffic simulation models: Methods and application. *Transportation Research Record 1999*, 1 (2007), 198–207.

[5] Bandeira, A. S., Scheinberg, K., and Vicente, L. N. Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization. *Mathematical programming 134*, 1 (2012), 223–257.

[6] Barton, R. R. Computing forward difference derivatives in engineering optimization. *Engineering optimization 20*, 3 (1992), 205–224.

[7] Berahas, A. S., Byrd, R. H., and Nocedal, J. Derivative-free optimization of noisy functions via quasi-Newton methods. *SIAM Journal on Optimization 29*, 2 (2019), 965–993.

[8] Berahas, A. S., Cao, L., Choromanski, K., and Scheinberg, K. Linear interpolation gives better gradients than Gaussian smoothing in derivative-free optimization. *arXiv preprint arXiv:1905.13043* (2019).

[9] Berahas, A. S., Cao, L., Choromanski, K., and Scheinberg, K. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *arXiv preprint arXiv:1905.01332* (2019).

[10] Berahas, A. S., Cao, L., Choromanski, K., and Scheinberg, K. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Foundations of Computational Mathematics* (2021), 1–54.

[11] Berghen, F. V., and Bersini, H. Condor, a new parallel, constrained extension of powell's uobyqa algorithm: Experimental results and comparison with the dfo algorithm. *Journal of computational and applied mathematics 181*, 1 (2005), 157–175.

[12] Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326* (2015).

[13] Branch, M. A., Coleman, T. F., and Li, Y. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing 21*, 1 (1999), 1–23.

[14] Brekelmans, R. C. M., Driessen, L. T., Hamers, H. J. M., and Den Hertog, D. Gradient estimation schemes for noisy functions. *Journal of Optimization Theory and Applications 126*, 3 (2005), 529–551.

[15] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems 33* (2020), 1877–1901.

[16] Byrd, R. H., Nocedal, J., and Waltz, R. KNITRO: An integrated package for nonlinear optimization. In *Large-Scale Nonlinear Optimization* (2006), G. di Pillo and M. Roma, Eds., Springer, pp. 35–59.

[17] Cao, L., Berahas, A. S., and Scheinberg, K. First-and second-order high probability complexity bounds for trust-region methods with noisy oracles. *arXiv preprint arXiv:2205.03667* (2022).

[18] Cartis, C., Fiala, J., Marteau, B., and Roberts, L. Improving the flexibility and robustness of model-based derivative-free optimization solvers. *ACM Transactions on Mathematical Software (TOMS) 45*, 3 (2019), 1–41.

[19] Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (2017), ACM, pp. 15–26.

[20] Chen, R., Menickelly, M., and Scheinberg, K. Stochastic optimization using

a trust-region method and random models. *Mathematical Programming 169*, 2 (2018), 447–487.

[21] Choi, T., and Kelley, C. T. Superlinear convergence and implicit filtering. *SIAM Journal on Optimization 10*, 4 (2000), 1149–1162.

[22] Choromanski, K., Pacchiano, A., Parker-Holder, J., Tang, Y., Jain, D., Yang, Y., Iscen, A., Hsu, J., and Sindhwani, V. Provably robust blackbox optimization for reinforcement learning, 2019.

[23] Conejo, P., Karas, E. W., and Pedroso, L. G. A trust-region derivative-free algorithm for constrained optimization. *Optimization Methods and Software 30*, 6 (2015), 1126–1145.

[24] Conejo, P., Karas, E. W., Pedroso, L. G., Ribeiro, A. A., and Sachine, M. Global convergence of trust-region algorithms for convex constrained minimization without derivatives. *Applied Mathematics and Computation 220* (2013), 324–330.

[25] Conn, A. R., Scheinberg, K., and Toint, P. L. On the convergence of derivative-free methods for unconstrained optimization. *Approximation theory and optimization: tributes to MJD Powell* (1997), 83–108.

[26] Conn, A. R., Scheinberg, K., and Toint, P. L. A derivative free optimization algorithm in practice. In *Proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO* (1998), vol. 48, p. 3.

[27] Conn, A. R., Scheinberg, K., and Vicente, L. Error estimates and poisedness in multivariate polynomial interpolation. Tech. rep., IBM T. J. Watson Research Center, 2006.

[28] Conn, A. R., Scheinberg, K., and Vicente, L. Geometry of interpolation sets in derivative free optimization. *Mathematical Programming, Series A 111* (2007), 141–172.

[29] Conn, A. R., Scheinberg, K., and Vicente, L. N. *Introduction to derivative-free optimization*, vol. 8. SIAM, 2009.

[30] Curtis, A. R., and Reid, J. K. The choice of step lengths when using differences to approximate Jacobian matrices. *IMA Journal of Applied Mathematics 13*, 1 (1974), 121–126.

[31] Deng, G., and Ferris, M. C. Adaptation of the uobyqa algorithm for noisy functions. In *Proceedings of the 2006 winter simulation conference* (2006), IEEE,

pp. 312–319.

[32] Deng, M., Wang, J., Hsieh, C.-P., Wang, Y., Guo, H., Shu, T., Song, M., Xing, E. P., and Hu, Z. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548* (2022).

[33] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[34] Diao, S., Huang, Z., Xu, R., Li, X., Lin, Y., Zhou, X., and Zhang, T. Black-box prompt learning for pre-trained language models. *arXiv preprint arXiv:2201.08531* (2022).

[35] Digabel, S. L., and Wild, S. M. A taxonomy of constraints in simulation-based optimization. *arXiv preprint arXiv:1505.07881* (2015).

[36] Diouane, Y., Gratton, S., and Vicente, L. N. Globally convergent evolution strategies. *Mathematical Programming 152* (2015), 467–490.

[37] Eriksson, D., Bindel, D., and Shoemaker, C. A. pysot and poap: An event-driven asynchronous framework for surrogate optimization. *arXiv preprint arXiv:1908.00420* (2019).

[38] Fornberg, B. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of computation 51*, 184 (1988), 699–706.

[39] Frazier, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811* (2018).

[40] Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. Computing forward-difference intervals for numerical optimization. *SIAM Journal on Scientific and Statistical Computing 4*, 2 (1983), 310–321.

[41] Gill, P. E., Murray, W., and Wright, M. H. *Practical Optimization*. Academic Press, London, 1981.

[42] Gould, N. I., 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 (2015), 545–557.

[43] Gould, N. I. M., Orban, D., and Toint, P. L. GALAHAD, a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. *ACM Trans.*

Math. Softw. 29, 4 (2003), 353–372.

[44] Gray, G. A., and Kolda, T. G. Algorithm 856: Appspack 4.0: Asynchronous parallel pattern search for derivative-free optimization. *ACM Transactions on Mathematical Software (TOMS) 32*, 3 (2006), 485–507.

[45] Hansen, N. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[46] Hansen, N., Arnold, D. V., and Auger, A. Evolution strategies. *Springer handbook of computational intelligence* (2015), 871–898.

[47] Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation* (2010), pp. 1689–1696.

[48] Hansen, N., and Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation 9*, 2 (2001), 159–195.

[49] Hare, W., Jarry-Bolduc, G., and Planiden, C. Error bounds for overdetermined and underdetermined generalized centred simplex gradients. *arXiv preprint arXiv:2006.00742* (2020).

[50] Hare, W., and Srivastava, K. Applying complex-step derivative approximations in model-based derivative-free optimization.

[51] Hough, M., and Roberts, L. Model-based derivative-free methods for convex-constrained optimization. *arXiv preprint arXiv:2111.05443* (2021).

[52] Howard, J., and Ruder, S. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146* (2018).

[53] Jin, B., Scheinberg, K., and Xie, M. High probability complexity bounds for line search based on stochastic oracles. *Advances in Neural Information Processing Systems 34* (2021).

[54] Junges, S., Jansen, N., Dehnert, C., Topcu, U., and Katoen, J.-P. Safety-constrained reinforcement learning for mdps. *Lecture Notes in Computer Science* (2016), 130–146.

[55] Kelley, C. T. *Implicit filtering*, vol. 23. SIAM, 2011.

[56] Kiefer, J., Wolfowitz, J., et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics 23*, 3 (1952), 462–466.

[57] Kimiaei, M. Line search in noisy unconstrained black box optimization. Tech. rep., Technical report, University of Vienna, 2020.

[58] Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. Fast bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics 11*, 2 (2017), 4945–4968.

[59] Larson, J., Menickelly, M., and Wild, S. M. Derivative-free optimization methods. *Acta Numerica 28* (2019), 287–404.

[60] Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).

[61] Li, X. L., and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190* (2021).

[62] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys 55*, 9 (2023), 1–35.

[63] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[64] Loshchilov, I., and Hutter, F. Cma-es for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269* (2016).

[65] Lyness, J. N. Has numerical differentiation a future. In *Proceedings Seventh Manitoba Conference on Numerical Mathematics, Utilitas Mathematica Publishing* (1977).

[66] Lyness, J. N., and Moler, C. B. Numerical differentiation of analytic functions. *SIAM Journal on Numerical Analysis 4*, 2 (1967), 202–210.

[67] Morales, J. L. A numerical study of limited memory BFGS methods, 2002. Applied Mathematics Letters.

[68] Moré, J. J., Garbow, B. S., and Hillstrom, K. E. User guide for MINPACK-1. Tech. Rep. 80–74, Argonne National Laboratory, Argonne, Illinois, USA, 1980.

[69] Moré, J. J., and Wild, S. M. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization 20*, 1 (2009), 172–191.

[70] Moré, J. J., and Wild, S. M. Estimating computational noise. *SIAM Journal on Scientific Computing 33*, 3 (2011), 1292–1314.

[71] Moré, J. J., and Wild, S. M. Estimating derivatives of noisy simulations. *ACM Transactions on Mathematical Software (TOMS) 38*, 3 (2012), 19.

[72] Nelder, J. A., and Mead, R. A simplex method for function minimization. *Computer Journal 7* (1965), 308–313.

[73] Nesterov, Y., and Spokoiny, V. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics 17*, 2 (2017), 527–566.

[74] Neumaier, A. Complete search in continuous global optimization and constraint satisfaction. *Acta numerica 13* (2004), 271–369.

[75] Nocedal, J., and Wright, S. *Numerical Optimization*, 2 ed. Springer New York, 1999.

[76] Powell, M. Linearly constrained optimization algorithm. Tech. rep., Cambridge University, 2005.

[77] Powell, M. J. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*. Springer, 1994, pp. 51–67.

[78] Powell, M. J. The NEWUOA software for unconstrained optimization without derivatives. In *Large-scale nonlinear optimization*. Springer, 2006, pp. 255–297.

[79] Powell, M. J. The bobyqa algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge* (2009), 26–46.

[80] Powell, M. J. On fast trust region methods for quadratic models with linear constraints. *Mathematical Programming Computation 7*, 3 (2015), 237–267.

[81] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog 1*, 8 (2019), 9.

[82] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M.,

Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research 21*, 1 (2020), 5485–5551.

[83] Ragonneau, T. M. Model-based derivative-free optimization methods and software. *arXiv preprint arXiv:2210.12018* (2022).

[84] Ragonneau, T. M., and Zhang, Z. PDFO: Cross-platform interfaces for Powell's derivative-free optimization solvers (version 1.0), 2020.

[85] Rios, L. M., and Sahinidis, N. V. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization 56*, 3 (2013), 1247–1293.

[86] Roberts, L., and Cartis, C. A derivative-free gauss–newton method. *Mathematical Programming Computation* (2019).

[87] Sahinidis, N. V., and Tawarmalani, M. *BARON 7.2.5: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2005.

[88] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[89] Shashaani, S., Hashemi, F. S., and Pasupathy, R. ASTRO-DF: A class of adaptive sampling trust-region algorithms for derivative-free simulation optimization. *optimization online* (2015).

[90] Shi, H.-J. M., Xie, Y., Byrd, R., and Nocedal, J. A noise-tolerant quasi-newton algorithm for unconstrained optimization. *arXiv preprint arXiv:2010.04352* (2020).

[91] Shi, H.-J. M., Xie, Y., Byrd, R., and Nocedal, J. A noise-tolerant quasi-newton algorithm for unconstrained optimization. *SIAM Journal on Optimization 32*, 1 (2022), 29–55.

[92] Shi, H.-J. M., Xie, Y., Xuan, M. Q., and Nocedal, J. Adaptive finite-difference interval estimation for noisy derivative-free optimization. *arXiv preprint arXiv:2110.06380* (2021).

[93] Shi, H.-J. M., Xuan, M. Q., Oztoprak, F., and Nocedal, J. On the numerical performance of derivative-free optimization methods based on finite-difference approximations. *arXiv preprint arXiv:2102.09762* (2021).

[94] Shi, M., Nocedal, J., Oztoprak, F., and Xuan, M. Additional numerical results for the paper: "On the numerical performance of finite-difference based methods for derivative-free optimization. Technical report, Northwestern University, Evanston, IL U.S.A., 2022.

[95] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (2013), pp. 1631–1642.

[96] Squire, W., and Trapp, G. Using complex variables to estimate derivatives of real functions. *SIAM review 40*, 1 (1998), 110–112.

[97] Stepleman, R. S., and Winarsky, N. D. Adaptive numerical differentiation. *Mathematics of Computation 33*, 148 (1979), 1257–1264.

[98] Stulp, F., and Sigaud, O. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621* (2012).

[99] Sun, S., and Nocedal, J. A trust region method for the optimization of noisy functions. *arXiv preprint arXiv:2201.00973* (2022).

[100] Sun, T., Shao, Y., Qian, H., Huang, X., and Qiu, X. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning* (2022), PMLR, pp. 20841–20855.

[101] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

[102] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

[103] Wild, S. M., Sarich, J., and Schunck, N. Derivative-free optimization for parameter estimation in computational nuclear physics. *Journal of Physics G: Nuclear and Particle Physics 42*, 3 (2015), 034031.

[104] Xie, Y., Byrd, R. H., and Nocedal, J. Analysis of the BFGS method with errors. *SIAM Journal on Optimization 30*, 1 (2020), 182–209.

[105] Zhang, H., Conn, A. R., and Scheinberg, K. A derivative-free algorithm for least-squares minimization. *SIAM Journal on Optimization 20*, 6 (2010), 3555–3576.

[106] Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. *Advances in neural information processing systems 28* (2015).

APPENDIX A

On the Numerical Performance of Finite-Difference Based Methods for Derivative-Free Optimization

A.1. Numerical Investigation of Lipschitz Estimation

A.1.1. Investigation of Theoretical Lipschitz Estimates

In Section 2.2.2, we approximated the bound on the second and third derivative L and M (or the Lipschitz constant of the first and second derivative) along the interval $I = \{x \pm tp : t \in [0, h_0]\}$ for $h_0 > 0$ every time finite-differencing is performed.

For forward-differencing, we employed the Moré and Wild heuristic [71]. The heuristic estimates the bound on the second derivative of a univariate function with noise. Assume $\phi : \mathbb{R} \to \mathbb{R}$ is univariate. If we let $\Delta(t) = f(x + t) - 2f(x) + f(x - t)$, then $t > 0$ must satisfy two conditions:

$$|\Delta(t)| \geq \tau_1 \epsilon_f, \qquad \tau_1 \gg 1 \tag{A.1.1}$$

$$|f(x \pm t) - f(x)| \leq \tau_2 \max\{|f(x)|, |f(x \pm t)|\}, \qquad \tau_2 \in (0, 1). \tag{A.1.2}$$

In practice, $\tau_1 = 100$ and $\tau_2 = 0.1$. The first condition ensures that h is sufficiently large such that the second-order difference is not dominated by noise, while the second condition enforces that the difference is not dominated by a particular evaluation, so that there is

"equal" contribution from each function evaluation in the finite-difference approximation. If conditions (A.1.1) and (A.1.2) are satisfied, then we take $L = \max\{10^{-1}, |\Delta(t)|/t^2\}$.

For central differencing, we used a theoretical estimate based on knowledge of the true Hessian $\nabla^2\phi(x)$:

$$M = \max\left\{10^{-1}, \frac{|p^T(\nabla^2\phi(x + \tilde{h}\frac{p}{\|p\|}) - \nabla^2\phi(x))p|}{\tilde{h}\|p\|^2}\right\} \tag{A.1.3}$$

where $\tilde{h} = \sqrt{\epsilon_M}$. Since this estimate is ideal, we do not include the cost of evaluating M in the number of function evaluations. In both cases, the maximum is taken to ensure that the bound is strictly bounded away from 0 so that (2.2.9) and (2.2.10) remain well-defined.

One may ask whether or not a refined choice of L or M is necessary for finite-difference L-BFGS. To show the impact of Lipschitz estimation on the performance of finite-difference methods, we focus on forward-difference L-BFGS and compare nine different theoretical Lipschitz estimation schemes. The first five consider techniques where L is fixed for the entire run based on information at the initial point. We test this because it is frequently claimed that using an initial estimate of L is sufficient for the entire run; see [40, 71]. The first approach requires no additional information about the function, while the others incorporate information about the Hessian at the current point. The latter four techniques similarly incorporate information about the Hessian but re-estimate L whenever the finite-difference gradient or directional derivative is evaluated.

All of these techniques rely on the assumption that $|D_p^2\phi(x)| \approx |D_p^2\phi(\xi)|$ for some $\xi \in [x, x+h]$. As we will see, both of these approximations that are based on conventional wisdom are challenged in our experiments.

(1) Fix $L = 1$. This requires no additional knowledge about the problem at no added cost.

(2) Fix $L = \max\{10^{-1}, \|\nabla^2\phi(x_0)\|_2\}$. Similar to fixing $L = 1$, but incorporates knowledge of the initial Hessian.

(3) Fix $L = \max\left\{10^{-1}, \frac{1}{n}\sum_{i=1}^{n}|[\nabla^2\phi(x_0)]_{ii}|\right\}$. This can be obtained by choosing h such that the bound on $\|g(x_0) - \nabla\phi(x_0)\|_1$ at the initial point is minimized.

(4) Fix $L = \max\left\{10^{-1}, \frac{1}{\sqrt{n}}\sqrt{\sum_{i=1}^{n}[\nabla^2\phi(x_0)]_{ii}^2}\right\}$. This can be obtained by choosing h such that the bound on $\|g(x_0) - \nabla\phi(x_0)\|_2^2$ at the initial point is minimized.

(5) Fix L to a vector $(\max\{10^{-1}, |[\nabla^2\phi(x_0)]_{ii}|\})_{i=1}^{n}$ and use the ith component for estimating the ith component of $g(x)$. Uses $\|L\|_2/\sqrt{n}$ when estimating h for the directional derivative in the line search.

(6) Evaluate $L = \max\{10^{-1}, \|\nabla^2\phi(x)\|_2\}$ each time finite-differencing is performed.

(7) Evaluate $L = \max\left\{10^{-1}, \frac{1}{n}\sum_{i=1}^{n}|[\nabla^2\phi(x)]_{ii}|\right\}$ each time finite-differencing is performed.

(8) Evaluate $L = \max\left\{10^{-1}, \frac{1}{\sqrt{n}}\sqrt{\sum_{i=1}^{n}[\nabla^2\phi(x)]_{ii}^2}\right\}$ each time finite-differencing is performed.

(9) Evaluate $L = \max\{10^{-1}, |[\nabla^2\phi(x)]_{ii}|\}$ when evaluating $[g(x)]_i$. When the directional derivative along $p \in \mathbb{R}^n$ is computed, evaluates $L = \max\left\{10^{-1}, \frac{|p^T\nabla^2\phi(x)p|}{\|p\|_2^2}\right\}$.

Each method is terminated when it cannot make more progress over 5 consecutive iterations. The optimal value ϕ^* is obtained by running L-BFGS on the non-noisy function until no more progress can be made.

To compare the solution quality between two algorithms, we will use log-ratio profiles as proposed in [67] over the optimality gaps for each algorithm. The log-ratio profiles report

$$\log_2\left(\frac{\phi_{\text{new}} - \phi^*}{\phi_{\text{old}} - \phi^*}\right) \tag{A.1.4}$$

for each problem plotted in increasing order. The area of the shaded region is representative of the general success of the algorithm.

One may ask whether or not using $L = 1$ is sufficient, without any additional knowledge of the second derivative. As seen in Figure A.1, when compared against the other fixed Lipschitz estimation schemes, the difference is not significant. In particular, the additional information from the Hessian does not yield significant benefits over setting $L = 1$ because the bound on the second derivative does not remain valid over the entire run of the algorithm. This may be surprising as it has been commonly suggested that it is sufficient to fix the Lipschitz estimate; see [40, 71]. In fact, we will see that it is more crucial for the algorithm to have a right estimate of L during the later stages of the run than at the beginning of the run in order to achieve high accuracy.

To further support why an adaptive L is important to achieve high accuracy, we compare the fixed L approaches against the adaptive L in Figure A.2.

As seen in Figure A.2, we see that using Lipschitz estimation at every iteration yields much higher accuracy than the alternatives. Upon inspection of individual runs, one can observe many cases where fixing the Lipschitz constant is unstable and inadequate, potentially leading to poor gradient approximations that result in early stagnation of the algorithm. This suggests that it is imperative to adaptively re-estimate L in order to

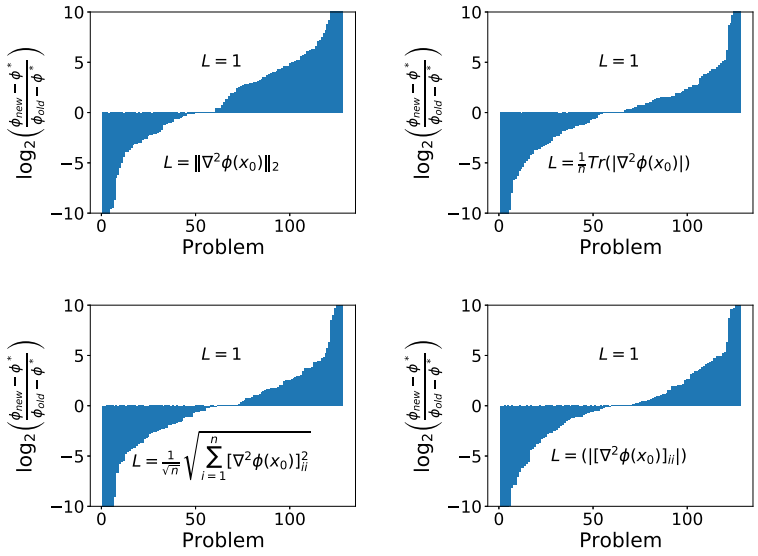

Figure A.1. *Accuracy, Noisy Case with* $\sigma_f = 10^{-3}$. Log-ratio optimality gap profiles comparing forward difference L-BFGS with $L = 1$ and other fixed Lipschitz estimation schemes. The noise level is $\sigma_f = 10^{-3}$, but is representative for $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$.

reduce the noise in the gradient and squeeze the best possible accuracy out of forward difference L-BFGS.

In addition, since estimating L for each direction is most competitive out of the adaptive variants as seen in Figure A.3, we present the results for a heuristic approach in our main work. However, our investigation reveals that the simple mean or root mean square can provide potentially cheaper alternatives if they can be estimated without knowledge of the componentwise Lipschitz constants.

To our knowledge, there are two weaknesses with the componentwise estimation approach. The first is that for a small subset of problems, it is prone to underestimate the Lipschitz constant, as discussed in Section 2.2. The second weakness is that the approach, if performed at each iteration and approximated through finite-differencing, is far too

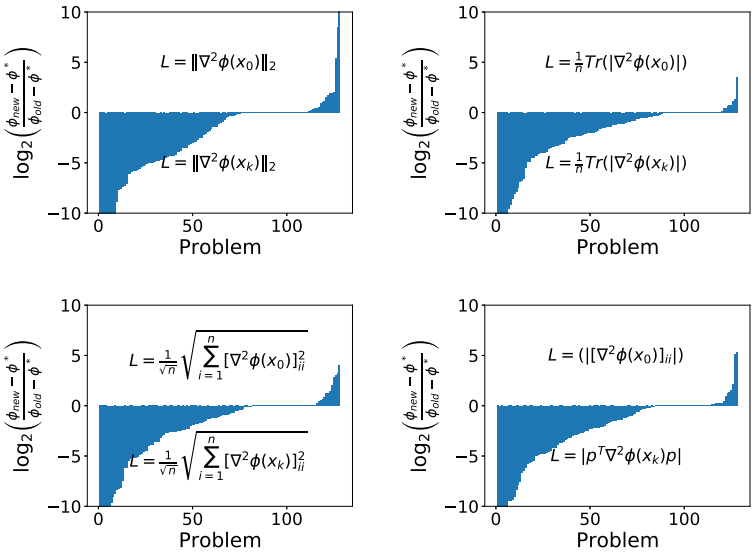

Figure A.2. *Accuracy, Noisy Case with* $\sigma_f = 10^{-3}$. Log-ratio optimality gap profiles comparing forward difference L-BFGS with fixed and adaptive Lipschitz estimation schemes. The noise level is $\sigma_f = 10^{-3}$, but is representative for $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$.

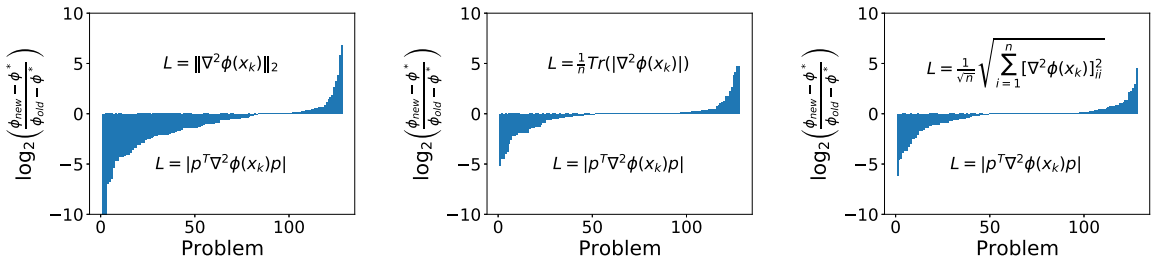

Figure A.3. *Accuracy, Noisy Case with* $\sigma_f = 10^{-3}$. Log-ratio optimality gap profiles comparing forward difference L-BFGS with fixed and adaptive Lipschitz estimation schemes. The noise level is $\sigma_f = 10^{-3}$, but is representative for $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$.

expensive. We propose a few possible practical heuristics for handling this in the following section.

A.1.2. Practical Heuristics for Lipschitz Estimation

Two heuristics were proposed for estimating the Lipschitz constant. Moré and Wild [71] proposed a simple heuristic for estimating the bound on the second derivative of a univariate function with noise, as described in Section 2.2. Gill, et al. [40] proposed a similar procedure that instead enforces that the relative cancellation error lies within an interval

$$\frac{4\epsilon_f}{|\Delta(h)|} \in [0.001, 0.1]. \tag{A.1.5}$$

Note that the lower bound on the interval in (A.1.5) corresponds to (A.1.1).

Both of these heuristics were proposed with particular initial guesses of h and additional conditions to handle certain cases where the methods can fail. These heuristics were employed only at the beginning of the iteration for each variable component, then remains fixed for the entire run.

The only work to our knowledge that employs re-estimation of the Lipschitz constant for finite-differencing derivative-free optimization is Berahas, et al. [7]. In his work, the Lipschitz constant is re-estimated whenever the line search fails and the recovery procedure is triggered. Similarly, we will use the Moré and Wild heuristic to estimate the Lipschitz constant, but only re-estimate the Lipschitz constant when $\alpha < 0.5$ after the first iteration, as described in Procedure I in Section 2.2. We compare two variants of the Lipschitz estimation procedure:

(1) `Component MW`: We use the Moré and Wild heuristic to estimate the Lipschitz constant with respect to each component. This is performed at the first iteration

and subsequent iterations for line search methods when the line search from the prior iteration gives a steplength $\alpha_k < 0.5$. When estimating the directional derivative along the search direction p_k, we use the root mean square of the component-wise estimates.

(2) `Random MW`: We use the Moré and Wild heuristic to estimate the Lipschitz constant along a random direction sampled uniformly from a sphere, i.e., $p \sim S(0, I)$. This is performed at the first iteration and subsequent iterations for line search methods when the line search from the prior iteration gives a steplength $\alpha_k < 0.5$.

We compare both Moré and Wild heuristics against NEWUOA and theoretical componentwise Lipschitz estimation in Figures A.4, A.5, and A.6. In general, `Component MW` gives a better solution than `Random MW`. When we compare these methods against NEWUOA in Figure A.6, the performance of the methods are relatively the same as the theoretical Lipschitz estimates.

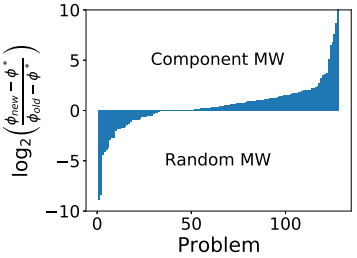

Figure A.4. *Accuracy, Noisy Case with* $\sigma_f = 10^{-3}$. Log-ratio optimality gap profiles comparing forward difference L-BFGS with `component MW` and `random MW` Lipschitz estimation schemes.

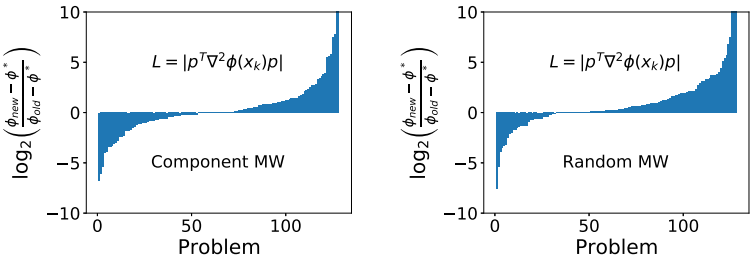

Figure A.5. *Accuracy, Noisy Case with* $\sigma_f = 10^{-3}$. Log-ratio optimality gap profiles comparing forward difference L-BFGS with theoretical componentwise Lipschitz estimates and the Moré and Wild Lipschitz estimation schemes.

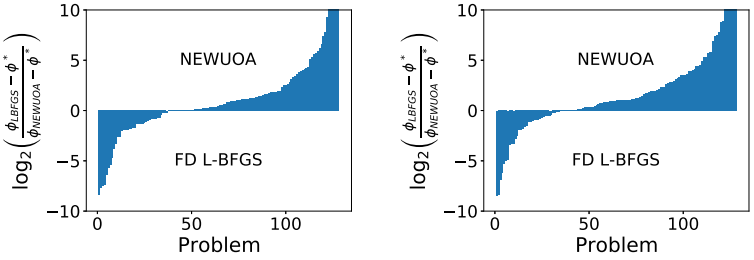

Figure A.6. *Accuracy, Noisy Case with* $\sigma_f = 10^{-3}$. Log-ratio optimality gap profiles comparing NEWUOA against forward difference L-BFGS with the Moré and Wild Lipschitz estimation schemes. We compare L-BFGS with `Component MW` (left) and `Random MW` (right).

A.2. Investigation of Parameters for NEWUOA

In this section, we empirically investigate the influence of the parameters for NEWUOA for the noisy setting. Although these parameters have been optimized for the noiseless setting, it is not generally known how these parameters may impact its performance on noisy unconstrained problems. By default, NEWUOA employs $p = 2n + 1$ interpolation points when constructing the quadratic model at each iteration with an initial trust region radius of $\rho_{\text{beg}} = 1$ and a final trust region radius of $\rho_{\text{end}} = 10^{-6}$.

A.2.1. Number of Interpolation Points

In Section 2.2, we observed that NEWUOA is able to converge to a better quality neighborhood than L-BFGS with forward differencing, but inferior to central differencing. Since NEWUOA by default employs $p = 2n + 1$ points, it is natural to both ask if: (1) decreasing the number of interpolation points would yield a less accurate quadratic model, with a potentially less accurate gradient; and (2) increasing the number of points used in the interpolation may improve the accuracy of the solution to be competitive with central differencing.

To do this, we run NEWUOA with $p = n + 2$ and $p = \min\left\{3n + 1, \frac{(n+1)(n+2)}{2}\right\}$ interpolation points and compare their optimality gaps and number of function evaluations. In Figures A.7 and A.8, we report the log-ratio profiles for the objective function and function evaluations when comparing NEWUOA with $p = n + 2$ and $p = 2n + 1$ points. We see that with higher noise levels, NEWUOA with $p = n + 2$ tends to terminate earlier, while for lower noise levels, it is less efficient than NEWUOA with $p = 2n + 1$ points. In terms of solution quality, NEWUOA with $p = 2n + 1$ is able to converge to a higher accuracy in general than NEWUOA with $p = n + 2$ points. This is similarly seen when we compare $p = 3n + 1$ and $p = 2n + 1$ points in Figures A.9 and A.10.

When we compare NEWUOA with $p = n + 2$ points against forward difference L-BFGS, we see that L-BFGS is now competitive against NEWUOA, unlike when NEWUOA employed $p = 2n + 1$ points; see Figure A.11. However, when we increase the number of points to $p = 3n + 1$, NEWUOA is still not competitive against central differencing, as seen in Figure A.12.

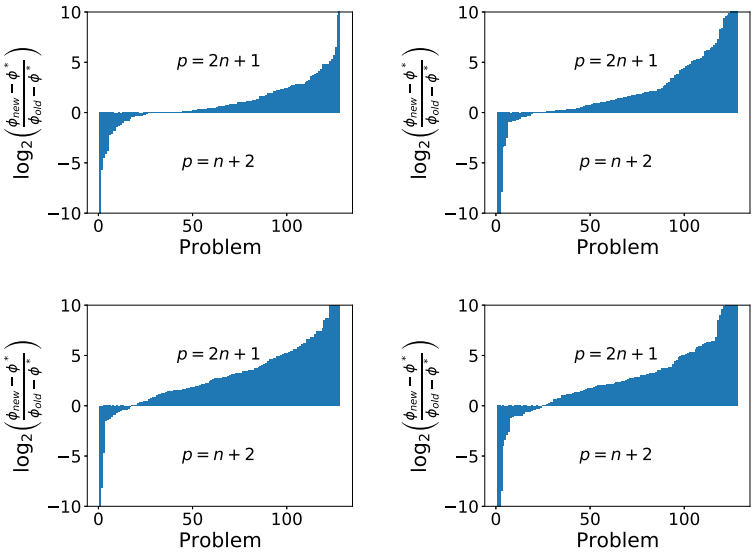

Figure A.7. Noisy Case. Log-ratio optimality gap profiles comparing NEWUOA with $p = 2n + 1$ and $p = n + 2$ points. The noise levels are $\sigma_f = 10^{-1}$ (top left), $\sigma_f = 10^{-3}$ (top right), 10^{-5} (bottom left), and 10^{-7} (bottom right).

A.2.2. Trust Region Radius

One can also ask if decreasing the final trust region radius could allow NEWUOA to converge to a better solution. To test this, we change $\rho_{\text{end}} = 10^{-12}$ and compare against the default $\rho_{\text{end}} = 10^{-6}$ in Figure A.13. From our experiments, changing the final trust region radius makes almost no difference on the solution quality and efficiency.

A.3. Complete Numerical Results

In this appendix, we present our complete numerical results.

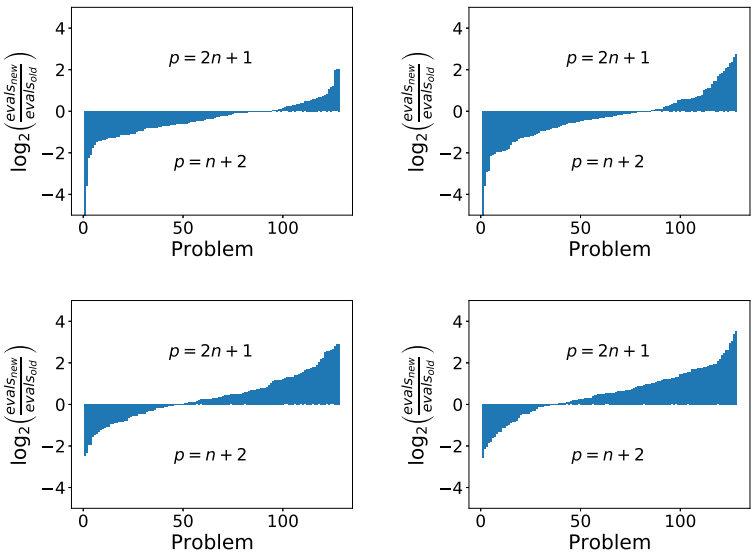

Figure A.8. Noisy Case. Log-ratio function evaluation profiles comparing NEWUOA with $p = 2n + 1$ and $p = n + 2$ points. The noise levels are $\sigma_f = 10^{-1}$ (top left), $\sigma_f = 10^{-3}$ (top right), 10^{-5} (bottom left), and 10^{-7} (bottom right).

A.3.1. Unconstrained Optimization

A.3.1.1. Noiseless Functions. In this section, we list the number of function evaluations (#feval), ratio between number of function evaluations to the number of function evaluations taken by NEWUOA (RATIO), CPU time (CPU), and optimality gap ($\phi(x) - \phi^*$) for each problem instance in Tables A.1-A.5. Function evaluations marked with a * denote cases where the algorithm reached the maximum number of function evaluations. Instances where NEWUOA samples a point that satisfies (2.2.7) within the initial $2n + 1$ evaluations are denoted by **. Problems marked with a ‡ denote cases where NEWUOA and L-BFGS converge to different minimizers. Optimality gaps marked with a † denote failures of the algorithm to converge to a valid solution satisfying (2.2.7).

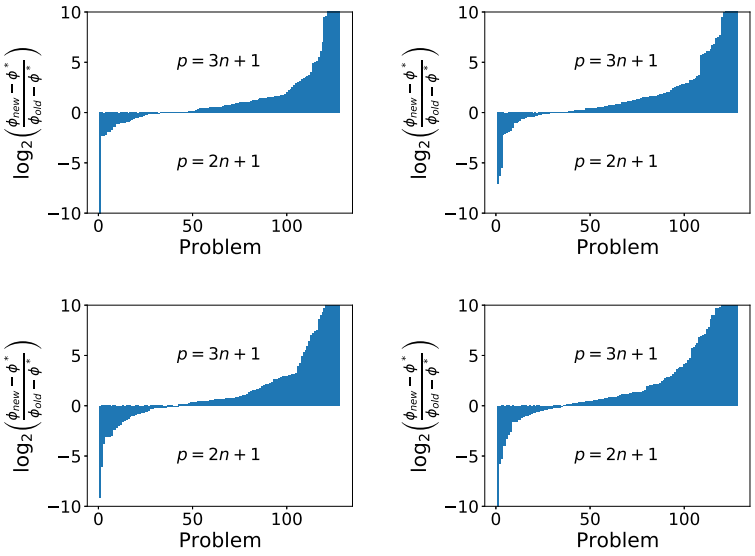

Figure A.9. Noisy Case. Log-ratio optimality gap profiles comparing NEWUOA with $p = 3n + 1$ and $p = 2n + 1$ points. The noise levels are $\sigma_f = 10^{-1}$ (top left), $\sigma_f = 10^{-3}$ (top right), 10^{-5} (bottom left), and 10^{-7} (bottom right).

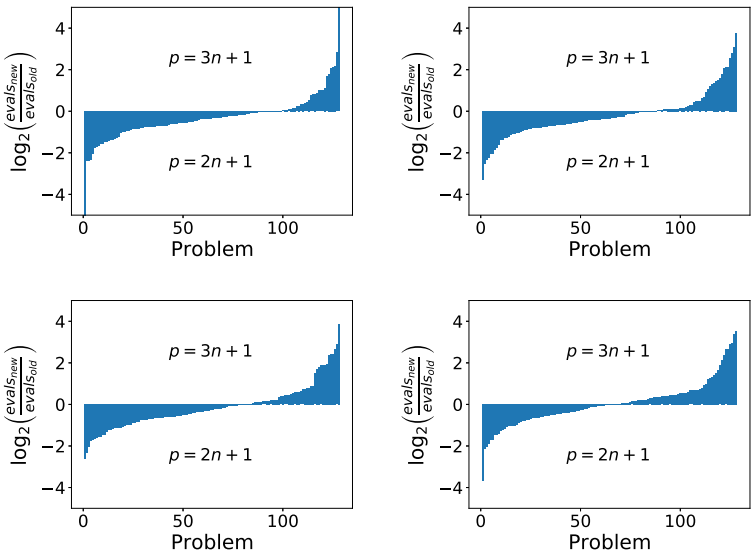

Figure A.10. Noisy Case. Log-ratio function evaluation profiles comparing NEWUOA with $p = 3n + 1$ and $p = 2n + 1$ points. The noise levels are $\sigma_f = 10^{-1}$ (top left), $\sigma_f = 10^{-3}$ (top right), 10^{-5} (bottom left), and 10^{-7} (bottom right).

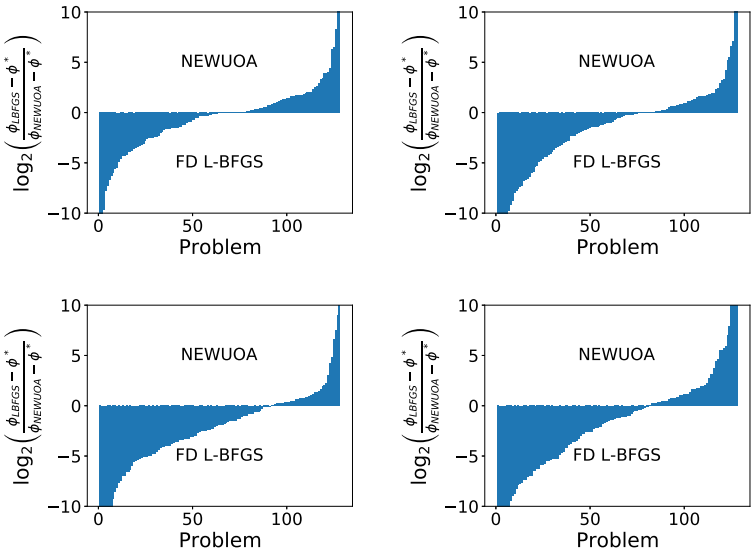

Figure A.11. Noisy Case. Log-ratio optimality gap profiles comparing forward difference L-BFGS against NEWUOA with $p = n + 2$ points. The noise levels are $\sigma_f = 10^{-1}$ (top left), $\sigma_f = 10^{-3}$ (top right), 10^{-5} (bottom left), and 10^{-7} (bottom right).

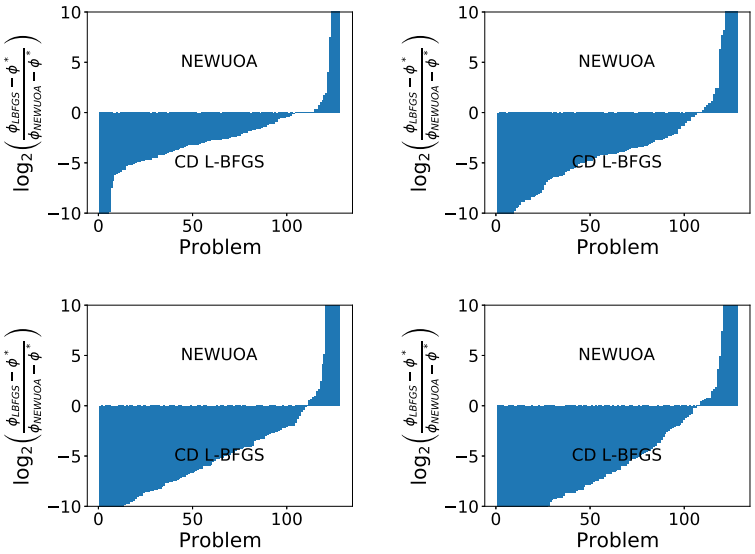

Figure A.12. Noisy Case. Log-ratio optimality gap profiles comparing central difference L-BFGS against NEWUOA with $p = 3n + 1$ points. The noise levels are $\sigma_f = 10^{-1}$ (top left), $\sigma_f = 10^{-3}$ (top right), 10^{-5} (bottom left), and 10^{-7} (bottom right).

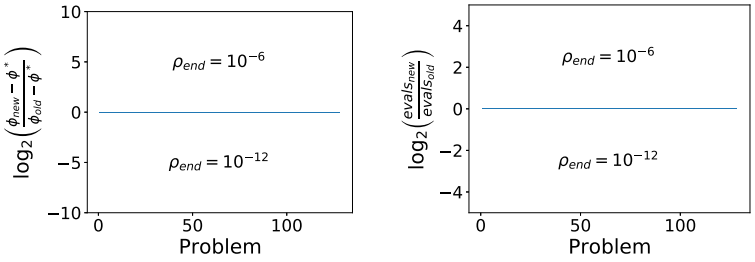

Figure A.13. Noisy Case with $\sigma_f = 10^{-5}$. Log-ratio optimality gap profiles comparing NEWUOA with $\rho_{\text{end}} = 10^{-6}$ and 10^{-12}.

		NEWUOA				FD L-BFGS				CD L-BFGS			
Problem	n	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$
AIRCRFTB	5	661	1.000	0.1	6.963×10^{-7}	296	4.48×10^{-1}	0.02	6.518×10^{-7}	536	8.11×10^{-1}	0.03	8.288×10^{-7}
ALLINITU	4	41	1.000	0.09	3.073×10^{-6}	58	1.415	0.01	4.240×10^{-6}	103	2.512	0.01	4.240×10^{-6}
ARWHEAD	100	201**	1.000	0.09	0.000	1130	5.622	0.06	2.474×10^{-7}	2240	1.1144×10^{1}	0.07	2.476×10^{-7}
BARD	3	69	1.000	0.09	7.032×10^{-8}	102	1.478	0.01	5.244×10^{-7}	176	2.551	0.01	5.246×10^{-7}
BDQRTIC	100	3682	1.000	1.85	3.731×10^{-4}	3178	8.63×10^{-1}	0.14	2.923×10^{-4}	6308	1.713	0.18	2.914×10^{-4}
BIGGS3	3	73	1.000	0.09	4.599×10^{-7}	85	1.164	0.01	5.422×10^{-7}	152	2.082	0.01	5.423×10^{-7}
BIGGS5‡	5	103	1.000	0.08	-3.722×10^{-5}	416	4.039	0.04	6.651×10^{-7}	767	7.447	0.05	4.321×10^{-7}
BIGGS6‡	6	338	1.000	0.08	-1.444×10^{-4}	323	9.56×10^{-1}	0.03	5.357×10^{-7}	593	1.754	0.03	5.654×10^{-7}
BOX2	2	2**	1.000	0.08	5.847×10^{-19}	57	2.8500×10^{1}	0.01	9.947×10^{-7}	95	4.7500×10^{1}	0.01	9.947×10^{-7}
BOX3	3	2**	1.000	0.08	5.847×10^{-19}	52	2.6000×10^{1}	0.01	7.371×10^{-7}	91	4.5500×10^{1}	0.01	7.371×10^{-7}
BRKMCC	2	10	1.000	0.08	1.440×10^{-7}	32	3.200	0.0	4.066×10^{-9}	51	5.100	0.0	4.088×10^{-9}
BROWNAL	10	917	1.000	0.11	9.998×10^{-7}	177	1.93×10^{-1}	0.01	1.707×10^{-7}	331	3.61×10^{-1}	0.01	1.707×10^{-7}
BROWNAL	100	18069	1.000	9.53	9.993×10^{-7}	1039	5.8×10^{-2}	0.07	1.040×10^{-7}	2051	1.14×10^{-1}	0.1	1.047×10^{-7}
BROWNAL	200	100000*	1.000	207.31	3.184×10^{-6}	823	8×10^{-3}	0.13	8.254×10^{-7}	1626	1.6×10^{-2}	0.16	8.243×10^{-7}
BROWNDEN	4	140	1.000	0.09	2.900×10^{-2}	131	9.36×10^{-1}	0.01	1.448×10^{-2}	225	1.607	0.01	1.449×10^{-2}
CLIFF	2	84	1.000	0.09	5.220×10^{-7}	240	2.857	0.02	3.101×10^{-7}	365	4.345	0.02	2.076×10^{-7}
CRAGGLVY	4	182	1.000	0.09	8.761×10^{-7}	135	7.42×10^{-1}	0.01	5.854×10^{-7}	244	1.341	0.01	5.854×10^{-7}
CRAGGLVY	10	511	1.000	0.1	1.650×10^{-6}	482	9.43×10^{-1}	0.03	1.176×10^{-6}	910	1.781	0.04	1.168×10^{-6}
CRAGGLVY	50	2670	1.000	0.45	1.523×10^{-5}	2301	8.62×10^{-1}	0.1	1.289×10^{-5}	4544	1.702	0.14	1.292×10^{-5}
CRAGGLVY	100	5679	1.000	2.92	3.219×10^{-5}	4503	7.93×10^{-1}	0.21	1.932×10^{-5}	8946	1.575	0.29	1.934×10^{-5}
CUBE	2	173	1.000	0.08	8.157×10^{-7}	114	6.59×10^{-1}	0.01	3.107×10^{-7}	190	1.098	0.01	3.036×10^{-7}
DENSCHNA	2	22	1.000	0.09	2.893×10^{-10}	38	1.727	0.01	9.517×10^{-10}	64	2.909	0.01	9.521×10^{-10}
DENSCHNB	2	21	1.000	0.09	9.135×10^{-8}	25	1.190	0.0	3.867×10^{-11}	42	2.000	0.0	3.877×10^{-11}
DENSCHNC	2	66	1.000	0.09	7.333×10^{-8}	68	1.030	0.01	8.949×10^{-8}	111	1.682	0.01	8.951×10^{-8}
DENSCHND	3	312	1.000	0.07	9.794×10^{-7}	208	6.67×10^{-1}	0.02	9.534×10^{-7}	356	1.141	0.02	9.537×10^{-7}
DENSCHNE	3	127	1.000	0.09	4.487×10^{-7}	141	1.110	0.01	1.599×10^{-7}	237	1.866	0.01	1.210×10^{-7}
DENSCHNF	2	28	1.000	0.09	5.079×10^{-9}	48	1.714	0.01	5.450×10^{-9}	77	2.750	0.01	5.468×10^{-9}

Table A.1. Noiseless Unconstrained `CUTEst` Problems Tested. n is the number of variables.

		NEWUOA				FD L-BFGS				CD L-BFGS			
Problem	n	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$
DIXMAANA	15	596	1.000	0.09	9.909×10^{-7}	138	2.32×10^{-1}	0.01	1.096×10^{-7}	265	4.45×10^{-1}	0.01	1.096×10^{-7}
DIXMAANA	90	1594	1.000	0.61	9.626×10^{-7}	738	4.63×10^{-1}	0.03	6.577×10^{-7}	1465	9.19×10^{-1}	0.04	6.578×10^{-7}
DIXMAANA	300	8117	1.000	37.53	9.936×10^{-7}	2720	3.35×10^{-1}	0.15	2.979×10^{-9}	5428	6.69×10^{-1}	0.21	2.978×10^{-9}
DIXMAANB	15	395	1.000	0.1	8.586×10^{-7}	122	3.09×10^{-1}	0.01	3.845×10^{-9}	233	5.90×10^{-1}	0.01	3.845×10^{-9}
DIXMAANB	90	1491	1.000	0.56	9.724×10^{-7}	647	4.34×10^{-1}	0.03	6.536×10^{-10}	1283	8.60×10^{-1}	0.04	6.537×10^{-10}
DIXMAANB	300	6446	1.000	27.48	9.980×10^{-7}	2117	3.28×10^{-1}	0.15	7.825×10^{-10}	4223	6.55×10^{-1}	0.15	7.845×10^{-10}
DIXMAANC	15	197	1.000	0.08	9.747×10^{-7}	158	8.02×10^{-1}	0.01	8.216×10^{-7}	302	1.533	0.01	8.216×10^{-7}
DIXMAANC	90	1611	1.000	0.63	9.914×10^{-7}	558	3.46×10^{-1}	0.03	1.352×10^{-7}	1104	6.85×10^{-1}	0.03	1.353×10^{-7}
DIXMAANC	300	4643	1.000	18.87	9.964×10^{-7}	2120	4.57×10^{-1}	0.15	5.674×10^{-7}	4227	9.10×10^{-1}	0.15	5.672×10^{-7}
DIXMAAND	15	251	1.000	0.1	9.036×10^{-7}	179	7.13×10^{-1}	0.01	4.536×10^{-7}	340	1.355	0.01	4.536×10^{-7}
DIXMAAND	90	2090	1.000	0.81	9.552×10^{-7}	745	3.56×10^{-1}	0.04	6.242×10^{-7}	1474	7.05×10^{-1}	0.04	6.242×10^{-7}
DIXMAAND	300	7155	1.000	33.18	9.791×10^{-7}	2425	3.39×10^{-1}	0.18	3.136×10^{-7}	4834	6.76×10^{-1}	0.18	3.136×10^{-7}
DIXMAANE	15	377	1.000	0.1	7.225×10^{-7}	293	7.77×10^{-1}	0.02	4.958×10^{-7}	564	1.496	0.02	4.959×10^{-7}
DIXMAANE	90	3081	1.000	1.28	9.840×10^{-7}	3499	1.136	0.15	8.197×10^{-7}	6956	2.258	0.2	8.202×10^{-7}
DIXMAANE	300	18666	1.000	107.25	9.993×10^{-7}	19633	1.052	0.96	5.643×10^{-7}	39197	2.100	1.42	6.031×10^{-7}
DIXMAANF	15	274	1.000	0.09	9.301×10^{-7}	226	8.25×10^{-1}	0.01	7.285×10^{-7}	434	1.584	0.01	7.286×10^{-7}
DIXMAANF	90	3104	1.000	1.33	9.964×10^{-7}	2674	8.61×10^{-1}	0.11	3.730×10^{-7}	5313	1.712	0.15	3.731×10^{-7}
DIXMAANF	300	16182	1.000	92.5	9.988×10^{-7}	14504	8.96×10^{-1}	0.69	9.006×10^{-7}	28952	1.789	1.06	9.011×10^{-7}
DIXMAANG	15	286	1.000	0.08	8.435×10^{-7}	277	9.69×10^{-1}	0.01	2.101×10^{-7}	533	1.864	0.02	2.102×10^{-7}
DIXMAANG	90	3429	1.000	1.47	9.922×10^{-7}	2857	8.33×10^{-1}	0.12	6.520×10^{-7}	5678	1.656	0.16	6.522×10^{-7}
DIXMAANG	300	21081	1.000	126.91	9.999×10^{-7}	14806	7.02×10^{-1}	0.78	9.894×10^{-7}	30158	1.431	1.1	8.871×10^{-7}
DIXMAANH	15	375	1.000	0.1	8.025×10^{-7}	280	7.47×10^{-1}	0.02	1.341×10^{-7}	537	1.432	0.02	1.342×10^{-7}
DIXMAANH	90	3106	1.000	1.27	9.943×10^{-7}	2588	8.33×10^{-1}	0.11	6.407×10^{-7}	5138	1.654	0.15	6.405×10^{-7}
DIXMAANH	300	35784	1.000	212.15	9.999×10^{-7}	14207	3.97×10^{-1}	0.69	7.042×10^{-7}	28356	7.92×10^{-1}	1.05	6.752×10^{-7}
DIXMAANI	15	586	1.000	0.1	9.553×10^{-7}	549	9.37×10^{-1}	0.03	9.880×10^{-7}	1060	1.809	0.04	9.880×10^{-7}
DIXMAANI	90	13092	1.000	6.05	9.966×10^{-7}	15925	1.216	0.62	9.862×10^{-7}	33132	2.531	0.91	9.270×10^{-7}
DIXMAANI	300	121834	1.000	774.33	9.999×10^{-7}	150107*	1.232	6.82	2.119×10^{-6}	150155*	1.232	5.3	1.720×10^{-5}
DIXMAANJ	15	563	1.000	0.1	9.736×10^{-7}	479	8.51×10^{-1}	0.03	1.809×10^{-7}	926	1.645	0.03	1.797×10^{-7}
DIXMAANJ	90	12417	1.000	6.11	9.923×10^{-7}	11508	9.27×10^{-1}	0.53	9.628×10^{-7}	22883	1.843	0.74	9.706×10^{-7}
DIXMAANJ	300	120362	1.000	718.6	9.987×10^{-7}	77321	6.42×10^{-1}	3.61	9.974×10^{-7}	150156*	1.248	5.37	1.028×10^{-6}
DIXMAANK	15	559	1.000	0.1	9.181×10^{-7}	533	9.53×10^{-1}	0.03	3.242×10^{-7}	1029	1.841	0.04	3.230×10^{-7}
DIXMAANK	90	12601	1.000	5.72	9.969×10^{-7}	7463	5.92×10^{-1}	0.3	9.902×10^{-7}	15018	1.192	0.41	9.893×10^{-7}
DIXMAANK	300	80808	1.000	455.59	9.997×10^{-7}	23567	2.92×10^{-1}	1.12	9.623×10^{-7}	47649	5.90×10^{-1}	1.74	9.108×10^{-7}
DIXMAANL	15	752	1.000	0.1	9.740×10^{-7}	502	6.68×10^{-1}	0.03	4.740×10^{-7}	967	1.286	0.03	4.714×10^{-7}
DIXMAANL	90	10608	1.000	4.87	9.970×10^{-7}	10323	9.73×10^{-1}	0.41	9.519×10^{-7}	20517	1.934	0.57	8.891×10^{-7}
DIXMAANL	300	150000*	1.000	875.24	3.192×10^{-6}	34747	2.32×10^{-1}	1.67	9.946×10^{-7}	69364	4.62×10^{-1}	2.51	9.926×10^{-7}

Table A.2. Noiseless Unconstrained `CUTEst` Problems Tested. n is the number of variables.

		NEWUOA				FD L-BFGS				CD L-BFGS			
Problem	n	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$
DQRTIC	10	502	1.000	0.1	9.719×10^{-7}	258	5.14×10^{-1}	0.02	4.439×10^{-7}	488	9.72×10^{-1}	0.02	4.439×10^{-7}
DQRTIC	50	2847	1.000	0.43	8.852×10^{-7}	1467	5.15×10^{-1}	0.06	7.890×10^{-7}	2894	1.017	0.08	7.888×10^{-7}
DQRTIC	100	6061	1.000	2.81	9.769×10^{-7}	3277	5.41×10^{-1}	0.13	5.471×10^{-7}	6508	1.074	0.17	5.462×10^{-7}
EDENSCH	36	928	1.000	0.14	2.175×10^{-4}	577	6.22×10^{-1}	0.03	7.341×10^{-5}	1131	1.219	0.03	7.341×10^{-5}
EIGENALS	6	5**	1.000	0.08	0.000	83	1.6600×10^{1}	0.01	7.344×10^{-9}	152	3.0400×10^{1}	0.01	7.099×10^{-9}
EIGENALS	110	55000*	1.000	40.44	1.716×10^{-3}	42138	7.66×10^{-1}	2.12	9.758×10^{-7}	55108*	1.002	2.13	2.142×10^{-5}
EIGENBLS‡	6	113	1.000	0.09	-3.033×10^{-4}	91	8.05×10^{-1}	0.01	9.727×10^{-9}	167	1.478	0.01	9.145×10^{-9}
EIGENBLS	110	37505	1.000	27.4	9.993×10^{-7}	55010*	1.467	2.68	9.280×10^{-4}	55088*	1.469	2.1	4.002×10^{-2}
EIGENCLS	30	1630	1.000	0.19	9.383×10^{-7}	2634	1.616	0.12	9.353×10^{-7}	5174	3.174	0.16	8.934×10^{-7}
ENGVAL1	2	32	1.000	0.08	6.651×10^{-9}	38	1.188	0.0	2.191×10^{-7}	60	1.875	0.0	2.191×10^{-7}
ENGVAL1	50	2234	1.000	0.38	5.211×10^{-5}	585	2.62×10^{-1}	0.02	3.074×10^{-5}	1148	5.14×10^{-1}	0.03	3.074×10^{-5}
ENGVAL1	100	1941	1.000	0.92	1.076×10^{-4}	1337	6.89×10^{-1}	0.07	4.197×10^{-6}	2651	1.366	0.07	4.198×10^{-6}
EXPFIT	2	42	1.000	0.09	9.588×10^{-7}	46	1.095	0.01	1.427×10^{-9}	75	1.786	0.0	1.420×10^{-9}
FLETCBV3‡	10	1106	1.000	0.11	1.063×10^{-3}	261	2.36×10^{-1}	0.01	-1.664×10^{-5}	521	4.71×10^{-1}	0.02	-4.710×10^{-5}
FLETCBV3‡	100	50000*	1.000	28.66	1.333×10^{5}	19708	3.94×10^{-1}	0.87	-3.486×10^{-1}	16125	3.23×10^{-1}	0.54	-1.107×10^{1}
FLETCHBV‡	10	945	1.000	0.09	1.490×10^{5}	263	2.78×10^{-1}	0.02	-1.278×10^{1}	278	2.94×10^{-1}	0.01	-5.322×10^{3}
FLETCHBV‡	100	50000*	1.000	27.77	1.210×10^{13}	18068	3.61×10^{-1}	0.8	-6.240×10^{6}	20933	4.19×10^{-1}	0.67	-7.016×10^{7}
FREUROTH	2	55	1.000	0.09	2.214×10^{-5}	81	1.473	0.01	3.932×10^{-5}	134	2.436	0.01	3.932×10^{-5}
FREUROTH	10	367	1.000	0.09	6.524×10^{-4}	286	7.79×10^{-1}	0.02	1.194×10^{-6}	538	1.466	0.02	1.194×10^{-6}
FREUROTH	50	5840	1.000	0.86	5.844×10^{-3}	1053	1.80×10^{-1}	0.04	4.888×10^{-4}	2073	3.55×10^{-1}	0.06	4.886×10^{-4}
FREUROTH	100	2881	1.000	1.44	1.188×10^{-2}	1950	6.77×10^{-1}	0.09	2.601×10^{-3}	3868	1.343	0.12	2.589×10^{-3}
GENROSE	5	199	1.000	0.08	9.086×10^{-7}	226	1.136	0.02	9.527×10^{-9}	411	2.065	0.02	9.367×10^{-9}
GENROSE	10	645	1.000	0.1	8.369×10^{-7}	896	1.389	0.06	8.050×10^{-7}	1614	2.502	0.06	1.032×10^{-7}
GENROSE	100	17249	1.000	9.45	9.954×10^{-7}	25453	1.476	1.01	8.013×10^{-7}	50198*	2.910	1.4	1.939×10^{-5}
GULF	3	876	1.000	0.12	6.954×10^{-7}	225	2.57×10^{-1}	0.03	7.913×10^{-8}	392	4.47×10^{-1}	0.04	6.048×10^{-7}
HAIRY	2	261	1.000	0.09	1.214×10^{-6}	86	3.30×10^{-1}	0.01	2.209×10^{-7}	149	5.71×10^{-1}	0.01	2.996×10^{-7}
HELIX	3	65	1.000	0.09	3.569×10^{-8}	149	2.292	0.01	6.812×10^{-7}	86	1.323	0.0	$8.521 \times 10^{2\dagger}$
JENSMP‡	2	4**	1.000	0.09	-5.469×10^{2}	16	4.000	0.0	0.000	21	5.250	0.0	0.000
KOWOSB	4	150	1.000	0.09	7.089×10^{-7}	183	1.220	0.02	2.824×10^{-7}	332	2.213	0.02	2.825×10^{-7}
MEXHAT	2	46	1.000	0.09	$1.036 \times 10^{-2\dagger}$	203	4.413	0.02	7.395×10^{-7}	339	7.370	0.02	4.707×10^{-7}

Table A.3. Noiseless Unconstrained `CUTEst` Problems Tested. n is the number of variables.

		NEWUOA				FD L-BFGS				CD L-BFGS			
Problem	n	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$	#feval	RATIO	CPU	$\phi(x) - \phi^*$
MOREBV	10	379	1.000	0.09	9.458×10^{-7}	366	9.66×10^{-1}	0.02	6.782×10^{-7}	695	1.834	0.03	6.843×10^{-7}
MOREBV	50	21488	1.000	3.71	9.988×10^{-7}	25029*	1.165	0.98	1.368×10^{-6}	25035*	1.165	0.68	6.088×10^{-6}
MOREBV	100	11143	1.000	6.29	9.999×10^{-7}	15512	1.392	0.59	9.985×10^{-7}	31067	2.788	0.82	9.992×10^{-7}
NCB20B	21	478	1.000	0.11	4.093×10^{-5}	169	3.54×10^{-1}	0.01	-4.368×10^{-11}	322	6.74×10^{-1}	0.01	-4.334×10^{-11}
NCB20B	22	627	1.000	0.11	4.361×10^{-5}	154	2.46×10^{-1}	0.01	2.493×10^{-5}	292	4.66×10^{-1}	0.01	2.493×10^{-5}
NCB20B	50	3345	1.000	0.59	9.936×10^{-5}	1465	4.38×10^{-1}	0.07	9.906×10^{-5}	2892	8.65×10^{-1}	0.1	9.912×10^{-5}
NCB20B	100	7309	1.000	4.02	1.963×10^{-4}	3376	4.62×10^{-1}	0.21	1.794×10^{-4}	6708	9.18×10^{-1}	0.3	1.797×10^{-4}
NCB20B	180	11313	1.000	22.07	3.470×10^{-4}	4375	3.87×10^{-1}	0.35	3.384×10^{-4}	8718	7.71×10^{-1}	0.56	3.379×10^{-4}
NONDIA	10	198	1.000	0.1	4.267×10^{-7}	167	8.43×10^{-1}	0.01	1.459×10^{-11}	298	1.505	0.01	1.053×10^{-11}
NONDIA	20	426	1.000	0.1	8.903×10^{-7}	341	8.00×10^{-1}	0.02	2.571×10^{-10}	655	1.538	0.02	2.929×10^{-10}
NONDIA	30	569	1.000	0.12	9.936×10^{-7}	492	8.65×10^{-1}	0.02	3.712×10^{-7}	956	1.680	0.03	3.738×10^{-7}
NONDIA	50	799	1.000	0.2	9.829×10^{-7}	792	9.91×10^{-1}	0.03	3.825×10^{-8}	1556	1.947	0.04	3.691×10^{-8}
NONDIA	90	1302	1.000	0.58	9.192×10^{-7}	1301	9.99×10^{-1}	0.06	1.233×10^{-7}	2574	1.977	0.07	1.188×10^{-7}
NONDIA	100	1683	1.000	0.89	9.873×10^{-7}	1543	9.17×10^{-1}	0.07	1.028×10^{-10}	3057	1.816	0.09	1.125×10^{-10}
NONDQUAR	100	50000*	1.000	29.28	8.675×10^{-6}	50056*	1.001	1.88	1.137×10^{-5}	50189*	1.004	1.32	3.611×10^{-5}
OSBORNEA‡	5	1094	1.000	0.09	1.204×10^{-5}	477	4.36×10^{-1}	0.05	8.442×10^{-7}	888	8.12×10^{-1}	0.06	8.390×10^{-7}
OSBORNEB	11	1529	1.000	0.14	9.678×10^{-7}	1044	6.83×10^{-1}	0.08	9.551×10^{-7}	2216	1.449	0.11	6.312×10^{-7}
PENALTY1	4	577	1.000	0.09	9.994×10^{-7}	868	1.504	0.09	9.134×10^{-7}	1425	2.470	0.09	9.856×10^{-7}
PENALTY1	10	1332	1.000	0.11	9.944×10^{-7}	1523	1.143	0.09	9.903×10^{-7}	2942	2.209	0.11	8.873×10^{-7}
PENALTY1	50	6082	1.000	0.89	9.985×10^{-7}	5773	9.49×10^{-1}	0.22	9.994×10^{-7}	11797	1.940	0.32	9.652×10^{-7}
PENALTY1	100	13215	1.000	6.44	9.969×10^{-7}	11175	8.46×10^{-1}	0.43	7.672×10^{-7}	21381	1.618	0.57	9.919×10^{-7}
PFIT1LS‡	3	417	1.000	0.11	2.892×10^{-4}	351	8.42×10^{-1}	0.03	9.871×10^{-7}	403	9.66×10^{-1}	0.03	6.795×10^{-7}
PFIT2LS‡	3	767	1.000	0.1	1.243×10^{-2}	1502*	1.958	0.17	2.818×10^{-4}	1509*	1.967	0.11	1.528×10^{-3}
PFIT3LS‡	3	907	1.000	0.1	8.228×10^{-2}	1502*	1.656	0.17	1.504×10^{-2}	1501*	1.655	0.11	3.622×10^{-2}
PFIT4LS‡	3	1088	1.000	0.1	2.673×10^{-1}	1505*	1.383	0.15	5.424×10^{-2}	1508*	1.386	0.1	9.180×10^{-2}
QUARTC	25	1004	1.000	0.13	8.886×10^{-7}	765	7.62×10^{-1}	0.04	4.947×10^{-7}	1492	1.486	0.05	4.946×10^{-7}
QUARTC	100	6061	1.000	2.82	9.769×10^{-7}	3277	5.41×10^{-1}	0.13	5.471×10^{-7}	6508	1.074	0.16	5.462×10^{-7}
SINEVAL	2	254	1.000	0.08	6.706×10^{-9}	309	1.217	0.04	3.585×10^{-9}	522	2.055	0.04	1.305×10^{-7}

Table A.4. Noiseless Unconstrained `CUTEst` Problems Tested. n is the number of variables.

		NEWUOA				FD L-BFGS				CD L-BFGS			
Problem	n	#feval	RATIO	CPU	$\phi(x)-\phi^*$	#feval	RATIO	CPU	$\phi(x)-\phi^*$	#feval	RATIO	CPU	$\phi(x)-\phi^*$
SINQUAD	5	125	1.000	0.09	6.682×10^{-6}	61	4.88×10^{-1}	0.01	3.061×10^{-6}	108	8.64×10^{-1}	0.01	3.061×10^{-6}
SINQUAD	50	3325	1.000	0.55	1.132×10^{-3}	795	2.39×10^{-1}	0.04	4.489×10^{-6}	1562	4.70×10^{-1}	0.05	4.553×10^{-6}
SINQUAD	100	8318	1.000	4.3	3.985×10^{-3}	1545	1.86×10^{-1}	0.07	2.755×10^{-3}	3062	3.68×10^{-1}	0.09	2.758×10^{-3}
SISSER	2	24	1.000	0.09	8.489×10^{-7}	54	2.250	0.01	5.876×10^{-7}	92	3.833	0.01	5.876×10^{-7}
SPARSQUR	10	184	1.000	0.09	8.876×10^{-7}	184	1.000	0.01	6.336×10^{-7}	348	1.891	0.01	6.336×10^{-7}
SPARSQUR	50	1645	1.000	0.29	6.775×10^{-7}	1046	6.36×10^{-1}	0.04	3.744×10^{-7}	2065	1.255	0.06	3.744×10^{-7}
SPARSQUR	100	2932	1.000	1.37	8.999×10^{-7}	2150	7.33×10^{-1}	0.09	3.880×10^{-7}	4270	1.456	0.12	3.880×10^{-7}
TOINTGSS	10	234	1.000	0.09	1.038×10^{-5}	24	1.03×10^{-1}	0.0	0.000	45	1.92×10^{-1}	0.0	0.000
TOINTGSS	50	775	1.000	0.18	9.723×10^{-6}	104	1.34×10^{-1}	0.0	-3.000×10^{-9}	205	2.65×10^{-1}	0.01	-3.000×10^{-9}
TOINTGSS	100	1199	1.000	0.55	9.778×10^{-6}	204	1.70×10^{-1}	0.02	4.000×10^{-9}	405	3.38×10^{-1}	0.01	4.000×10^{-9}
TQUARTIC	5	92	1.000	0.09	5.444×10^{-7}	85	9.24×10^{-1}	0.01	1.078×10^{-8}	150	1.630	0.01	1.074×10^{-8}
TQUARTIC	10	277	1.000	0.09	9.847×10^{-7}	172	6.21×10^{-1}	0.01	5.228×10^{-9}	325	1.173	0.01	5.313×10^{-9}
TQUARTIC	50	5022	1.000	0.75	9.949×10^{-7}	646	1.29×10^{-1}	0.03	1.681×10^{-8}	1265	2.52×10^{-1}	0.03	1.725×10^{-8}
TQUARTIC	100	20515	1.000	10.51	9.991×10^{-7}	1538	7.5×10^{-2}	0.07	6.818×10^{-9}	3053	1.49×10^{-1}	0.08	6.223×10^{-9}
TRIDIA	10	186	1.000	0.09	9.019×10^{-7}	210	1.129	0.01	5.451×10^{-8}	396	2.129	0.01	5.446×10^{-8}
TRIDIA	20	446	1.000	0.1	8.062×10^{-7}	690	1.547	0.04	8.068×10^{-7}	1340	3.004	0.04	8.064×10^{-7}
TRIDIA	30	768	1.000	0.13	9.962×10^{-7}	1481	1.928	0.07	6.309×10^{-7}	2906	3.784	0.08	6.315×10^{-7}
TRIDIA	50	1446	1.000	0.3	8.806×10^{-7}	3128	2.163	0.15	9.300×10^{-7}	6187	4.279	0.19	9.292×10^{-7}
TRIDIA	100	3450	1.000	2.11	9.968×10^{-7}	8783	2.546	0.32	9.677×10^{-7}	17468	5.063	0.43	9.208×10^{-7}
WATSON	12	4966	1.000	0.21	9.982×10^{-7}	1324	2.67×10^{-1}	0.08	9.773×10^{-7}	2708	5.45×10^{-1}	0.11	9.950×10^{-7}
WATSON	31	15500*	1.000	1.11	6.438×10^{-5}	13875	8.95×10^{-1}	0.65	9.957×10^{-7}	15530*	1.002	0.51	5.664×10^{-6}
WOODS	4	497	1.000	0.08	1.434×10^{-7}	178	3.58×10^{-1}	0.02	4.447×10^{-9}	312	6.28×10^{-1}	0.02	4.353×10^{-9}
WOODS	100	50000*	1.000	28.69	1.371×10^{-4}	2972	5.9×10^{-2}	0.13	7.208×10^{-7}	5902	1.18×10^{-1}	0.16	6.252×10^{-7}
ZANGWIL2	2	12	1.000	0.09	5.085×10^{-7}	11	9.17×10^{-1}	0.0	-1.000×10^{-10}	19	1.583	0.0	-9.999×10^{-11}

Table A.5. Noiseless Unconstrained `CUTEst` Problems Tested. n is the number of variables.

A.3.1.2. Noisy Functions. In this section, we list the best optimality gap $(\phi(x) - \phi^*)$ and the number of function evaluations (#feval) needed to achieve this for each problem instance, varying the noise level $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$, in Tables A.6-A.18. Function evaluations marked with a * denote cases where the algorithm reached the maximum number of function evaluations. Instances where NEWUOA samples a point that satisfies (2.2.7) within the initial $2d + 1$ evaluations are denoted by **. Problems marked with a ‡ denote cases where NEWUOA and L-BFGS converge to different minimizers.

A.3.2. Nonlinear Least Squares Problems

A.3.2.1. Noiseless Functions. In this section, we list the number of function evaluations (#feval), CPU time (CPU), and optimality gap $(\phi(x) - \phi^*)$ for each problem instance in Tables A.19. Function evaluations marked with a * denote cases where the algorithm reached the maximum number of function evaluations.

A.3.2.2. Noisy Functions. In this section, we list the best optimality gap $(\phi(x) - \phi^*)$ and the number of function evaluations (#feval) needed to achieve this for each problem instance, varying the noise level $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$, in Tables A.21-A.24. Function evaluations marked with a * denote cases where the algorithm reached the maximum number of function evaluations.

A.3.3. Constrained Optimization

A.3.3.1. Test Problem Summary.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
AIRCRFTB	5	1×10^{-1}	43	7.257×10^{-1}	53	7.277×10^{-1}	231	2.623×10^{-1}
AIRCRFTB	5	1×10^{-3}	70	1.772×10^{-1}	715	9.938×10^{-2}	904	6.648×10^{-5}
AIRCRFTB	5	1×10^{-5}	159	1.006×10^{-2}	1221	3.355×10^{-4}	927	1.042×10^{-5}
AIRCRFTB	5	1×10^{-7}	317	2.893×10^{-4}	1176	4.119×10^{-6}	1106	7.139×10^{-12}
ALLINITU	4	1×10^{-1}	26	1.020×10^{-1}	24	2.881	108	1.980×10^{-3}
ALLINITU	4	1×10^{-3}	38	9.901×10^{-4}	97	2.614×10^{-4}	189	7.467×10^{-7}
ALLINITU	4	1×10^{-5}	44	1.337×10^{-6}	62	1.721×10^{-5}	320	5.606×10^{-9}
ALLINITU	4	1×10^{-7}	58	1.186×10^{-8}	74	1.612×10^{-7}	170	3.518×10^{-10}
ARWHEAD	100	1×10^{-1}	201**	0.000	508	8.433×10^{-2}	1022	5.894×10^{-2}
ARWHEAD	100	1×10^{-3}	201**	0.000	1035	3.525×10^{-2}	2037	1.305×10^{-3}
ARWHEAD	100	1×10^{-5}	201**	0.000	1765	1.925×10^{-4}	2444	5.061×10^{-6}
ARWHEAD	100	1×10^{-7}	201**	0.000	1434	1.346×10^{-5}	2660	6.841×10^{-9}
BARD	3	1×10^{-1}	20	1.665×10^{-1}	53	9.347×10^{-1}	102	2.119×10^{-3}
BARD	3	1×10^{-3}	36	5.403×10^{-4}	77	1.030×10^{-3}	116	1.876×10^{-3}
BARD	3	1×10^{-5}	47	1.256×10^{-3}	294	1.898×10^{-3}	310	2.731×10^{-6}
BARD	3	1×10^{-7}	87	1.685×10^{-7}	198	6.076×10^{-8}	230	3.753×10^{-9}
BDQRTIC	100	1×10^{-1}	2283	1.132	2482	5.246	3476	2.697×10^{-1}
BDQRTIC	100	1×10^{-3}	4736	9.554×10^{-2}	2664	1.099×10^{-1}	6583	4.061×10^{-3}
BDQRTIC	100	1×10^{-5}	7755	4.134×10^{-3}	6619	3.957×10^{-4}	11431	9.216×10^{-7}
BDQRTIC	100	1×10^{-7}	5083	6.419×10^{-6}	6543	6.829×10^{-6}	9934	4.423×10^{-9}
BIGGS3‡	3	1×10^{-1}	8	1.366	160	1.311	168	9.756×10^{-1}
BIGGS3‡	3	1×10^{-3}	67	2.251×10^{-3}	155	6.556×10^{-4}	389	1.958×10^{-5}
BIGGS3‡	3	1×10^{-5}	73	1.776×10^{-6}	78	2.858×10^{-6}	293	1.380×10^{-6}
BIGGS3‡	3	1×10^{-7}	85	1.154×10^{-8}	95	1.538×10^{-6}	389	8.357×10^{-9}
BIGGS5‡	5	1×10^{-1}	30	1.300	15	1.373	435	1.113×10^{-1}
BIGGS5‡	5	1×10^{-3}	113	3.932×10^{-2}	266	1.181×10^{-1}	253	3.142×10^{-2}
BIGGS5‡	5	1×10^{-5}	176	-4.496×10^{-3}	285	1.808×10^{-2}	818	8.487×10^{-6}
BIGGS‡5	5	1×10^{-7}	265	-5.529×10^{-3}	580	6.905×10^{-5}	1077	2.321×10^{-8}
BIGGS6	6	1×10^{-1}	30	3.144×10^{-1}	121	3.515×10^{-1}	283	2.855×10^{-1}
BIGGS6	6	1×10^{-3}	43	2.937×10^{-1}	126	2.897×10^{-1}	1549	3.092×10^{-2}
BIGGS6	6	1×10^{-5}	363	-4.550×10^{-3}	586	-3.402×10^{-3}	1324	-5.568×10^{-3}
BIGGS6	6	1×10^{-7}	657	-5.612×10^{-3}	1612	-5.608×10^{-3}	1778	-5.648×10^{-3}
BOX2	2	1×10^{-1}	2**	5.847×10^{-19}	187	7.590×10^{-2}	26	3.168×10^{-1}
BOX2	2	1×10^{-3}	2**	5.847×10^{-19}	66	4.861×10^{-1}	387	1.276×10^{-6}
BOX2	2	1×10^{-5}	2**	5.847×10^{-19}	42	4.089×10^{-6}	132	1.362×10^{-6}
BOX2	2	1×10^{-7}	2**	5.847×10^{-19}	56	1.849×10^{-6}	108	2.693×10^{-8}
BOX3	3	1×10^{-1}	2**	5.847×10^{-19}	28	2.056×10^{-2}	55	2.102×10^{-2}
BOX3	3	1×10^{-3}	2**	5.847×10^{-19}	181	1.408×10^{-3}	94	4.054×10^{-6}
BOX3	3	1×10^{-5}	2**	5.847×10^{-19}	86	9.697×10^{-6}	149	6.273×10^{-7}
BOX3	3	1×10^{-7}	2**	5.847×10^{-19}	67	6.791×10^{-7}	370	4.093×10^{-8}
BRKMCC	2	1×10^{-1}	14	1.160×10^{-2}	107	1.085×10^{-1}	71	4.542×10^{-3}
BRKMCC	2	1×10^{-3}	22	8.434×10^{-8}	55	3.649×10^{-4}	51	1.407×10^{-6}
BRKMCC	2	1×10^{-5}	10	1.371×10^{-7}	45	5.221×10^{-5}	62	2.565×10^{-10}
BRKMCC	2	1×10^{-7}	19	3.325×10^{-10}	34	4.961×10^{-8}	156	1.995×10^{-10}

Table A.6. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
BROWNAL	10	1×10^{-1}	46	9.212×10^{-2}	53	2.719×10^{-2}	384	4.202×10^{-5}
BROWNAL	10	1×10^{-3}	123	1.232×10^{-2}	65	5.759×10^{-5}	223	2.969×10^{-5}
BROWNAL	10	1×10^{-5}	248	3.218×10^{-4}	77	3.062×10^{-5}	740	1.092×10^{-5}
BROWNAL	10	1×10^{-7}	458	6.915×10^{-6}	77	2.974×10^{-5}	821	2.971×10^{-5}
BROWNAL	100	1×10^{-1}	610	1.164	725	5.055×10^{-2}	2664	1.297×10^{-5}
BROWNAL	100	1×10^{-3}	4084	2.877×10^{-2}	725	5.541×10^{-4}	3677	1.287×10^{-5}
BROWNAL	100	1×10^{-5}	7907	3.839×10^{-4}	725	1.908×10^{-5}	3763	3.096×10^{-8}
BROWNAL	100	1×10^{-7}	11449	7.999×10^{-5}	827	1.336×10^{-5}	3540	2.445×10^{-8}
BROWNAL	200	1×10^{-1}	1514	7.945×10^{-1}	821	5.423×10^{-3}	7494	8.719×10^{-7}
BROWNAL	200	1×10^{-3}	11839	3.645×10^{-2}	821	4.205×10^{-4}	7494	8.234×10^{-7}
BROWNAL	200	1×10^{-5}	22817	3.574×10^{-4}	1833	1.073×10^{-5}	5291	9.198×10^{-9}
BROWNAL	200	1×10^{-7}	26754	5.228×10^{-6}	1833	9.284×10^{-7}	1626	8.243×10^{-7}
BROWNDEN	4	1×10^{-1}	219	3.762×10^{-2}	151	5.209×10^{-1}	327	1.623×10^{-4}
BROWNDEN	4	1×10^{-3}	163	3.183×10^{-4}	358	3.298×10^{-3}	276	5.083×10^{-7}
BROWNDEN	4	1×10^{-5}	239	1.428×10^{-6}	351	3.116×10^{-5}	751	2.212×10^{-9}
BROWNDEN	4	1×10^{-7}	224	1.020×10^{-8}	159	1.909×10^{-7}	292	-4.075×10^{-10}
CLIFF	2	1×10^{-1}	19	9.252×10^{-2}	51	2.902×10^{2}	809	5.153×10^{-1}
CLIFF	2	1×10^{-3}	31	1.389×10^{-3}	201	2.902×10^{2}	412	3.190×10^{-4}
CLIFF	2	1×10^{-5}	25	8.089×10^{-4}	644	1.027	272	2.209×10^{-4}
CLIFF	2	1×10^{-7}	68	1.705×10^{-8}	407	4.054×10^{-2}	335	2.192×10^{-4}
CRAGGLVY	4	1×10^{-1}	14	8.946×10^{-1}	253	7.327×10^{-1}	285	4.782×10^{-2}
CRAGGLVY	4	1×10^{-3}	72	4.048×10^{-3}	271	7.551×10^{-3}	147	5.397×10^{-4}
CRAGGLVY	4	1×10^{-5}	173	1.960×10^{-5}	262	8.396×10^{-5}	338	6.253×10^{-6}
CRAGGLVY	4	1×10^{-7}	199	1.186×10^{-6}	160	3.749×10^{-4}	620	3.188×10^{-8}
CRAGGLVY	10	1×10^{-1}	195	6.511×10^{-1}	663	3.023×10^{1}	912	2.336×10^{-1}
CRAGGLVY	10	1×10^{-3}	345	4.337×10^{-2}	731	2.444×10^{-2}	828	5.647×10^{-4}
CRAGGLVY	10	1×10^{-5}	573	7.120×10^{-4}	716	6.673×10^{-4}	1466	4.751×10^{-7}
CRAGGLVY	10	1×10^{-7}	1066	7.874×10^{-7}	1602	5.150×10^{-6}	1466	4.121×10^{-9}
CRAGGLVY	50	1×10^{-1}	1364	1.968	2041	5.834	3529	8.585×10^{-1}
CRAGGLVY	50	1×10^{-3}	1853	4.989×10^{-2}	4908	5.233×10^{-2}	4192	9.901×10^{-3}
CRAGGLVY	50	1×10^{-5}	2312	4.363×10^{-4}	3101	9.898×10^{-4}	5860	1.281×10^{-5}
CRAGGLVY	50	1×10^{-7}	3654	2.357×10^{-6}	7127	9.978×10^{-6}	7120	4.856×10^{-8}
CRAGGLVY	100	1×10^{-1}	3091	2.974	5275	1.228×10^{1}	7272	1.469
CRAGGLVY	100	1×10^{-3}	5041	1.878×10^{-1}	3918	6.386×10^{-1}	7959	1.298×10^{-2}
CRAGGLVY	100	1×10^{-5}	4955	4.727×10^{-4}	6241	5.729×10^{-3}	15314	1.573×10^{-5}
CRAGGLVY	100	1×10^{-7}	7008	5.223×10^{-6}	11106	3.053×10^{-5}	12688	2.635×10^{-7}
CUBE	2	1×10^{-1}	26	2.098	34	6.758×10^{-2}	65	4.380×10^{-2}
CUBE	2	1×10^{-3}	68	9.524×10^{-2}	43	4.164×10^{-2}	346	7.350×10^{-4}
CUBE	2	1×10^{-5}	135	1.075×10^{-3}	40	4.267×10^{-2}	213	2.049×10^{-7}
CUBE	2	1×10^{-7}	145	3.614×10^{-6}	141	9.286×10^{-4}	563	8.023×10^{-8}
DENSCHNA	2	1×10^{-1}	12	1.802×10^{-1}	31	1.032×10^{-1}	72	7.139×10^{-2}
DENSCHNA	2	1×10^{-3}	18	2.990×10^{-5}	66	1.196×10^{-3}	387	1.375×10^{-6}
DENSCHNA	2	1×10^{-5}	22	3.337×10^{-7}	73	1.087×10^{-5}	82	6.062×10^{-9}
DENSCHNA	2	1×10^{-7}	22	1.233×10^{-10}	44	5.951×10^{-8}	151	1.453×10^{-11}

Table A.7. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

A.3.3.2. Noiseless Functions. In this section, we list the final objective value($\phi(x)$), number of function evaluations (#feval), CPU time (CPU), and feasibility error(feaserr) for each problem instance in Tables A.28-A.29.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
DENSCHNB	2	1×10^{-1}	14	2.282×10^{-2}	23	2.369×10^{-1}	293	4.982×10^{-4}
DENSCHNB	2	1×10^{-3}	17	1.807×10^{-5}	40	8.803×10^{-4}	387	1.432×10^{-6}
DENSCHNB	2	1×10^{-5}	27	2.693×10^{-7}	61	1.017×10^{-6}	42	5.983×10^{-11}
DENSCHNB	2	1×10^{-7}	26	7.582×10^{-9}	32	2.194×10^{-8}	136	9.498×10^{-12}
DENSCHNC	2	1×10^{-1}	30	-7.755×10^{-2}	18	5.033×10^{-2}	281	3.323×10^{-2}
DENSCHNC	2	1×10^{-3}	62	2.780×10^{-5}	80	1.775×10^{-2}	219	2.208×10^{-5}
DENSCHNC	2	1×10^{-5}	55	-1.834×10^{-1}	160	6.362×10^{-6}	387	1.055×10^{-8}
DENSCHNC	2	1×10^{-7}	54	-1.834×10^{-1}	218	2.200×10^{-6}	136	-1.285×10^{-10}
DENSCHND	3	1×10^{-1}	142	3.571×10^{1}	269	3.030×10^{1}	228	1.318×10^{-3}
DENSCHND	3	1×10^{-3}	279	2.258×10^{-3}	216	1.288×10^{1}	414	4.836×10^{-5}
DENSCHND	3	1×10^{-5}	216	2.909×10^{-4}	240	2.218×10^{-5}	326	7.702×10^{-6}
DENSCHND	3	1×10^{-7}	319	1.084×10^{-6}	432	1.318×10^{-7}	679	8.619×10^{-8}
DENSCHNE	3	1×10^{-1}	29	1.570	44	1.003	66	1.038
DENSCHNE	3	1×10^{-3}	42	1.003	132	9.994×10^{-1}	138	9.993×10^{-1}
DENSCHNE	3	1×10^{-5}	73	9.999×10^{-1}	75	9.993×10^{-1}	389	9.993×10^{-1}
DENSCHNE	3	1×10^{-7}	116	1.080×10^{-8}	346	9.993×10^{-1}	295	1.651×10^{-10}
DENSCHNF	2	1×10^{-1}	19	3.023×10^{-3}	199	1.909×10^{-2}	100	2.349×10^{-3}
DENSCHNF	2	1×10^{-3}	26	1.388×10^{-5}	80	2.299×10^{-4}	100	8.045×10^{-6}
DENSCHNF	2	1×10^{-5}	32	2.913×10^{-8}	59	6.314×10^{-7}	77	2.843×10^{-8}
DENSCHNF	2	1×10^{-7}	38	2.403×10^{-9}	58	8.740×10^{-10}	136	4.048×10^{-11}
DIXMAANA	15	1×10^{-1}	202	2.011	278	3.989	727	3.807×10^{-3}
DIXMAANA	15	1×10^{-3}	401	1.671×10^{-2}	426	4.111×10^{-3}	516	7.512×10^{-6}
DIXMAANA	15	1×10^{-5}	564	5.826×10^{-5}	291	2.041×10^{-5}	413	2.145×10^{-8}
DIXMAANA	15	1×10^{-7}	650	7.482×10^{-7}	170	1.288×10^{-6}	660	5.438×10^{-11}
DIXMAANA	90	1×10^{-1}	1357	9.089×10^{-1}	1596	1.935×10^{1}	4121	1.338×10^{-2}
DIXMAANA	90	1×10^{-3}	1820	1.564×10^{-2}	6783	4.443×10^{-2}	3657	3.093×10^{-5}
DIXMAANA	90	1×10^{-5}	2384	2.123×10^{-4}	3552	2.365×10^{-4}	3657	1.304×10^{-7}
DIXMAANA	90	1×10^{-7}	2302	1.072×10^{-6}	2031	1.824×10^{-6}	4663	1.690×10^{-10}
DIXMAANA	300	1×10^{-1}	11454	2.772	2091	5.752×10^{1}	13555	6.279×10^{-2}
DIXMAANA	300	1×10^{-3}	12409	3.370×10^{-2}	5739	4.750×10^{-2}	7272	1.397×10^{-4}
DIXMAANA	300	1×10^{-5}	21687	2.406×10^{-4}	4530	6.880×10^{-4}	17895	3.066×10^{-7}
DIXMAANA	300	1×10^{-7}	18612	3.404×10^{-6}	5134	6.485×10^{-6}	7860	8.727×10^{-10}
DIXMAANB	15	1×10^{-1}	199	1.796	136	2.402	278	6.595×10^{-3}
DIXMAANB	15	1×10^{-3}	384	1.563×10^{-2}	700	7.149×10^{-3}	270	1.424×10^{-5}
DIXMAANB	15	1×10^{-5}	389	1.644×10^{-5}	351	5.285×10^{-5}	505	1.552×10^{-8}
DIXMAANB	15	1×10^{-7}	373	2.098×10^{-7}	274	6.291×10^{-7}	491	5.018×10^{-10}
DIXMAANB	90	1×10^{-1}	1588	1.101	277	1.692×10^{1}	1880	2.910×10^{-2}
DIXMAANB	90	1×10^{-3}	1406	1.029×10^{-2}	4163	1.213×10^{-2}	1101	5.186×10^{-5}
DIXMAANB	90	1×10^{-5}	1918	2.728×10^{-4}	1568	9.443×10^{-5}	2937	2.263×10^{-7}
DIXMAANB	90	1×10^{-7}	1932	1.523×10^{-6}	2282	5.621×10^{-7}	2937	1.318×10^{-9}
DIXMAANB	300	1×10^{-1}	7971	2.504	905	5.660×10^{1}	11680	5.947×10^{-2}
DIXMAANB	300	1×10^{-3}	12927	3.555×10^{-2}	11239	9.529×10^{-2}	7371	2.935×10^{-4}
DIXMAANB	300	1×10^{-5}	14656	2.960×10^{-4}	8463	2.769×10^{-3}	14914	1.849×10^{-6}
DIXMAANB	300	1×10^{-7}	19729	3.256×10^{-6}	5138	2.711×10^{-5}	7282	1.165×10^{-9}

Table A.8. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

We now comment on outcomes (ii)-(iv) mentioned in Section 2.4.1.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
DIXMAANC	15	1×10^{-1}	123	3.678×10^{-1}	696	9.417×10^{-1}	413	4.277×10^{-3}
DIXMAANC	15	1×10^{-3}	214	1.694×10^{-3}	784	9.657×10^{-3}	627	7.834×10^{-5}
DIXMAANC	15	1×10^{-5}	252	1.523×10^{-4}	621	3.658×10^{-5}	413	2.695×10^{-8}
DIXMAANC	15	1×10^{-7}	249	2.681×10^{-7}	528	4.150×10^{-8}	627	1.283×10^{-10}
DIXMAANC	90	1×10^{-1}	853	1.678	2077	3.626	2649	1.112×10^{-2}
DIXMAANC	90	1×10^{-3}	2546	1.886×10^{-2}	1016	6.657×10^{-2}	1880	4.973×10^{-5}
DIXMAANC	90	1×10^{-5}	2245	1.953×10^{-4}	2971	4.662×10^{-4}	1880	1.398×10^{-7}
DIXMAANC	90	1×10^{-7}	2873	2.169×10^{-6}	2869	9.707×10^{-6}	5905	1.894×10^{-10}
DIXMAANC	300	1×10^{-1}	5706	2.660	908	2.909×10^{1}	12855	5.278×10^{-2}
DIXMAANC	300	1×10^{-3}	15396	2.200×10^{-2}	9990	1.601×10^{-1}	13555	1.924×10^{-4}
DIXMAANC	300	1×10^{-5}	17952	3.248×10^{-4}	8167	2.040×10^{-4}	9111	2.439×10^{-7}
DIXMAANC	300	1×10^{-7}	20832	3.234×10^{-6}	7261	1.070×10^{-5}	6639	9.021×10^{-10}
DIXMAAND	15	1×10^{-1}	197	2.209	71	2.464	394	1.834×10^{-2}
DIXMAAND	15	1×10^{-3}	189	1.568×10^{-3}	210	3.272×10^{-3}	1730	6.885×10^{-6}
DIXMAAND	15	1×10^{-5}	265	2.640×10^{-5}	517	1.972×10^{-4}	660	1.423×10^{-7}
DIXMAAND	15	1×10^{-7}	245	3.120×10^{-7}	583	1.876×10^{-6}	1730	6.047×10^{-11}
DIXMAAND	90	1×10^{-1}	1468	1.597	647	4.471	1880	2.476×10^{-2}
DIXMAAND	90	1×10^{-3}	2428	2.428×10^{-2}	2637	6.034×10^{-2}	3075	1.075×10^{-4}
DIXMAAND	90	1×10^{-5}	1511	2.151×10^{-4}	5883	2.672×10^{-4}	2449	2.126×10^{-7}
DIXMAAND	90	1×10^{-7}	2339	1.634×10^{-6}	2957	1.455×10^{-6}	3970	4.576×10^{-10}
DIXMAAND	300	1×10^{-1}	6087	3.295	1211	2.487×10^{1}	11680	6.232×10^{-2}
DIXMAAND	300	1×10^{-3}	28875	4.695×10^{-2}	15937	2.861×10^{-2}	13555	2.324×10^{-4}
DIXMAAND	300	1×10^{-5}	36583	1.968×10^{-4}	7259	1.237×10^{-3}	6695	8.301×10^{-7}
DIXMAAND	300	1×10^{-7}	31821	2.379×10^{-6}	6958	1.154×10^{-5}	7282	1.251×10^{-9}
DIXMAANE	15	1×10^{-1}	117	2.680	228	2.593	516	4.750×10^{-2}
DIXMAANE	15	1×10^{-3}	483	6.719×10^{-2}	907	2.567×10^{-3}	759	9.723×10^{-5}
DIXMAANE	15	1×10^{-5}	359	6.662×10^{-5}	496	4.389×10^{-5}	875	3.665×10^{-8}
DIXMAANE	15	1×10^{-7}	473	5.981×10^{-8}	515	4.413×10^{-7}	1730	2.043×10^{-10}
DIXMAANE	90	1×10^{-1}	957	2.077	2228	1.048×10^{1}	5905	1.522×10^{-1}
DIXMAANE	90	1×10^{-3}	2231	5.295×10^{-2}	2558	5.724×10^{-2}	7854	2.745×10^{-3}
DIXMAANE	90	1×10^{-5}	2422	1.047×10^{-3}	3592	1.658×10^{-3}	14178	2.990×10^{-6}
DIXMAANE	90	1×10^{-7}	3565	3.468×10^{-6}	6643	5.020×10^{-6}	9472	1.434×10^{-8}
DIXMAANE	300	1×10^{-1}	5797	6.472	5878	1.576×10^{1}	17895	5.082×10^{-1}
DIXMAANE	300	1×10^{-3}	39321	1.160×10^{-1}	11096	1.216×10^{-1}	42978	5.040×10^{-3}
DIXMAANE	300	1×10^{-5}	35575	3.907×10^{-3}	16015	3.572×10^{-3}	41288	6.960×10^{-5}
DIXMAANE	300	1×10^{-7}	29189	1.864×10^{-5}	27787	3.545×10^{-5}	51466	5.905×10^{-8}
DIXMAANF	15	1×10^{-1}	186	1.292	133	9.307×10^{-1}	394	8.259×10^{-2}
DIXMAANF	15	1×10^{-3}	189	1.051×10^{-2}	409	5.640×10^{-3}	413	3.139×10^{-5}
DIXMAANF	15	1×10^{-5}	296	2.329×10^{-5}	574	1.101×10^{-4}	825	8.220×10^{-8}
DIXMAANF	15	1×10^{-7}	351	2.811×10^{-7}	546	1.188×10^{-6}	1730	1.116×10^{-10}
DIXMAANF	90	1×10^{-1}	1055	1.623	277	6.676	3143	7.567×10^{-2}
DIXMAANF	90	1×10^{-3}	1855	1.150×10^{-1}	2279	3.255×10^{-2}	6862	1.656×10^{-3}
DIXMAANF	90	1×10^{-5}	3117	2.379×10^{-4}	4615	5.649×10^{-4}	6955	1.941×10^{-6}
DIXMAANF	90	1×10^{-7}	6296	1.248×10^{-5}	4327	6.372×10^{-6}	9524	6.810×10^{-9}
DIXMAANF	300	1×10^{-1}	4652	2.581	904	2.266×10^{1}	11680	1.549×10^{-1}
DIXMAANF	300	1×10^{-3}	10797	1.173×10^{-1}	4840	2.153×10^{-1}	31414	5.266×10^{-3}
DIXMAANF	300	1×10^{-5}	41760	2.501×10^{-3}	15493	6.550×10^{-3}	41516	2.248×10^{-5}
DIXMAANF	300	1×10^{-7}	17405	3.175×10^{-5}	22658	3.595×10^{-5}	59871	2.087×10^{-8}

Table A.9. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
DIXMAANG	15	1×10^{-1}	118	1.301	53	8.760×10^{-1}	1730	2.446×10^{-2}
DIXMAANG	15	1×10^{-3}	219	3.590×10^{-3}	532	8.830×10^{-3}	1104	4.679×10^{-5}
DIXMAANG	15	1×10^{-5}	259	1.628×10^{-5}	310	9.965×10^{-4}	1730	9.830×10^{-8}
DIXMAANG	15	1×10^{-7}	354	1.419×10^{-7}	1235	2.106×10^{-6}	1730	3.410×10^{-10}
DIXMAANG	90	1×10^{-1}	1253	1.630	2284	1.752	4496	9.061×10^{-2}
DIXMAANG	90	1×10^{-3}	3690	9.785×10^{-2}	2011	3.834×10^{-2}	6194	1.981×10^{-3}
DIXMAANG	90	1×10^{-5}	2545	3.136×10^{-4}	2579	8.481×10^{-4}	6124	7.697×10^{-6}
DIXMAANG	90	1×10^{-7}	3047	8.060×10^{-6}	7112	6.479×10^{-6}	8861	7.244×10^{-9}
DIXMAANG	300	1×10^{-1}	7837	3.509	908	1.259×10^{1}	12260	3.429×10^{-1}
DIXMAANG	300	1×10^{-3}	40449	1.096×10^{-1}	9353	2.259×10^{-1}	17101	4.560×10^{-3}
DIXMAANG	300	1×10^{-5}	33893	3.070×10^{-3}	8463	2.846×10^{-3}	35863	3.047×10^{-5}
DIXMAANG	300	1×10^{-7}	149998	4.680×10^{-4}	21147	6.501×10^{-5}	44939	9.867×10^{-8}
DIXMAANH	15	1×10^{-1}	167	5.656×10^{-1}	54	9.557×10^{-1}	394	2.593×10^{-2}
DIXMAANH	15	1×10^{-3}	280	7.738×10^{-3}	395	4.275×10^{-3}	705	3.527×10^{-5}
DIXMAANH	15	1×10^{-5}	365	3.603×10^{-5}	419	6.578×10^{-5}	825	7.652×10^{-8}
DIXMAANH	15	1×10^{-7}	400	1.836×10^{-7}	767	2.555×10^{-7}	830	9.052×10^{-10}
DIXMAANH	90	1×10^{-1}	1598	2.375	963	1.997	3163	8.808×10^{-2}
DIXMAANH	90	1×10^{-3}	2287	1.070×10^{-1}	3112	9.064×10^{-2}	8780	3.029×10^{-3}
DIXMAANH	90	1×10^{-5}	3252	2.565×10^{-4}	5492	8.586×10^{-4}	6194	4.423×10^{-6}
DIXMAANH	90	1×10^{-7}	5217	3.169×10^{-6}	4430	6.061×10^{-6}	8780	4.295×10^{-9}
DIXMAANH	300	1×10^{-1}	19419	5.492	1211	1.205×10^{1}	6759	3.222×10^{-1}
DIXMAANH	300	1×10^{-3}	33582	2.678×10^{-1}	15009	1.093×10^{-1}	17702	4.861×10^{-3}
DIXMAANH	300	1×10^{-5}	25154	4.098×10^{-3}	15460	4.812×10^{-3}	40314	1.688×10^{-5}
DIXMAANH	300	1×10^{-7}	25753	1.920×10^{-5}	22065	2.289×10^{-5}	41288	8.904×10^{-8}
DIXMAANI	15	1×10^{-1}	113	5.926×10^{-1}	1218	1.032	516	3.909×10^{-2}
DIXMAANI	15	1×10^{-3}	358	6.615×10^{-3}	504	1.797×10^{-2}	1053	2.345×10^{-3}
DIXMAANI	15	1×10^{-5}	467	6.109×10^{-4}	1343	3.261×10^{-4}	1476	1.374×10^{-6}
DIXMAANI	15	1×10^{-7}	636	6.946×10^{-6}	792	1.308×10^{-6}	1611	4.603×10^{-9}
DIXMAANI	90	1×10^{-1}	951	2.135	3000	4.382	7696	2.633×10^{-1}
DIXMAANI	90	1×10^{-3}	1824	4.215×10^{-2}	2824	5.241×10^{-2}	7466	5.757×10^{-3}
DIXMAANI	90	1×10^{-5}	3278	7.114×10^{-3}	5618	4.747×10^{-3}	18792	1.816×10^{-4}
DIXMAANI	90	1×10^{-7}	10692	2.134×10^{-4}	12783	1.281×10^{-4}	36022	5.632×10^{-7}
DIXMAANI	300	1×10^{-1}	7480	5.731	12317	1.749×10^{1}	21554	9.401×10^{-1}
DIXMAANI	300	1×10^{-3}	12194	1.431×10^{-1}	10792	1.467×10^{-1}	43977	1.445×10^{-2}
DIXMAANI	300	1×10^{-5}	25477	9.917×10^{-3}	34341	6.574×10^{-3}	70880	3.511×10^{-4}
DIXMAANI	300	1×10^{-7}	36578	1.928×10^{-4}	74070	2.406×10^{-4}	150151*	1.779×10^{-5}
DIXMAANJ	15	1×10^{-1}	156	1.786×10^{-1}	52	6.219×10^{-1}	1124	3.543×10^{-2}
DIXMAANJ	15	1×10^{-3}	309	2.567×10^{-2}	584	1.557×10^{-2}	1536	7.979×10^{-4}
DIXMAANJ	15	1×10^{-5}	672	7.838×10^{-5}	835	2.748×10^{-4}	1124	7.681×10^{-7}
DIXMAANJ	15	1×10^{-7}	685	4.621×10^{-7}	789	3.246×10^{-5}	2031	1.804×10^{-8}
DIXMAANJ	90	1×10^{-1}	1141	1.187	277	3.668	1880	7.149×10^{-2}
DIXMAANJ	90	1×10^{-3}	2186	4.912×10^{-2}	2853	5.441×10^{-2}	7696	2.858×10^{-3}
DIXMAANJ	90	1×10^{-5}	4146	3.591×10^{-3}	3791	3.090×10^{-3}	11399	4.242×10^{-5}
DIXMAANJ	90	1×10^{-7}	13470	6.305×10^{-4}	7251	3.692×10^{-5}	29130	2.883×10^{-7}
DIXMAANJ	300	1×10^{-1}	4325	2.391	904	1.236×10^{1}	11680	1.834×10^{-1}
DIXMAANJ	300	1×10^{-3}	23360	1.285×10^{-1}	8042	1.123×10^{-1}	36345	8.119×10^{-3}
DIXMAANJ	300	1×10^{-5}	19573	3.531×10^{-3}	18352	5.437×10^{-3}	41369	5.491×10^{-5}
DIXMAANJ	300	1×10^{-7}	31059	1.801×10^{-4}	33637	5.060×10^{-5}	84610	2.777×10^{-6}

Table A.10. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
DIXMAANK	15	1×10^{-1}	123	2.036×10^{-1}	70	3.310×10^{-1}	270	5.345×10^{-2}
DIXMAANK	15	1×10^{-3}	226	4.148×10^{-2}	1229	1.660×10^{-2}	1024	8.734×10^{-4}
DIXMAANK	15	1×10^{-5}	533	4.054×10^{-4}	759	7.894×10^{-5}	1730	3.452×10^{-7}
DIXMAANK	15	1×10^{-7}	710	1.111×10^{-6}	724	3.219×10^{-7}	1595	4.578×10^{-9}
DIXMAANK	90	1×10^{-1}	962	2.115	961	9.536×10^{-1}	1880	3.049×10^{-1}
DIXMAANK	90	1×10^{-3}	1921	4.371×10^{-2}	2718	9.449×10^{-2}	10211	3.178×10^{-3}
DIXMAANK	90	1×10^{-5}	3646	8.827×10^{-4}	7155	1.234×10^{-3}	8157	3.217×10^{-5}
DIXMAANK	90	1×10^{-7}	6961	5.780×10^{-4}	5158	2.208×10^{-5}	14626	1.152×10^{-6}
DIXMAANK	300	1×10^{-1}	4131	3.657	908	6.765	12777	4.528×10^{-1}
DIXMAANK	300	1×10^{-3}	30156	1.396×10^{-1}	10483	3.126×10^{-1}	21925	3.813×10^{-3}
DIXMAANK	300	1×10^{-5}	17158	7.029×10^{-3}	16876	3.556×10^{-3}	26675	6.708×10^{-5}
DIXMAANK	300	1×10^{-7}	52657	1.527×10^{-4}	15408	7.813×10^{-5}	82360	6.627×10^{-7}
DIXMAANL	15	1×10^{-1}	152	1.451	88	5.246×10^{-1}	1246	5.972×10^{-2}
DIXMAANL	15	1×10^{-3}	370	3.861×10^{-2}	866	1.485×10^{-2}	1694	5.900×10^{-4}
DIXMAANL	15	1×10^{-5}	782	6.330×10^{-5}	1050	1.029×10^{-4}	1476	1.103×10^{-6}
DIXMAANL	15	1×10^{-7}	873	1.152×10^{-6}	638	1.039×10^{-6}	1730	2.064×10^{-8}
DIXMAANL	90	1×10^{-1}	1230	1.224	1363	1.218	2817	1.846×10^{-1}
DIXMAANL	90	1×10^{-3}	2191	4.680×10^{-2}	2539	1.429×10^{-1}	5193	2.433×10^{-3}
DIXMAANL	90	1×10^{-5}	3048	1.131×10^{-3}	3152	2.455×10^{-3}	8780	3.050×10^{-5}
DIXMAANL	90	1×10^{-7}	7705	2.888×10^{-4}	5990	1.848×10^{-5}	32927	1.567×10^{-7}
DIXMAANL	300	1×10^{-1}	7660	3.606	1211	6.900	11674	1.739×10^{-1}
DIXMAANL	300	1×10^{-3}	33194	1.813×10^{-1}	11079	2.767×10^{-1}	19562	4.733×10^{-3}
DIXMAANL	300	1×10^{-5}	149998	3.705×10^{-2}	10571	5.175×10^{-3}	26767	4.075×10^{-5}
DIXMAANL	300	1×10^{-7}	61830	7.296×10^{-4}	24171	5.364×10^{-5}	57101	1.891×10^{-6}
DQRTIC	10	1×10^{-1}	139	1.786×10^{-1}	927	1.449×10^{-1}	762	2.816×10^{-4}
DQRTIC	10	1×10^{-3}	152	4.676×10^{-3}	304	6.109×10^{-3}	749	3.770×10^{-5}
DQRTIC	10	1×10^{-5}	429	8.020×10^{-5}	943	3.905×10^{-5}	675	8.733×10^{-9}
DQRTIC	10	1×10^{-7}	405	3.782×10^{-8}	676	1.675×10^{-6}	675	1.032×10^{-8}
DQRTIC	50	1×10^{-1}	1672	7.132×10^{-1}	1197	8.681×10^{-1}	2857	7.958×10^{-2}
DQRTIC	50	1×10^{-3}	1719	7.324×10^{-3}	2008	7.825×10^{-3}	2651	3.038×10^{-4}
DQRTIC	50	1×10^{-5}	2369	2.603×10^{-5}	2629	1.445×10^{-4}	3341	1.758×10^{-6}
DQRTIC	50	1×10^{-7}	2435	1.473×10^{-6}	2493	1.915×10^{-6}	3341	2.992×10^{-8}
DQRTIC	100	1×10^{-1}	4920	3.885×10^{-1}	4766	6.422	7146	4.248×10^{-2}
DQRTIC	100	1×10^{-3}	5612	9.569×10^{-3}	4517	2.474×10^{-3}	5960	1.807×10^{-4}
DQRTIC	100	1×10^{-5}	6494	5.767×10^{-5}	6534	7.941×10^{-4}	7146	2.060×10^{-6}
DQRTIC	100	1×10^{-7}	6913	8.633×10^{-7}	1119	1.906×10^{7}	7344	2.041×10^{-8}
EDENSCH	36	1×10^{-1}	873	9.721×10^{-1}	1071	1.223	2104	1.290×10^{-1}
EDENSCH	36	1×10^{-3}	1175	1.187×10^{-2}	2031	1.116×10^{-2}	1772	3.795×10^{-4}
EDENSCH	36	1×10^{-5}	1166	1.001×10^{-4}	1324	5.176×10^{-4}	1772	8.173×10^{-7}
EDENSCH	36	1×10^{-7}	1331	1.083×10^{-6}	1300	3.572×10^{-6}	1824	2.417×10^{-9}
EIGENALS	6	1×10^{-1}	5**	0.000	96	5.364×10^{-1}	509	2.108×10^{-1}
EIGENALS	6	1×10^{-3}	5**	0.000	82	1.041×10^{-2}	252	5.299×10^{-4}
EIGENALS	6	1×10^{-5}	5**	0.000	250	4.298×10^{-4}	152	1.284×10^{-7}
EIGENALS	6	1×10^{-7}	5**	0.000	294	5.331×10^{-6}	395	1.289×10^{-10}
EIGENALS	110	1×10^{-1}	2393	9.021	3176	1.692×10^{1}	8820	1.404×10^{-1}
EIGENALS	110	1×10^{-3}	6471	8.081×10^{-1}	3365	2.275×10^{-1}	5137	1.253×10^{-2}
EIGENALS	110	1×10^{-5}	36272	2.114×10^{-2}	6187	1.407×10^{-2}	41789	2.313×10^{-4}
EIGENALS	110	1×10^{-7}	54998	2.423×10^{-4}	32572	1.418×10^{-4}	55103*	2.471×10^{-5}

Table A.11. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
EIGENBLS‡	6	1×10^{-1}	51	1.421×10^{-1}	32	1.096	252	3.986×10^{-3}
EIGENBLS‡	6	1×10^{-3}	109	3.843×10^{-2}	240	1.223×10^{-3}	376	4.336×10^{-5}
EIGENBLS‡	6	1×10^{-5}	217	-1.840×10^{-1}	247	2.792×10^{-5}	299	1.340×10^{-7}
EIGENBLS‡	6	1×10^{-7}	307	-1.849×10^{-1}	190	9.576×10^{-8}	256	5.404×10^{-10}
EIGENBLS	110	1×10^{-1}	705	3.389	1761	1.122×10^{1}	4023	1.707
EIGENBLS	110	1×10^{-3}	5295	1.452	2754	1.596	7292	1.552
EIGENBLS	110	1×10^{-5}	18185	2.700×10^{-2}	21793	2.398×10^{-2}	55096*	5.814×10^{-3}
EIGENBLS	110	1×10^{-7}	30763	5.299×10^{-4}	46722	9.959×10^{-4}	55088*	2.591×10^{-2}
EIGENCLS	30	1×10^{-1}	400	1.610	1107	1.185×10^{1}	1812	4.235×10^{-1}
EIGENCLS	30	1×10^{-3}	686	3.905×10^{-1}	1546	2.774×10^{-2}	3424	1.360×10^{-3}
EIGENCLS	30	1×10^{-5}	1590	1.176×10^{-3}	1935	1.016×10^{-3}	4427	1.116×10^{-5}
EIGENCLS	30	1×10^{-7}	2212	6.465×10^{-6}	3159	1.266×10^{-5}	6325	3.770×10^{-8}
ENGVAL1	2	1×10^{-1}	22	3.602×10^{-2}	24	5.487×10^{-1}	146	1.987×10^{-2}
ENGVAL1	2	1×10^{-3}	30	9.036×10^{-5}	203	4.722×10^{-3}	146	2.480×10^{-5}
ENGVAL1	2	1×10^{-5}	33	5.987×10^{-8}	54	2.397×10^{-6}	91	2.265×10^{-8}
ENGVAL1	2	1×10^{-7}	36	3.361×10^{-9}	56	3.129×10^{-8}	91	1.713×10^{-11}
ENGVAL1	50	1×10^{-1}	1298	1.868	2612	6.147	1160	2.136×10^{-1}
ENGVAL1	50	1×10^{-3}	1125	2.268×10^{-2}	634	1.857×10^{-2}	3720	2.678×10^{-4}
ENGVAL1	50	1×10^{-5}	2643	1.160×10^{-4}	894	5.713×10^{-5}	1460	2.359×10^{-6}
ENGVAL1	50	1×10^{-7}	2802	9.129×10^{-7}	1354	1.628×10^{-6}	2651	2.889×10^{-9}
ENGVAL1	100	1×10^{-1}	2362	1.585	3555	1.187×10^{1}	1638	3.397×10^{-1}
ENGVAL1	100	1×10^{-3}	3400	1.462×10^{-2}	2805	4.521×10^{-2}	4885	8.850×10^{-4}
ENGVAL1	100	1×10^{-5}	3501	1.729×10^{-4}	1334	5.313×10^{-4}	5960	2.445×10^{-6}
ENGVAL1	100	1×10^{-7}	3166	1.293×10^{-6}	4913	3.878×10^{-6}	4095	6.088×10^{-9}
EXPFIT	2	1×10^{-1}	17	3.207	54	4.954	126	4.794×10^{-2}
EXPFIT	2	1×10^{-3}	33	3.946×10^{-3}	58	4.299×10^{-3}	67	4.256×10^{-5}
EXPFIT	2	1×10^{-5}	38	3.463×10^{-7}	107	2.064×10^{-6}	260	7.468×10^{-10}
EXPFIT	2	1×10^{-7}	49	8.778×10^{-10}	68	2.677×10^{-8}	91	5.019×10^{-11}
FLETCBV3‡	10	1×10^{-1}	21**	3.228×10^{-2}	32	3.228×10^{-2}	21	3.228×10^{-2}
FLETCBV3‡	10	1×10^{-3}	21**	3.228×10^{-2}	29	3.228×10^{-2}	21	3.228×10^{-2}
FLETCBV3‡	10	1×10^{-5}	22	3.228×10^{-2}	27	3.228×10^{-2}	21	3.228×10^{-2}
FLETCBV3‡	10	1×10^{-7}	965	8.774×10^{-4}	27	3.228×10^{-2}	403	3.228×10^{-2}
FLETCBV3‡	100	1×10^{-1}	202	1.785×10^{5}	2486	1.785×10^{5}	1952	1.785×10^{5}
FLETCBV3‡	100	1×10^{-3}	202	1.785×10^{5}	447	1.785×10^{5}	4029	1.785×10^{5}
FLETCBV3‡	100	1×10^{-5}	50000*	1.349×10^{5}	11892	1.785×10^{5}	49799	2.028×10^{2}
FLETCBV3‡	100	1×10^{-7}	49999	5.761×10^{4}	222	1.785×10^{5}	50010*	1.015×10^{1}
FLETCHBV‡	10	1×10^{-1}	863	2.240×10^{4}	639	2.653×10^{3}	1087	1.218×10^{4}
FLETCHBV‡	10	1×10^{-3}	1289	1.540×10^{5}	1118	8.389×10^{3}	1356	-4.764×10^{3}
FLETCHBV‡	10	1×10^{-5}	790	2.228×10^{5}	708	-4.605×10^{3}	1445	-7.519×10^{3}
FLETCHBV‡	10	1×10^{-7}	977	8.461×10^{4}	759	7.233×10^{2}	952	4.236×10^{2}
FLETCHBV‡	100	1×10^{-1}	50000*	1.581×10^{13}	40962	-7.187×10^{9}	50076*	-6.387×10^{9}
FLETCHBV‡	100	1×10^{-3}	50000*	1.731×10^{13}	50015*	-9.179×10^{9}	50085*	-3.093×10^{9}
FLETCHBV‡	100	1×10^{-5}	50000*	1.704×10^{13}	33119	-8.684×10^{8}	50068*	-5.710×10^{9}
FLETCHBV‡	100	1×10^{-7}	50000*	1.391×10^{13}	8369	7.425×10^{10}	50080*	-5.308×10^{9}

Table A.12. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

(ii) The solvers converged to feasible points with different objective function values.

There are three such problems, for all of which COBYLA terminated due to the limit on

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
FREUROTH	2	1×10^{-1}	38	6.304×10^{-2}	35	5.015×10^{1}	368	3.877×10^{-4}
FREUROTH	2	1×10^{-3}	35	5.507×10^{-1}	347	1.660×10^{-3}	141	1.496×10^{-5}
FREUROTH	2	1×10^{-5}	70	8.659×10^{-7}	83	9.381×10^{-5}	387	3.302×10^{-8}
FREUROTH	2	1×10^{-7}	73	4.594×10^{-10}	138	1.112×10^{-6}	261	4.996×10^{-10}
FREUROTH	10	1×10^{-1}	125	4.555×10^{1}	137	5.548×10^{1}	1466	8.708×10^{-3}
FREUROTH	10	1×10^{-3}	371	3.964×10^{-3}	467	1.341×10^{-1}	562	4.686×10^{-5}
FREUROTH	10	1×10^{-5}	433	3.004×10^{-4}	295	1.857×10^{-3}	842	5.755×10^{-8}
FREUROTH	10	1×10^{-7}	435	1.512×10^{-6}	602	1.348×10^{-5}	630	-3.206×10^{-10}
FREUROTH	50	1×10^{-1}	1629	5.744×10^{1}	2466	5.503×10^{1}	2651	6.734×10^{-2}
FREUROTH	50	1×10^{-3}	6822	6.137	3983	3.723×10^{-2}	2857	5.299×10^{-5}
FREUROTH	50	1×10^{-5}	7662	1.340×10^{-3}	5024	3.387×10^{-4}	2487	2.101×10^{-7}
FREUROTH	50	1×10^{-7}	6619	9.762×10^{-6}	1257	1.453×10^{-6}	3063	5.975×10^{-10}
FREUROTH	100	1×10^{-1}	1868	5.751×10^{1}	920	6.619×10^{1}	8609	6.268×10^{-2}
FREUROTH	100	1×10^{-3}	8308	9.577	3192	5.430×10^{-2}	6050	1.752×10^{-4}
FREUROTH	100	1×10^{-5}	5029	1.824×10^{-3}	4095	2.290×10^{-4}	5769	4.168×10^{-7}
FREUROTH	100	1×10^{-7}	5293	7.505×10^{-6}	6187	1.098×10^{-5}	5907	-5.191×10^{-9}
GENROSE	5	1×10^{-1}	35	3.849	68	3.246	180	2.711
GENROSE	5	1×10^{-3}	162	4.182×10^{-2}	234	2.566	446	9.023×10^{-4}
GENROSE	5	1×10^{-5}	221	8.543×10^{-5}	260	2.548×10^{-3}	413	2.278×10^{-7}
GENROSE	5	1×10^{-7}	261	9.023×10^{-7}	317	4.476×10^{-5}	454	1.113×10^{-9}
GENROSE	10	1×10^{-1}	84	9.259	221	8.902	805	8.892
GENROSE	10	1×10^{-3}	585	4.321×10^{-3}	446	8.816	1564	1.860×10^{-4}
GENROSE	10	1×10^{-5}	654	3.114×10^{-5}	1016	7.485×10^{-3}	1935	1.029×10^{-6}
GENROSE	10	1×10^{-7}	726	7.303×10^{-7}	964	8.881×10^{-5}	1904	5.181×10^{-10}
GENROSE	100	1×10^{-1}	1868	1.189×10^{2}	3231	1.335×10^{2}	7344	1.119×10^{2}
GENROSE	100	1×10^{-3}	15359	1.081×10^{2}	2247	1.108×10^{2}	50024*	6.559×10^{1}
GENROSE	100	1×10^{-5}	18075	6.885×10^{-3}	29786	5.081×10^{-3}	50009*	2.770×10^{-1}
GENROSE	100	1×10^{-7}	18030	1.719×10^{-6}	29847	4.748×10^{-5}	50005*	2.194×10^{-2}
GULF	3	1×10^{-1}	26	7.047	41	6.637	389	6.622
GULF	3	1×10^{-3}	38	7.032	38	6.622	229	4.534×10^{-3}
GULF	3	1×10^{-5}	115	6.882×10^{-2}	267	3.649×10^{-3}	370	4.462×10^{-3}
GULF	3	1×10^{-7}	333	6.930×10^{-3}	217	4.381×10^{-3}	688	2.460×10^{-7}
HAIRY	2	1×10^{-1}	49	3.471×10^{2}	71	2.721×10^{2}	387	1.950×10^{-4}
HAIRY	2	1×10^{-3}	351	2.426×10^{-4}	186	8.522×10^{-4}	413	2.326×10^{-8}
HAIRY	2	1×10^{-5}	226	1.256×10^{-8}	175	9.401×10^{-7}	516	5.074×10^{-11}
HAIRY	2	1×10^{-7}	252	2.339×10^{-8}	87	1.508×10^{-6}	210	4.974×10^{-14}
HELIX	3	1×10^{-1}	26	9.616×10^{-2}	188	7.615	374	2.785×10^{-1}
HELIX	3	1×10^{-3}	38	1.865×10^{-2}	314	4.147	23	8.522×10^{2}
HELIX	3	1×10^{-5}	72	1.822×10^{-6}	453	3.021×10^{-6}	23	8.521×10^{2}
HELIX	3	1×10^{-7}	84	4.567×10^{-7}	167	1.171×10^{-4}	23	8.521×10^{2}
JENSMP‡	2	1×10^{-1}	24	-1.786×10^{3}	18	0.000	21	0.000
JENSMP‡	2	1×10^{-3}	78	-1.896×10^{3}	18	0.000	21	0.000
JENSMP‡	2	1×10^{-5}	87	-1.896×10^{3}	18	0.000	21	0.000
JENSMP‡	2	1×10^{-7}	67	-1.896×10^{3}	18	0.000	21	0.000
KOWOSB	4	1×10^{-1}	1**	5.006×10^{-3}	72	1.383×10^{-3}	9	5.006×10^{-3}
KOWOSB	4	1×10^{-3}	20	2.240×10^{-3}	261	3.305×10^{-3}	104	2.435×10^{-4}
KOWOSB	4	1×10^{-5}	44	1.249×10^{-4}	112	1.767×10^{-4}	220	6.118×10^{-5}
KOWOSB	4	1×10^{-7}	111	2.694×10^{-5}	1309	6.915×10^{-7}	314	5.638×10^{-8}

Table A.13. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
MEXHAT	2	1×10^{-1}	30	1.166×10^{-2}	98	4.477×10^{-1}	146	4.231×10^{-1}
MEXHAT	2	1×10^{-3}	43	1.102×10^{-2}	80	8.450×10^{-1}	709	4.227×10^{-1}
MEXHAT	2	1×10^{-5}	45	1.110×10^{-2}	203	4.173×10^{-1}	232	1.568×10^{-3}
MEXHAT	2	1×10^{-7}	44	1.036×10^{-2}	210	4.226×10^{-1}	660	1.383×10^{-4}
MOREBV	10	1×10^{-1}	1**	1.599×10^{-2}	19	1.599×10^{-2}	21	1.599×10^{-2}
MOREBV	10	1×10^{-3}	81	1.262×10^{-2}	131	1.451×10^{-2}	937	1.506×10^{-3}
MOREBV	10	1×10^{-5}	184	2.345×10^{-3}	433	1.847×10^{-3}	796	1.441×10^{-6}
MOREBV	10	1×10^{-7}	386	1.283×10^{-7}	988	7.402×10^{-5}	1003	5.675×10^{-9}
MOREBV	50	1×10^{-1}	1**	1.321×10^{-3}	93	1.321×10^{-3}	101	1.321×10^{-3}
MOREBV	50	1×10^{-3}	105	1.122×10^{-3}	101	1.321×10^{-3}	828	4.204×10^{-4}
MOREBV	50	1×10^{-5}	746	3.981×10^{-4}	997	7.841×10^{-4}	2064	2.290×10^{-5}
MOREBV	50	1×10^{-7}	1322	3.021×10^{-5}	1283	4.621×10^{-5}	12825	7.628×10^{-6}
MOREBV	100	1×10^{-1}	1**	3.633×10^{-4}	190	3.633×10^{-4}	201	3.633×10^{-4}
MOREBV	100	1×10^{-3}	1**	3.633×10^{-4}	201	3.633×10^{-4}	2669	1.495×10^{-4}
MOREBV	100	1×10^{-5}	320	2.538×10^{-4}	309	3.571×10^{-4}	3457	6.772×10^{-6}
MOREBV	100	1×10^{-7}	3087	1.417×10^{-5}	2809	1.540×10^{-5}	14045	1.546×10^{-6}
NCB20B	21	1×10^{-1}	1**	3.000×10^{-3}	22	3.000×10^{-3}	232	3.177×10^{-4}
NCB20B	21	1×10^{-3}	113	1.441×10^{-5}	74	2.964×10^{-4}	421	6.287×10^{-6}
NCB20B	21	1×10^{-5}	260	1.402×10^{-4}	1332	5.741×10^{-6}	686	6.098×10^{-9}
NCB20B	21	1×10^{-7}	708	1.224×10^{-5}	260	4.816×10^{-8}	650	-5.574×10^{-11}
NCB20B	22	1×10^{-1}	121	5.384×10^{-3}	23	6.000×10^{-3}	388	1.036×10^{-3}
NCB20B	22	1×10^{-3}	121	1.817×10^{-3}	174	8.594×10^{-4}	935	8.921×10^{-4}
NCB20B	22	1×10^{-5}	265	2.971×10^{-4}	298	1.075×10^{-5}	578	3.829×10^{-6}
NCB20B	22	1×10^{-7}	715	2.307×10^{-5}	345	1.163×10^{-7}	908	2.372×10^{-9}
NCB20B	50	1×10^{-1}	212	9.199×10^{-2}	121	1.099×10^{-1}	938	1.883×10^{-2}
NCB20B	50	1×10^{-3}	734	7.836×10^{-3}	1513	1.296×10^{-3}	1866	3.565×10^{-4}
NCB20B	50	1×10^{-5}	1757	4.549×10^{-4}	838	3.965×10^{-4}	3623	9.630×10^{-5}
NCB20B	50	1×10^{-7}	4316	6.696×10^{-5}	2253	8.619×10^{-5}	23370	2.289×10^{-6}
NCB20B	100	1×10^{-1}	481	3.871×10^{-1}	1565	3.458×10^{-1}	2938	3.874×10^{-2}
NCB20B	100	1×10^{-3}	1748	1.447×10^{-2}	2557	2.895×10^{-2}	5703	8.588×10^{-4}
NCB20B	100	1×10^{-5}	5976	7.143×10^{-4}	5154	3.922×10^{-4}	10509	8.218×10^{-5}
NCB20B	100	1×10^{-7}	15651	4.301×10^{-5}	6944	6.476×10^{-5}	32343	1.546×10^{-5}
NCB20B	180	1×10^{-1}	1664	4.700×10^{-1}	1648	1.184	5833	1.772×10^{-1}
NCB20B	180	1×10^{-3}	6604	4.261×10^{-2}	2007	5.420×10^{-2}	12753	1.807×10^{-3}
NCB20B	180	1×10^{-5}	12066	1.067×10^{-3}	10462	3.299×10^{-4}	22900	2.856×10^{-5}
NCB20B	180	1×10^{-7}	26507	2.789×10^{-5}	11118	2.804×10^{-5}	28345	2.440×10^{-5}
NONDIA	10	1×10^{-1}	84	4.198	407	6.547	124	6.487
NONDIA	10	1×10^{-3}	144	1.406×10^{-3}	79	6.481	579	1.101×10^{-4}
NONDIA	10	1×10^{-5}	168	3.507×10^{-5}	93	6.480	349	5.342×10^{-7}
NONDIA	10	1×10^{-7}	195	2.695×10^{-7}	185	1.575×10^{-7}	322	4.429×10^{-10}
NONDIA	20	1×10^{-1}	154	2.666	814	4.149	885	4.485
NONDIA	20	1×10^{-3}	356	2.292×10^{-3}	140	4.435	744	5.560×10^{-4}
NONDIA	20	1×10^{-5}	403	3.163×10^{-5}	382	1.709×10^{-3}	996	4.738×10^{-7}
NONDIA	20	1×10^{-7}	468	1.399×10^{-6}	383	4.588×10^{-7}	1933	3.743×10^{-9}
NONDIA	30	1×10^{-1}	199	1.497	169	3.286	517	3.271
NONDIA	30	1×10^{-3}	566	6.818×10^{-3}	780	1.606×10^{-2}	1319	1.828×10^{-3}
NONDIA	30	1×10^{-5}	589	1.475×10^{-4}	689	8.898×10^{-4}	1082	5.285×10^{-6}
NONDIA	30	1×10^{-7}	634	3.771×10^{-7}	1319	6.491×10^{-7}	1551	4.326×10^{-9}

Table A.14. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
NONDIA	50	1×10^{-1}	317	8.665×10^{-1}	269	1.505	423	1.520
NONDIA	50	1×10^{-3}	635	8.814×10^{-3}	375	1.498	1659	2.467×10^{-3}
NONDIA	50	1×10^{-5}	851	7.108×10^{-5}	3599	4.640×10^{-4}	1659	6.908×10^{-6}
NONDIA	50	1×10^{-7}	1132	1.517×10^{-6}	1731	1.971×10^{-5}	1556	1.127×10^{-8}
NONDIA	90	1×10^{-1}	573	3.893×10^{-1}	1087	6.154×10^{-1}	744	5.843×10^{-1}
NONDIA	90	1×10^{-3}	1374	3.301×10^{-1}	563	5.766×10^{-1}	2052	9.232×10^{-3}
NONDIA	90	1×10^{-5}	1720	9.430×10^{-5}	564	5.781×10^{-1}	3319	2.610×10^{-5}
NONDIA	90	1×10^{-7}	1788	2.403×10^{-6}	1306	4.184×10^{-6}	4057	5.888×10^{-8}
NONDIA	100	1×10^{-1}	419	3.342×10^{-1}	418	4.850×10^{-1}	824	4.930×10^{-1}
NONDIA	100	1×10^{-3}	1610	2.086×10^{-1}	927	4.803×10^{-1}	4544	1.815×10^{-2}
NONDIA	100	1×10^{-5}	1907	1.474×10^{-4}	934	4.658×10^{-1}	4975	4.317×10^{-5}
NONDIA	100	1×10^{-7}	2235	2.091×10^{-6}	2668	5.756×10^{-5}	4072	1.010×10^{-7}
NONDQUAR	100	1×10^{-1}	828	5.744×10^{-1}	1433	7.564×10^{-1}	3532	1.006×10^{-1}
NONDQUAR	100	1×10^{-3}	1842	1.889×10^{-2}	5568	2.000×10^{-2}	7272	5.244×10^{-3}
NONDQUAR	100	1×10^{-5}	4661	1.475×10^{-3}	8339	1.946×10^{-3}	20194	3.394×10^{-4}
NONDQUAR	100	1×10^{-7}	26783	1.700×10^{-4}	21822	1.549×10^{-4}	50037*	3.497×10^{-5}
OSBORNEA‡	5	1×10^{-1}	52	1.127×10^{-2}	8	8.790×10^{-1}	170	1.485×10^{-1}
OSBORNEA‡	5	1×10^{-3}	64	5.240×10^{-2}	9	8.790×10^{-1}	496	7.589×10^{-4}
OSBORNEA‡	5	1×10^{-5}	135	3.040×10^{-3}	260	1.902×10^{-3}	1638	2.432×10^{-5}
OSBORNEA‡	5	1×10^{-7}	504	4.404×10^{-5}	677	2.252×10^{-5}	1033	2.247×10^{-5}
OSBORNEB	11	1×10^{-1}	89	5.190×10^{-1}	308	6.677×10^{-1}	1031	2.972×10^{-1}
OSBORNEB	11	1×10^{-3}	172	3.075×10^{-1}	829	3.219×10^{-1}	1238	7.480×10^{-2}
OSBORNEB	11	1×10^{-5}	1052	7.968×10^{-3}	1249	2.882×10^{-3}	2334	7.854×10^{-5}
OSBORNEB	11	1×10^{-7}	1708	8.191×10^{-6}	1555	6.681×10^{-6}	2395	9.143×10^{-9}
PENALTY1	4	1×10^{-1}	49	1.917×10^{-2}	40	8.686×10^{-3}	100	4.980×10^{-3}
PENALTY1	4	1×10^{-3}	87	2.835×10^{-5}	165	3.661×10^{-5}	95	4.498×10^{-5}
PENALTY1	4	1×10^{-5}	117	3.573×10^{-5}	55	1.741×10^{-4}	106	3.829×10^{-5}
PENALTY1	4	1×10^{-7}	113	2.254×10^{-5}	97	3.820×10^{-5}	140	3.826×10^{-5}
PENALTY1	10	1×10^{-1}	358	4.733×10^{-1}	1167	5.627×10^{-2}	284	1.327×10^{-3}
PENALTY1	10	1×10^{-3}	178	3.567×10^{-5}	347	2.207×10^{-4}	353	9.096×10^{-5}
PENALTY1	10	1×10^{-5}	194	2.463×10^{-5}	335	6.078×10^{-5}	595	6.005×10^{-5}
PENALTY1	10	1×10^{-7}	245	2.985×10^{-5}	352	5.882×10^{-5}	477	5.965×10^{-5}
PENALTY1	50	1×10^{-1}	2527	5.606×10^{-1}	2274	1.423	2076	8.180×10^{-3}
PENALTY1	50	1×10^{-3}	2455	6.140×10^{-5}	2697	1.201×10^{-4}	2281	2.999×10^{-4}
PENALTY1	50	1×10^{-5}	2298	5.576×10^{-5}	1786	9.926×10^{-5}	2384	1.322×10^{-4}
PENALTY1	50	1×10^{-7}	3919	8.208×10^{-6}	1523	8.616×10^{-5}	16116	7.338×10^{-6}
PENALTY1	100	1×10^{-1}	5649	8.758×10^{-1}	6442	6.236	4687	8.428×10^{-3}
PENALTY1	100	1×10^{-3}	6701	1.019×10^{-4}	3923	1.489×10^{-4}	5093	1.859×10^{-4}
PENALTY1	100	1×10^{-5}	5940	9.287×10^{-5}	4516	1.904×10^{-4}	5907	1.878×10^{-4}
PENALTY1	100	1×10^{-7}	10954	1.256×10^{-5}	6607	1.677×10^{-4}	28399	5.191×10^{-6}
PFIT1LS‡	3	1×10^{-1}	35	4.034	67	9.281	389	5.246
PFIT1LS‡	3	1×10^{-3}	111	9.255×10^{-1}	1415	7.002	447	2.288×10^{-4}
PFIT1LS‡	3	1×10^{-5}	325	1.154×10^{-2}	342	2.235×10^{-6}	502	2.581×10^{-6}
PFIT1LS‡	3	1×10^{-7}	482	2.955×10^{-4}	258	4.493×10^{-5}	453	2.489×10^{-6}
PFIT2LS‡	3	1×10^{-1}	113	5.174×10^{1}	81	9.874×10^{1}	620	2.712×10^{-1}
PFIT2LS‡	3	1×10^{-3}	410	2.136	123	1.245×10^{2}	537	6.269×10^{-3}
PFIT2LS‡	3	1×10^{-5}	618	5.570×10^{-2}	430	4.757×10^{-3}	537	1.772×10^{-3}
PFIT2LS‡	3	1×10^{-7}	706	1.251×10^{-2}	340	2.704×10^{-3}	870	1.648×10^{-3}

Table A.15. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
PFIT3LS‡	3	1×10^{-1}	162	1.868×10^{2}	390	7.944×10^{1}	817	3.682×10^{-1}
PFIT3LS‡	3	1×10^{-3}	546	7.561	640	9.159×10^{-1}	612	3.276×10^{-2}
PFIT3LS‡	3	1×10^{-5}	1012	9.384×10^{-2}	615	4.587×10^{-2}	702	3.154×10^{-2}
PFIT3LS‡	3	1×10^{-7}	1012	8.229×10^{-2}	444	4.374×10^{-2}	1504*	2.782×10^{-2}
PFIT4LS‡	3	1×10^{-1}	234	2.464×10^{2}	78	2.144×10^{3}	723	1.842×10^{-1}
PFIT4LS‡	3	1×10^{-3}	612	2.736×10^{1}	758	6.570	715	1.234×10^{-1}
PFIT4LS‡	3	1×10^{-5}	1181	8.871×10^{-1}	471	1.677×10^{-1}	1506*	1.433×10^{-1}
PFIT4LS‡	3	1×10^{-7}	1374	2.619×10^{-1}	597	1.249×10^{-1}	1502*	1.086×10^{-1}
QUARTC	25	1×10^{-1}	599	4.172×10^{-1}	1716	2.004	1203	1.376×10^{-2}
QUARTC	25	1×10^{-3}	759	7.392×10^{-3}	767	3.811×10^{-2}	1496	8.968×10^{-5}
QUARTC	25	1×10^{-5}	828	6.201×10^{-6}	1245	8.620×10^{-5}	1750	3.508×10^{-7}
QUARTC	25	1×10^{-7}	1040	4.136×10^{-7}	1584	1.148×10^{-6}	2807	1.313×10^{-8}
QUARTC	100	1×10^{-1}	4920	3.885×10^{-1}	4766	6.422	7146	4.248×10^{-2}
QUARTC	100	1×10^{-3}	5612	9.569×10^{-3}	4517	2.474×10^{-3}	5960	1.807×10^{-4}
QUARTC	100	1×10^{-5}	6494	5.767×10^{-5}	6534	7.941×10^{-4}	7146	2.060×10^{-6}
QUARTC	100	1×10^{-7}	6913	8.633×10^{-7}	1119	1.906×10^{7}	7344	2.041×10^{-8}
SINEVAL	2	1×10^{-1}	12	5.231	4	5.552	108	4.632
SINEVAL	2	1×10^{-3}	152	9.339×10^{-1}	194	3.587	700	2.082×10^{-4}
SINEVAL	2	1×10^{-5}	226	1.682×10^{-5}	450	1.856×10^{-1}	488	1.360×10^{-7}
SINEVAL	2	1×10^{-7}	295	1.841×10^{-8}	425	1.955×10^{-5}	563	9.899×10^{-13}
SINQUAD	5	1×10^{-1}	36	1.503	333	5.219	107	1.121×10^{-2}
SINQUAD	5	1×10^{-3}	98	8.682×10^{-3}	79	6.186×10^{-3}	121	3.326×10^{-5}
SINQUAD	5	1×10^{-5}	140	1.602×10^{-5}	77	3.260×10^{-5}	121	9.620×10^{-8}
SINQUAD	5	1×10^{-7}	170	1.177×10^{-6}	135	5.826×10^{-9}	205	-3.938×10^{-10}
SINQUAD	50	1×10^{-1}	1724	1.740	742	2.222×10^{1}	3929	1.376×10^{-1}
SINQUAD	50	1×10^{-3}	3724	3.119×10^{-2}	1101	9.615×10^{-3}	3616	2.544×10^{-4}
SINQUAD	50	1×10^{-5}	4234	3.236×10^{-4}	849	2.589×10^{-4}	2857	6.467×10^{-7}
SINQUAD	50	1×10^{-7}	6464	1.670×10^{-6}	1261	1.644×10^{-6}	2083	5.443×10^{-9}
SINQUAD	100	1×10^{-1}	4978	4.096	1247	1.048×10^{1}	3062	1.026×10^{-1}
SINQUAD	100	1×10^{-3}	7193	2.876×10^{-2}	2470	6.295×10^{-2}	3265	2.132×10^{-4}
SINQUAD	100	1×10^{-5}	10426	3.206×10^{-4}	1948	9.257×10^{-4}	3468	5.694×10^{-7}
SINQUAD	100	1×10^{-7}	15008	2.263×10^{-6}	2152	3.046×10^{-6}	4079	3.929×10^{-9}
SISSER	2	1×10^{-1}	4**	3.000×10^{-4}	20	2.326×10^{-2}	65	2.424×10^{-4}
SISSER	2	1×10^{-3}	15	1.764×10^{-4}	139	2.826×10^{-5}	204	1.415×10^{-4}
SISSER	2	1×10^{-5}	20	2.270×10^{-5}	44	3.731×10^{-5}	85	2.188×10^{-9}
SISSER	2	1×10^{-7}	27	1.467×10^{-8}	71	4.270×10^{-7}	116	6.695×10^{-12}
SPARSQUR	10	1×10^{-1}	42	2.779×10^{-1}	104	4.304×10^{-2}	403	5.561×10^{-3}
SPARSQUR	10	1×10^{-3}	85	2.675×10^{-3}	212	6.281×10^{-3}	384	6.274×10^{-5}
SPARSQUR	10	1×10^{-5}	184	4.915×10^{-5}	740	2.770×10^{-5}	384	3.317×10^{-7}
SPARSQUR	10	1×10^{-7}	241	1.377×10^{-6}	398	1.386×10^{-7}	579	1.307×10^{-8}
SPARSQUR	50	1×10^{-1}	1070	7.980×10^{-1}	1303	6.397×10^{-1}	1316	1.918×10^{-2}
SPARSQUR	50	1×10^{-3}	1034	4.918×10^{-3}	1078	1.952×10^{-2}	3577	9.425×10^{-5}
SPARSQUR	50	1×10^{-5}	1472	2.315×10^{-5}	2755	4.135×10^{-5}	2651	5.338×10^{-7}
SPARSQUR	50	1×10^{-7}	1847	3.804×10^{-7}	1706	5.903×10^{-6}	2651	6.858×10^{-9}
SPARSQUR	100	1×10^{-1}	2912	5.988×10^{-1}	3387	1.953	5960	1.100×10^{-2}
SPARSQUR	100	1×10^{-3}	2484	4.672×10^{-3}	2435	5.029×10^{-2}	3677	5.412×10^{-5}
SPARSQUR	100	1×10^{-5}	2664	7.151×10^{-5}	5794	1.737×10^{-4}	4292	6.552×10^{-6}
SPARSQUR	100	1×10^{-7}	3757	3.002×10^{-7}	4591	2.610×10^{-6}	5960	1.082×10^{-8}

Table A.16. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
TOINTGSS	10	1×10^{-1}	65	4.850×10^{-1}	759	2.072	101	1.831×10^{-2}
TOINTGSS	10	1×10^{-3}	164	1.448×10^{-2}	301	3.304×10^{-2}	165	8.167×10^{-6}
TOINTGSS	10	1×10^{-5}	211	2.845×10^{-5}	124	3.624×10^{-4}	200	2.744×10^{-8}
TOINTGSS	10	1×10^{-7}	277	2.113×10^{-7}	124	3.539×10^{-6}	200	5.802×10^{-11}
TOINTGSS	50	1×10^{-1}	388	1.033	563	1.183×10^{1}	724	6.155×10^{-2}
TOINTGSS	50	1×10^{-3}	878	1.276×10^{-2}	415	4.651×10^{-2}	2132	1.554×10^{-5}
TOINTGSS	50	1×10^{-5}	923	1.748×10^{-4}	693	1.140×10^{-3}	724	5.405×10^{-8}
TOINTGSS	50	1×10^{-7}	1574	1.030×10^{-6}	693	1.136×10^{-5}	724	-2.877×10^{-9}
TOINTGSS	100	1×10^{-1}	1638	1.693	590	7.090	2660	6.483×10^{-2}
TOINTGSS	100	1×10^{-3}	1521	2.165×10^{-2}	509	7.545×10^{-2}	1646	3.228×10^{-5}
TOINTGSS	100	1×10^{-5}	2692	2.385×10^{-4}	508	1.898×10^{-3}	2526	2.620×10^{-8}
TOINTGSS	100	1×10^{-7}	2312	3.442×10^{-6}	508	1.893×10^{-5}	2526	4.046×10^{-9}
TQUARTIC	5	1×10^{-1}	22	4.614×10^{-1}	24	6.142×10^{-1}	40	4.540×10^{-1}
TQUARTIC	5	1×10^{-3}	63	5.640×10^{-3}	242	9.902×10^{-2}	234	5.290×10^{-4}
TQUARTIC	5	1×10^{-5}	107	1.155×10^{-5}	202	8.160×10^{-4}	818	1.004×10^{-6}
TQUARTIC	5	1×10^{-7}	137	7.045×10^{-8}	95	1.098×10^{-6}	163	9.192×10^{-9}
TQUARTIC	10	1×10^{-1}	44	6.151×10^{-1}	274	7.097×10^{-1}	92	5.464×10^{-1}
TQUARTIC	10	1×10^{-3}	174	4.406×10^{-2}	543	2.485×10^{-1}	344	5.223×10^{-3}
TQUARTIC	10	1×10^{-5}	277	2.548×10^{-5}	466	4.217×10^{-2}	722	7.389×10^{-6}
TQUARTIC	10	1×10^{-7}	388	8.693×10^{-7}	805	6.160×10^{-5}	535	4.256×10^{-8}
TQUARTIC	50	1×10^{-1}	191	6.732×10^{-1}	170	7.480×10^{-1}	310	7.002×10^{-1}
TQUARTIC	50	1×10^{-3}	1487	3.648×10^{-1}	1155	5.611×10^{-1}	1534	6.796×10^{-1}
TQUARTIC	50	1×10^{-5}	3112	6.435×10^{-3}	1441	2.121×10^{-2}	2117	1.390×10^{-4}
TQUARTIC	50	1×10^{-7}	5704	8.027×10^{-5}	3736	3.333×10^{-4}	1785	3.649×10^{-7}
TQUARTIC	100	1×10^{-1}	226	7.236×10^{-1}	236	7.828×10^{-1}	610	7.423×10^{-1}
TQUARTIC	100	1×10^{-3}	1085	7.089×10^{-1}	790	7.232×10^{-1}	3629	7.219×10^{-1}
TQUARTIC	100	1×10^{-5}	26574	2.458×10^{-2}	2675	1.765×10^{-1}	4298	5.460×10^{-4}
TQUARTIC	100	1×10^{-7}	31937	2.808×10^{-4}	4860	4.761×10^{-4}	5838	2.894×10^{-7}
TRIDIA	10	1×10^{-1}	128	1.555	124	1.392	594	3.847×10^{-4}
TRIDIA	10	1×10^{-3}	174	1.649×10^{-3}	305	2.065×10^{-2}	495	1.429×10^{-6}
TRIDIA	10	1×10^{-5}	176	2.353×10^{-5}	242	3.449×10^{-4}	698	4.786×10^{-9}
TRIDIA	10	1×10^{-7}	226	1.205×10^{-7}	279	3.036×10^{-6}	848	2.043×10^{-12}
TRIDIA	20	1×10^{-1}	301	7.391×10^{-1}	812	1.072	2309	4.788×10^{-4}
TRIDIA	20	1×10^{-3}	312	9.505×10^{-3}	443	1.352×10^{-1}	1432	1.038×10^{-6}
TRIDIA	20	1×10^{-5}	396	4.890×10^{-5}	821	4.616×10^{-4}	2029	3.474×10^{-9}
TRIDIA	20	1×10^{-7}	500	5.289×10^{-7}	864	3.588×10^{-6}	1992	7.669×10^{-12}
TRIDIA	30	1×10^{-1}	454	8.863×10^{-1}	3249	7.566	2694	2.364×10^{-3}
TRIDIA	30	1×10^{-3}	599	7.428×10^{-3}	2486	6.494×10^{-2}	2403	4.586×10^{-5}
TRIDIA	30	1×10^{-5}	753	1.083×10^{-4}	1254	1.150×10^{-3}	3111	7.820×10^{-7}
TRIDIA	30	1×10^{-7}	849	7.455×10^{-7}	2292	5.997×10^{-6}	4598	6.846×10^{-11}
TRIDIA	50	1×10^{-1}	670	1.625	1604	8.841	3723	1.190×10^{-1}
TRIDIA	50	1×10^{-3}	1124	1.616×10^{-2}	1824	2.452×10^{-1}	6195	7.991×10^{-6}
TRIDIA	50	1×10^{-5}	1056	4.101×10^{-4}	3602	1.237×10^{-3}	5985	2.190×10^{-6}
TRIDIA	50	1×10^{-7}	1784	1.022×10^{-6}	3450	2.572×10^{-5}	7640	1.634×10^{-8}
TRIDIA	100	1×10^{-1}	2183	5.214	4778	2.202×10^{1}	9774	1.524×10^{-1}
TRIDIA	100	1×10^{-3}	4301	3.587×10^{-2}	3474	6.746×10^{-1}	15035	9.421×10^{-5}
TRIDIA	100	1×10^{-5}	4682	4.705×10^{-4}	10097	2.340×10^{-3}	20327	4.890×10^{-7}
TRIDIA	100	1×10^{-7}	3650	4.093×10^{-6}	11716	3.468×10^{-5}	22968	8.918×10^{-9}

Table A.17. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

			NEWUOA		FD L-BFGS		CD L-BFGS	
Problem	n	σ_f	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$	#feval	$\phi(x)-\phi^*$
WATSON	12	1×10^{-1}	91	1.602×10^{-1}	260	2.747×10^{-1}	407	9.362×10^{-2}
WATSON	12	1×10^{-3}	170	7.112×10^{-2}	752	7.265×10^{-3}	737	8.831×10^{-3}
WATSON	12	1×10^{-5}	708	6.531×10^{-4}	1487	2.403×10^{-3}	1470	1.808×10^{-4}
WATSON	12	1×10^{-7}	1664	2.355×10^{-4}	1077	1.868×10^{-4}	1559	1.367×10^{-5}
WATSON	31	1×10^{-1}	283	1.444	567	2.596	1379	2.727×10^{-1}
WATSON	31	1×10^{-3}	982	1.556×10^{-1}	1465	1.648×10^{-1}	4316	6.367×10^{-3}
WATSON	31	1×10^{-5}	4133	4.936×10^{-3}	2191	1.682×10^{-2}	3826	1.420×10^{-3}
WATSON	31	1×10^{-7}	13951	1.468×10^{-4}	4582	8.715×10^{-4}	8624	1.013×10^{-4}
WOODS	4	1×10^{-1}	86	8.655	378	5.200×10^{-1}	391	2.498×10^{-1}
WOODS	4	1×10^{-3}	87	7.873	300	1.818×10^{-1}	567	1.248×10^{-5}
WOODS	4	1×10^{-5}	432	1.680×10^{-5}	786	1.295×10^{-3}	513	6.023×10^{-7}
WOODS	4	1×10^{-7}	515	4.005×10^{-7}	278	3.224×10^{-6}	391	6.204×10^{-10}
WOODS	100	1×10^{-1}	8056	1.617×10^{2}	5272	4.980×10^{1}	5925	6.411
WOODS	100	1×10^{-3}	27605	4.700×10^{-1}	9371	4.582×10^{-1}	7344	1.666×10^{-3}
WOODS	100	1×10^{-5}	35366	1.415×10^{-2}	4207	4.455×10^{-3}	9097	9.510×10^{-6}
WOODS	100	1×10^{-7}	50000*	6.216×10^{-2}	4756	7.326×10^{-5}	7528	3.725×10^{-7}
ZANGWIL2	2	1×10^{-1}	7	1.455×10^{-3}	14	2.786×10^{-2}	31	1.009×10^{-3}
ZANGWIL2	2	1×10^{-3}	17	1.572×10^{-5}	13	1.557×10^{-3}	31	1.987×10^{-6}
ZANGWIL2	2	1×10^{-5}	21	6.543×10^{-8}	13	1.164×10^{-6}	516	3.655×10^{-9}
ZANGWIL2	2	1×10^{-7}	16	3.964×10^{-10}	13	1.167×10^{-8}	516	-9.150×10^{-11}

Table A.18. Noisy Unconstrained `CUTEst` Problems Tested. n is the number of variables. σ_f is the standard deviation of the noise.

the number of function evaluations. We removed the limit on the number of evaluations for COBYLA to see if it converges to a different local solution. For HS67, COBYLA terminates with a final objective function value of -1116.415 after 32606 evaluations, and for CRESC4 it converges to a feasible solution with $f = 1.03523$ after 4706634 evaluations.

(iii) One of the solvers terminated at an infeasible point. There are seven problems for which KNITRO terminated at a feasible point but COBYLA at an infeasible one. For all of these problems, COBYLA hits the limit on the number of function evaluations; it can make further progress in feasibility if the budget of evaluations is increased. For problems HS101, HS102, and HS103, KNITRO also hits the evaluation limit; however, it gets a better solution –a feasible one with a lower objective value– for all three problems.

		LMDER		DFOLS	
Problem	(n, m)	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
LINEAR(FR)	$(9, 45)$	21	1.421×10^{-14}	45	-1.421×10^{-14}
LINEAR(FR)	$(9, 45)$	21	1.279×10^{-13}	49	0.000
LINEAR(R1)	$(7, 35)$	16	-3.099×10^{-7}	61	-3.099×10^{-7}
LINEAR(R1)	$(7, 35)$	16	-3.099×10^{-7}	56	-3.099×10^{-7}
LINEAR(R10RC)	$(7, 35)$	16	1.493×10^{-8}	56	1.493×10^{-8}
LINEAR(R10RC)	$(7, 35)$	16	1.493×10^{-8}	62	1.493×10^{-8}
ROSENBR	$(2, 2)$	39	0.000	50	3.788×10^{-24}
ROSENBR	$(2, 2)$	17	0.000	14	1.498×10^{-20}
HELIX	$(3, 3)$	49	3.941×10^{-59}	44	2.532×10^{-28}
HELIX	$(3, 3)$	57	5.921×10^{-47}	80	5.476×10^{-20}
POWELLSG	$(4, 4)$	331	6.610×10^{-35}	54	1.074×10^{-18}
POWELLSG	$(4, 4)$	346	7.106×10^{-35}	47	1.467×10^{-14}
FREUROTH	$(2, 2)$	60	3.688×10^{-6}	122	3.679×10^{-6}
FREUROTH	$(2, 2)$	87	3.793×10^{-6}	107	3.679×10^{-6}
BARD	$(3, 15)$	21	3.066×10^{-10}	32	3.066×10^{-10}
BARD	$(3, 15)$	169	1.742×10^{1}	107	1.066×10^{-1}
KOWOSB	$(4, 11)$	99	4.429×10^{-12}	77	3.849×10^{-12}
MEYER	$(3, 16)$	456	-4.829×10^{-6}	1500^*	3.715×10^{4}
WATSON	$(6, 31)$	57	5.359×10^{-11}	81	5.355×10^{-11}
WATSON	$(6, 31)$	85	5.357×10^{-11}	105	5.355×10^{-11}
WATSON	$(9, 31)$	144	1.393×10^{-13}	409	1.468×10^{-13}
WATSON	$(9, 31)$	134	1.382×10^{-13}	499	1.454×10^{-13}
WATSON	$(12, 31)$	276	1.198×10^{-15}	340	2.214×10^{-9}
WATSON	$(12, 31)$	181	3.636×10^{-16}	1013	2.246×10^{-9}
BOX3D	$(3, 10)$	25	3.390×10^{-32}	52	1.921×10^{-27}
JENSMP	$(2, 10)$	46	-1.764×10^{-5}	70	-1.764×10^{-5}
BROWNDEN	$(4, 20)$	94	1.684×10^{-3}	167	1.626×10^{-3}
BROWNDEN	$(4, 20)$	96	1.733×10^{-3}	189	1.626×10^{-3}
CHEBYQUAD	$(6, 6)$	59	4.434×10^{-32}	54	1.239×10^{-23}
CHEBYQUAD	$(7, 7)$	58	6.915×10^{-32}	49	5.120×10^{-29}
CHEBYQUAD	$(8, 8)$	168	-2.725×10^{-10}	258	-2.743×10^{-10}
CHEBYQUAD	$(9, 9)$	94	4.143×10^{-32}	87	4.535×10^{-28}
CHEBYQUAD	$(10, 10)$	108	1.731×10^{-3}	191	-3.036×10^{-10}
CHEBYQUAD	$(11, 11)$	320	-4.428×10^{-10}	352	-4.481×10^{-10}
BROWNAL	$(10, 10)$	79	2.840×10^{-29}	52	7.676×10^{-19}
OSBORNE1	$(5, 33)$	157	-3.025×10^{-12}	783	3.737×10^{-12}
OSBORNE2	$(11, 65)$	101	-3.705×10^{-9}	150	-3.706×10^{-9}
OSBORNE2	$(11, 65)$	148	1.750	138	1.750
BDQRTIC	$(8, 8)$	508	3.759×10^{-6}	336	8.111×10^{-6}
BDQRTIC	$(10, 12)$	662	3.152×10^{-6}	457	3.874×10^{-6}
BDQRTIC	$(11, 14)$	1046	3.296×10^{-6}	554	2.046×10^{-5}
BDQRTIC	$(12, 16)$	1119	-1.268×10^{-6}	561	5.775×10^{-5}

Table A.19. Benchmarking Problems Tested. n is the number of variables and m is the dimension of the residual vector.

		LMDER		DFOLS	
Problem	(n, m)	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
CUBE	$(5, 5)$	388	0.000	295	3.119×10^{-18}
CUBE	$(6, 6)$	1024	0.000	714	4.127×10^{-24}
CUBE	$(8, 8)$	4008^*	2.090×10^{-8}	4000^*	4.753×10^{-11}
MANCINO	$(5, 5)$	19	4.224×10^{-22}	25	6.176×10^{-20}
MANCINO	$(5, 5)$	25	4.686×10^{-22}	27	8.536×10^{-18}
MANCINO	$(8, 8)$	28	5.795×10^{-22}	23	3.152×10^{-12}
MANCINO	$(10, 10)$	34	8.218×10^{-22}	25	3.069×10^{-11}
MANCINO	$(12, 12)$	53	1.322×10^{-22}	32	2.454×10^{-17}
MANCINO	$(12, 12)$	53	5.201×10^{-22}	35	2.798×10^{-9}
HEART8LS	$(8, 8)$	37	3.402×10^{-30}	46	1.082×10^{-23}
HEART8LS	$(8, 8)$	109	3.402×10^{-30}	2293	1.001×10^{-10}

Table A.20. Benchmarking Problems Tested. n is the number of variables and m is the dimension of the residual vector.

For two problems, COBYLA terminated at a feasible point but KNITRO at an infeasible one. For problem POLAK6, KNITRO stalls at an infeasible point. For SPIRAL, it gets close to feasibility but runs out its evaluation budget before it can reduce the feasibility error further.

(iv) Both solvers terminated at infeasible points. For all three of those problems, KNITRO declares convergence to infeasible stationary points. For problems S365 and BURKEHAN, COBYLA terminates due to the condition on the final trust region radius whereas for POLAK2 it stops due to the function evaluation limit.

			LMDER		DFOLS	
Problem	(n, m)	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
LINEAR(FR)	$(9, 45)$	1×10^{-1}	65	3.565×10^{1}	70	3.641×10^{1}
LINEAR(FR)	$(9, 45)$	1×10^{-3}	126	5.008×10^{-2}	97	6.606×10^{-2}
LINEAR(FR)	$(9, 45)$	1×10^{-5}	115	5.170×10^{-4}	81	2.889×10^{-5}
LINEAR(FR)	$(9, 45)$	1×10^{-7}	114	5.523×10^{-6}	76	2.296×10^{-6}
LINEAR(FR)	$(9, 45)$	1×10^{-1}	229	2.822×10^{2}	77	1.112×10^{3}
LINEAR(FR)	$(9, 45)$	1×10^{-3}	115	6.086×10^{-2}	89	2.179×10^{-2}
LINEAR(FR)	$(9, 45)$	1×10^{-5}	115	5.482×10^{-4}	78	2.305×10^{-4}
LINEAR(FR)	$(9, 45)$	1×10^{-7}	114	5.546×10^{-6}	76	2.240×10^{-6}
LINEAR(R1)	$(7, 35)$	1×10^{-1}	69	6.470×10^{-2}	74	3.081
LINEAR(R1)	$(7, 35)$	1×10^{-3}	65	3.221×10^{-3}	84	1.482×10^{-4}
LINEAR(R1)	$(7, 35)$	1×10^{-5}	70	-2.904×10^{-7}	74	-2.969×10^{-7}
LINEAR(R1)	$(7, 35)$	1×10^{-7}	108	-3.099×10^{-7}	73	-3.098×10^{-7}
LINEAR(R1)	$(7, 35)$	1×10^{-1}	68	6.471×10^{-2}	78	2.155
LINEAR(R1)	$(7, 35)$	1×10^{-3}	118	7.437×10^{-5}	86	1.577×10^{-4}
LINEAR(R1)	$(7, 35)$	1×10^{-5}	106	-3.049×10^{-7}	84	-2.926×10^{-7}
LINEAR(R1)	$(7, 35)$	1×10^{-7}	103	-3.099×10^{-7}	79	-3.099×10^{-7}
LINEAR(R10RC)	$(7, 35)$	1×10^{-1}	63	3.666×10^{-1}	71	8.280×10^{-1}
LINEAR(R10RC)	$(7, 35)$	1×10^{-3}	82	7.472×10^{-6}	82	2.771×10^{-4}
LINEAR(R10RC)	$(7, 35)$	1×10^{-5}	106	2.101×10^{-8}	75	2.978×10^{-8}
LINEAR(R10RC)	$(7, 35)$	1×10^{-7}	72	1.493×10^{-8}	73	1.493×10^{-8}
LINEAR(R10RC)	$(7, 35)$	1×10^{-1}	62	3.666×10^{-1}	79	9.209×10^{-1}
LINEAR(R10RC)	$(7, 35)$	1×10^{-3}	67	1.272×10^{-3}	89	2.258×10^{-4}
LINEAR(R10RC)	$(7, 35)$	1×10^{-5}	106	2.103×10^{-8}	82	1.660×10^{-8}
LINEAR(R10RC)	$(7, 35)$	1×10^{-7}	109	1.495×10^{-8}	75	1.985×10^{-8}
ROSENBR	$(2, 2)$	1×10^{-1}	44	7.600×10^{-2}	51	4.584×10^{-1}
ROSENBR	$(2, 2)$	1×10^{-3}	77	3.078×10^{-7}	50	6.040×10^{-8}
ROSENBR	$(2, 2)$	1×10^{-5}	67	4.501×10^{-10}	55	1.122×10^{-10}
ROSENBR	$(2, 2)$	1×10^{-7}	65	3.663×10^{-14}	49	2.148×10^{-14}
ROSENBR	$(2, 2)$	1×10^{-1}	105	4.118×10^{-3}	71	3.578×10^{-2}
ROSENBR	$(2, 2)$	1×10^{-3}	49	7.177×10^{-7}	35	6.097×10^{-11}
ROSENBR	$(2, 2)$	1×10^{-5}	44	4.539×10^{-10}	54	3.365×10^{-10}
ROSENBR	$(2, 2)$	1×10^{-7}	29	3.064×10^{-14}	28	2.191×10^{-15}
HELIX	$(3, 3)$	1×10^{-1}	61	4.158×10^{-2}	59	5.023×10^{-2}
HELIX	$(3, 3)$	1×10^{-3}	61	1.328×10^{-6}	62	1.265×10^{-5}
HELIX	$(3, 3)$	1×10^{-5}	71	7.012×10^{-10}	66	7.646×10^{-10}
HELIX	$(3, 3)$	1×10^{-7}	70	4.893×10^{-14}	77	2.031×10^{-14}
HELIX	$(3, 3)$	1×10^{-1}	61	1.767	66	8.701×10^{-3}
HELIX	$(3, 3)$	1×10^{-3}	75	5.707×10^{-6}	76	1.429×10^{-6}
HELIX	$(3, 3)$	1×10^{-5}	73	4.865×10^{-10}	74	8.032×10^{-10}
HELIX	$(3, 3)$	1×10^{-7}	70	5.016×10^{-14}	47	3.085×10^{-16}
POWELLSG	$(4, 4)$	1×10^{-1}	78	6.286×10^{-2}	62	9.436×10^{-2}
POWELLSG	$(4, 4)$	1×10^{-3}	82	3.495×10^{-6}	54	7.641×10^{-5}
POWELLSG	$(4, 4)$	1×10^{-5}	86	3.364×10^{-9}	54	7.255×10^{-10}
POWELLSG	$(4, 4)$	1×10^{-7}	131	2.803×10^{-14}	55	4.174×10^{-15}
POWELLSG	$(4, 4)$	1×10^{-1}	79	6.338×10^{-2}	59	2.036×10^{-3}
POWELLSG	$(4, 4)$	1×10^{-3}	86	3.404×10^{-5}	64	2.126×10^{-7}
POWELLSG	$(4, 4)$	1×10^{-5}	131	2.607×10^{-10}	61	1.502×10^{-8}
POWELLSG	$(4, 4)$	1×10^{-7}	121	1.694×10^{-13}	56	2.861×10^{-15}

Table A.21. Benchmarking Problems Tested. n is the number of variables and m is the dimension of the residual vector.

			LMDER		DFOLS	
Problem	(n, m)	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
FREUROTH	$(2, 2)$	1×10^{-1}	42	1.724	68	4.634×10^{-2}
FREUROTH	$(2, 2)$	1×10^{-3}	48	7.582×10^{-3}	75	4.555×10^{-6}
FREUROTH	$(2, 2)$	1×10^{-5}	59	7.704×10^{-5}	113	3.717×10^{-6}
FREUROTH	$(2, 2)$	1×10^{-7}	64	4.523×10^{-6}	91	3.679×10^{-6}
FREUROTH	$(2, 2)$	1×10^{-1}	55	2.546	83	1.310×10^{-2}
FREUROTH	$(2, 2)$	1×10^{-3}	66	1.265×10^{-2}	78	1.331×10^{-5}
FREUROTH	$(2, 2)$	1×10^{-5}	80	4.987×10^{-5}	102	3.681×10^{-6}
FREUROTH	$(2, 2)$	1×10^{-7}	91	7.027×10^{-6}	122	3.679×10^{-6}
BARD	$(3, 15)$	1×10^{-1}	45	2.370×10^{-1}	46	5.987×10^{-2}
BARD	$(3, 15)$	1×10^{-3}	41	1.168×10^{-5}	38	4.673×10^{-5}
BARD	$(3, 15)$	1×10^{-5}	53	4.713×10^{-8}	34	1.404×10^{-9}
BARD	$(3, 15)$	1×10^{-7}	44	8.436×10^{-10}	35	3.068×10^{-10}
BARD	$(3, 15)$	1×10^{-1}	41	1.355×10^{1}	56	9.256×10^{-2}
BARD	$(3, 15)$	1×10^{-3}	73	2.035×10^{1}	65	4.982×10^{-5}
BARD	$(3, 15)$	1×10^{-5}	98	2.282	54	1.114×10^{-9}
BARD	$(3, 15)$	1×10^{-7}	115	1.611×10^{1}	102	1.066×10^{-1}
KOWOSB	$(4, 11)$	1×10^{-1}	35	5.005×10^{-3}	42	3.088×10^{-2}
KOWOSB	$(4, 11)$	1×10^{-3}	57	1.418×10^{-4}	44	1.561×10^{-4}
KOWOSB	$(4, 11)$	1×10^{-5}	92	4.081×10^{-8}	60	4.916×10^{-8}
KOWOSB	$(4, 11)$	1×10^{-7}	84	1.302×10^{-8}	64	2.981×10^{-10}
MEYER	$(3, 16)$	1×10^{-1}	182	8.301×10^{4}	67	1.113×10^{5}
MEYER	$(3, 16)$	1×10^{-3}	666	4.770	252	7.380×10^{4}
MEYER	$(3, 16)$	1×10^{-5}	504	8.423×10^{-3}	1498	4.022×10^{4}
MEYER	$(3, 16)$	1×10^{-7}	497	-1.300×10^{-6}	1500^*	4.031×10^{4}
WATSON	$(6, 31)$	1×10^{-1}	96	1.592×10^{-1}	71	1.232×10^{-1}
WATSON	$(6, 31)$	1×10^{-3}	83	1.818×10^{-4}	79	1.328×10^{-4}
WATSON	$(6, 31)$	1×10^{-5}	82	6.905×10^{-5}	82	7.375×10^{-7}
WATSON	$(6, 31)$	1×10^{-7}	103	5.333×10^{-9}	78	8.703×10^{-10}
WATSON	$(6, 31)$	1×10^{-1}	95	2.562×10^{-1}	80	1.150×10^{-1}
WATSON	$(6, 31)$	1×10^{-3}	236	4.267×10^{-4}	82	2.052×10^{-4}
WATSON	$(6, 31)$	1×10^{-5}	116	5.311×10^{-5}	87	1.676×10^{-7}
WATSON	$(6, 31)$	1×10^{-7}	116	2.197×10^{-8}	85	6.039×10^{-10}
WATSON	$(9, 31)$	1×10^{-1}	95	1.791×10^{-1}	95	6.326×10^{-2}
WATSON	$(9, 31)$	1×10^{-3}	183	2.761×10^{-4}	113	1.112×10^{-4}
WATSON	$(9, 31)$	1×10^{-5}	386	3.603×10^{-5}	114	5.717×10^{-6}
WATSON	$(9, 31)$	1×10^{-7}	531	1.175×10^{-7}	220	9.384×10^{-10}
WATSON	$(9, 31)$	1×10^{-1}	134	1.338	128	3.406×10^{-2}
WATSON	$(9, 31)$	1×10^{-3}	390	7.293×10^{-3}	115	2.543×10^{-4}
WATSON	$(9, 31)$	1×10^{-5}	580	1.436×10^{-4}	113	4.061×10^{-6}
WATSON	$(9, 31)$	1×10^{-7}	533	4.147×10^{-7}	337	5.464×10^{-9}
WATSON	$(12, 31)$	1×10^{-1}	147	1.829×10^{-1}	133	4.518×10^{-1}
WATSON	$(12, 31)$	1×10^{-3}	199	1.942×10^{-4}	129	4.431×10^{-4}
WATSON	$(12, 31)$	1×10^{-5}	736	1.863×10^{-5}	130	1.202×10^{-7}
WATSON	$(12, 31)$	1×10^{-7}	713	4.164×10^{-8}	234	1.337×10^{-8}
WATSON	$(12, 31)$	1×10^{-1}	188	7.230×10^{-1}	127	1.600×10^{-1}
WATSON	$(12, 31)$	1×10^{-3}	255	1.086×10^{-2}	139	2.802×10^{-4}
WATSON	$(12, 31)$	1×10^{-5}	530	3.237×10^{-2}	196	1.911×10^{-8}
WATSON	$(12, 31)$	1×10^{-7}	851	1.854×10^{-4}	183	1.560×10^{-8}

Table A.22. Benchmarking Problems Tested. n is the number of variables and m is the dimension of the residual vector.

			LMDER		DFOLS	
Problem	(n, m)	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
BOX3D	$(3, 10)$	1×10^{-1}	44	1.674×10^{-1}	53	1.181×10^{-1}
BOX3D	$(3, 10)$	1×10^{-3}	63	1.532×10^{-3}	43	5.394×10^{-6}
BOX3D	$(3, 10)$	1×10^{-5}	56	1.738×10^{-9}	46	5.818×10^{-12}
BOX3D	$(3, 10)$	1×10^{-7}	55	1.758×10^{-13}	57	2.248×10^{-13}
JENSMP	$(2, 10)$	1×10^{-1}	46	3.728×10^{3}	47	2.684×10^{-1}
JENSMP	$(2, 10)$	1×10^{-3}	44	4.102×10^{-2}	51	1.660×10^{-2}
JENSMP	$(2, 10)$	1×10^{-5}	53	1.445×10^{-3}	59	6.415×10^{-5}
JENSMP	$(2, 10)$	1×10^{-7}	51	-7.215×10^{-6}	47	-8.720×10^{-6}
BROWNDEN	$(4, 20)$	1×10^{-1}	75	1.521×10^{3}	84	3.922×10^{2}
BROWNDEN	$(4, 20)$	1×10^{-3}	86	3.277×10^{1}	97	2.897
BROWNDEN	$(4, 20)$	1×10^{-5}	98	7.498×10^{-2}	140	9.534×10^{-3}
BROWNDEN	$(4, 20)$	1×10^{-7}	105	2.529×10^{-3}	144	4.056×10^{-3}
BROWNDEN	$(4, 20)$	1×10^{-1}	90	2.384×10^{3}	102	8.007×10^{2}
BROWNDEN	$(4, 20)$	1×10^{-3}	102	4.493×10^{1}	97	3.732
BROWNDEN	$(4, 20)$	1×10^{-5}	117	4.912×10^{-2}	115	1.440×10^{-1}
BROWNDEN	$(4, 20)$	1×10^{-7}	114	2.743×10^{-3}	130	2.798×10^{-3}
CHEBYQUAD	$(6, 6)$	1×10^{-1}	54	4.243×10^{-2}	64	2.304×10^{-2}
CHEBYQUAD	$(6, 6)$	1×10^{-3}	57	3.020×10^{-2}	73	9.161×10^{-7}
CHEBYQUAD	$(6, 6)$	1×10^{-5}	93	4.322×10^{-10}	61	1.447×10^{-11}
CHEBYQUAD	$(6, 6)$	1×10^{-7}	91	4.785×10^{-14}	63	1.792×10^{-13}
CHEBYQUAD	$(7, 7)$	1×10^{-1}	46	3.377×10^{-2}	67	1.045×10^{-2}
CHEBYQUAD	$(7, 7)$	1×10^{-3}	118	6.182×10^{-6}	64	1.901×10^{-5}
CHEBYQUAD	$(7, 7)$	1×10^{-5}	97	9.566×10^{-10}	73	2.030×10^{-10}
CHEBYQUAD	$(7, 7)$	1×10^{-7}	95	1.043×10^{-13}	73	6.456×10^{-14}
CHEBYQUAD	$(8, 8)$	1×10^{-1}	69	2.683×10^{-2}	79	6.164×10^{-2}
CHEBYQUAD	$(8, 8)$	1×10^{-3}	93	1.261×10^{-2}	96	1.182×10^{-3}
CHEBYQUAD	$(8, 8)$	1×10^{-5}	128	1.039×10^{-3}	199	6.120×10^{-7}
CHEBYQUAD	$(8, 8)$	1×10^{-7}	240	6.147×10^{-7}	208	1.288×10^{-9}
CHEBYQUAD	$(9, 9)$	1×10^{-1}	54**	2.888×10^{-2}	86	2.597×10^{-2}
CHEBYQUAD	$(9, 9)$	1×10^{-3}	100	2.013×10^{-2}	100	4.981×10^{-5}
CHEBYQUAD	$(9, 9)$	1×10^{-5}	134	2.326×10^{-9}	102	6.629×10^{-10}
CHEBYQUAD	$(9, 9)$	1×10^{-7}	161	3.856×10^{-14}	104	3.012×10^{-14}
CHEBYQUAD	$(10, 10)$	1×10^{-1}	94	4.026×10^{-2}	89	2.091×10^{-2}
CHEBYQUAD	$(10, 10)$	1×10^{-3}	153	9.237×10^{-4}	87	4.596×10^{-3}
CHEBYQUAD	$(10, 10)$	1×10^{-5}	163	5.114×10^{-6}	113	4.622×10^{-5}
CHEBYQUAD	$(10, 10)$	1×10^{-7}	207	1.732×10^{-3}	163	5.151×10^{-8}
CHEBYQUAD	$(11, 11)$	1×10^{-1}	78	2.983×10^{-2}	87	1.920×10^{-2}
CHEBYQUAD	$(11, 11)$	1×10^{-3}	104	9.225×10^{-3}	93	6.273×10^{-3}
CHEBYQUAD	$(11, 11)$	1×10^{-5}	213	4.205×10^{-4}	235	8.670×10^{-7}
CHEBYQUAD	$(11, 11)$	1×10^{-7}	449	1.141×10^{-6}	246	1.473×10^{-8}

Table A.23. Benchmarking Problems Tested. n is the number of variables and m is the dimension of the residual vector.

			LMDER		DFOLS	
Problem	(n, m)	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
BROWNAL	$(10, 10)$	1×10^{-1}	127	1.419	89	1.655×10^{2}
BROWNAL	$(10, 10)$	1×10^{-3}	147	5.220×10^{-7}	99	3.284×10^{-5}
BROWNAL	$(10, 10)$	1×10^{-5}	155	4.102×10^{-10}	89	9.707×10^{-10}
BROWNAL	$(10, 10)$	1×10^{-7}	131	1.084×10^{-13}	75	1.308×10^{-14}
OSBORNE1	$(5, 33)$	1×10^{-1}	53	1.279	53	5.737
OSBORNE1	$(5, 33)$	1×10^{-3}	75	3.424×10^{-3}	86	4.512×10^{-1}
OSBORNE1	$(5, 33)$	1×10^{-5}	86	1.010×10^{-2}	225	2.446×10^{-2}
OSBORNE1	$(5, 33)$	1×10^{-7}	376	4.951×10^{-8}	185	2.446×10^{-2}
OSBORNE2	$(11, 65)$	1×10^{-1}	113	1.297	104	2.226
OSBORNE2	$(11, 65)$	1×10^{-3}	426	6.798×10^{-3}	153	2.216×10^{-4}
OSBORNE2	$(11, 65)$	1×10^{-5}	308	9.726×10^{-5}	174	6.715×10^{-7}
OSBORNE2	$(11, 65)$	1×10^{-7}	235	1.654×10^{-7}	148	8.159×10^{-10}
OSBORNE2	$(11, 65)$	1×10^{-1}	124	2.799×10^{1}	111	2.293×10^{1}
OSBORNE2	$(11, 65)$	1×10^{-3}	204	1.754	132	1.750
OSBORNE2	$(11, 65)$	1×10^{-5}	277	1.750	134	1.750
OSBORNE2	$(11, 65)$	1×10^{-7}	223	1.750	139	1.750
BDQRTIC	$(8, 8)$	1×10^{-1}	124	5.023	89	4.566
BDQRTIC	$(8, 8)$	1×10^{-3}	128	2.317	103	2.289×10^{-1}
BDQRTIC	$(8, 8)$	1×10^{-5}	265	2.905×10^{-3}	122	1.864×10^{-2}
BDQRTIC	$(8, 8)$	1×10^{-7}	437	2.902×10^{-5}	173	8.502×10^{-5}
BDQRTIC	$(10, 12)$	1×10^{-1}	128	1.221×10^{1}	97	1.447×10^{2}
BDQRTIC	$(10, 12)$	1×10^{-3}	197	1.021×10^{-1}	124	1.141
BDQRTIC	$(10, 12)$	1×10^{-5}	340	5.836×10^{-3}	153	3.739×10^{-2}
BDQRTIC	$(10, 12)$	1×10^{-7}	574	6.811×10^{-5}	262	7.248×10^{-4}
BDQRTIC	$(11, 14)$	1×10^{-1}	139	7.006	106	9.772×10^{1}
BDQRTIC	$(11, 14)$	1×10^{-3}	188	1.299×10^{-1}	127	3.141
BDQRTIC	$(11, 14)$	1×10^{-5}	396	6.226×10^{-3}	190	7.543×10^{-2}
BDQRTIC	$(11, 14)$	1×10^{-7}	725	4.880×10^{-5}	333	2.513×10^{-3}
BDQRTIC	$(12, 16)$	1×10^{-1}	137	5.551	125	1.355×10^{2}
BDQRTIC	$(12, 16)$	1×10^{-3}	204	2.591×10^{-1}	135	2.972
BDQRTIC	$(12, 16)$	1×10^{-5}	283	1.565×10^{-2}	191	5.335×10^{-2}
BDQRTIC	$(12, 16)$	1×10^{-7}	626	3.785×10^{-4}	323	3.455×10^{-3}
CUBE	$(5, 5)$	1×10^{-1}	96	2.386×10^{-2}	54	6.129×10^{-2}
CUBE	$(5, 5)$	1×10^{-3}	112	3.481×10^{-3}	90	1.519×10^{-4}
CUBE	$(5, 5)$	1×10^{-5}	250	1.971×10^{-10}	367	4.920×10^{-12}
CUBE	$(5, 5)$	1×10^{-7}	415	9.699×10^{-14}	272	1.508×10^{-13}
CUBE	$(6, 6)$	1×10^{-1}	127	1.312×10^{-2}	56	5.761×10^{-2}
CUBE	$(6, 6)$	1×10^{-3}	162	1.853×10^{-4}	82	1.329×10^{-4}
CUBE	$(6, 6)$	1×10^{-5}	832	8.835×10^{-7}	460	2.319×10^{-6}
CUBE	$(6, 6)$	1×10^{-7}	958	9.762×10^{-14}	758	4.059×10^{-14}
CUBE	$(8, 8)$	1×10^{-1}	168	2.402×10^{-1}	63	9.869×10^{1}
CUBE	$(8, 8)$	1×10^{-3}	185	8.640×10^{-6}	87	7.661×10^{-4}
CUBE	$(8, 8)$	1×10^{-5}	621	1.251×10^{-5}	263	1.030×10^{-4}
CUBE	$(8, 8)$	1×10^{-7}	3431	6.217×10^{-8}	1763	2.522×10^{-7}

Table A.24. Benchmarking Problems Tested. n is the number of variables and m is the dimension of the residual vector.

			LMDER		DFOLS	
Problem	(n, m)	σ_f	#feval	$\phi(x) - \phi^*$	#feval	$\phi(x) - \phi^*$
MANCINO	$(5, 5)$	1×10^{-1}	47	7.329×10^{-2}	48	1.334×10^{-2}
MANCINO	$(5, 5)$	1×10^{-3}	45	7.341×10^{-6}	44	5.405×10^{-8}
MANCINO	$(5, 5)$	1×10^{-5}	39	7.341×10^{-10}	33	1.657×10^{-12}
MANCINO	$(5, 5)$	1×10^{-7}	39	7.342×10^{-14}	25	3.807×10^{-15}
MANCINO	$(5, 5)$	1×10^{-1}	47	7.435×10^{-2}	58	2.132×10^{-2}
MANCINO	$(5, 5)$	1×10^{-3}	45	7.511×10^{-6}	44	1.457×10^{-5}
MANCINO	$(5, 5)$	1×10^{-5}	45	1.247×10^{-9}	27	4.630×10^{-10}
MANCINO	$(5, 5)$	1×10^{-7}	45	8.751×10^{-14}	27	4.120×10^{-14}
MANCINO	$(8, 8)$	1×10^{-1}	72	1.550×10^{-1}	82	1.228×10^{-1}
MANCINO	$(8, 8)$	1×10^{-3}	78	1.130×10^{-7}	57	8.045×10^{-6}
MANCINO	$(8, 8)$	1×10^{-5}	60	1.549×10^{-9}	46	2.898×10^{-10}
MANCINO	$(8, 8)$	1×10^{-7}	60	1.549×10^{-13}	23	7.222×10^{-13}
MANCINO	$(10, 10)$	1×10^{-1}	109	8.335×10^{-5}	82	7.748×10^{-3}
MANCINO	$(10, 10)$	1×10^{-3}	107	8.191×10^{-9}	59	8.847×10^{-6}
MANCINO	$(10, 10)$	1×10^{-5}	74	6.522×10^{-10}	45	1.015×10^{-10}
MANCINO	$(10, 10)$	1×10^{-7}	74	6.526×10^{-14}	36	5.372×10^{-15}
MANCINO	$(12, 12)$	1×10^{-1}	114	3.374×10^{-4}	107	3.447×10^{-2}
MANCINO	$(12, 12)$	1×10^{-3}	101	3.354×10^{-8}	77	8.807×10^{-6}
MANCINO	$(12, 12)$	1×10^{-5}	101	3.353×10^{-12}	60	9.663×10^{-10}
MANCINO	$(12, 12)$	1×10^{-7}	101	3.357×10^{-16}	32	9.721×10^{-14}
MANCINO	$(12, 12)$	1×10^{-1}	114	3.374×10^{-4}	126	9.117×10^{-2}
MANCINO	$(12, 12)$	1×10^{-3}	114	3.348×10^{-8}	72	9.369×10^{-6}
MANCINO	$(12, 12)$	1×10^{-5}	101	3.357×10^{-12}	35	4.632×10^{-9}
MANCINO	$(12, 12)$	1×10^{-7}	101	3.452×10^{-16}	35	2.607×10^{-9}
HEART8LS	$(8, 8)$	1×10^{-1}	113	6.289×10^{-2}	71	1.356×10^{-1}
HEART8LS	$(8, 8)$	1×10^{-3}	83	1.244×10^{-6}	66	3.717×10^{-6}
HEART8LS	$(8, 8)$	1×10^{-5}	82	1.508×10^{-11}	66	3.928×10^{-12}
HEART8LS	$(8, 8)$	1×10^{-7}	79	1.167×10^{-15}	67	1.275×10^{-14}
HEART8LS	$(8, 8)$	1×10^{-1}	156	5.632	86	5.270
HEART8LS	$(8, 8)$	1×10^{-3}	566	7.222×10^{-6}	77	4.911
HEART8LS	$(8, 8)$	1×10^{-5}	164	1.102×10^{-9}	172	4.794
HEART8LS	$(8, 8)$	1×10^{-7}	161	5.788×10^{-14}	376	4.428

Table A.25. Benchmarking Problems Tested. n is the number of variables and m is the dimension of the residual vector.

Problem	Objective Type	Number of Variables (n)	One-sided Bounds	Two-sided Bounds	Number of Constraints (m)	Linear Ineq.	Nonlinear Ineq.
BURKEHAN	quadratic	1	1	0	1	0	1
CANTILVR	linear	5	5	0	1	0	1
CB2	linear	3	0	0	3	0	3
CB3	linear	3	0	0	3	0	3
CHACONN1	linear	3	0	0	3	0	3
CHACONN2	linear	3	0	0	3	0	3
CRESC4	general	6	3	1	8	0	8
CRESC50	general	6	3	1	100	0	100
DEMBO7	quadratic	16	0	16	20	1	19
DIPIGRI	general	7	0	0	4	0	4
GIGOMEZ2	linear	3	0	0	3	0	3
GIGOMEZ3	linear	3	0	0	3	0	3
HALDMADS	linear	6	0	0	42	0	42
HS100	general	7	0	0	4	0	4
HS100MOD	general	7	0	0	4	0	4
HS101	general	7	0	7	5	0	5
HS102	general	7	0	7	5	0	5
HS103	general	7	0	7	5	0	5
HS104	general	8	0	8	5	0	5
HS13	quadratic	2	2	0	1	0	1
HS34	linear	3	0	3	2	0	2
HS64	general	3	3	0	1	0	1
HS66	linear	3	0	3	2	0	2
HS67	general	3	0	3	14	0	14
HS72	linear	4	0	4	2	0	2
HS85	general	5	0	5	21	1	20
HS88	quadratic	2	0	0	1	0	1
HS89	quadratic	3	0	0	1	0	1
HS90	quadratic	4	0	0	1	0	1
HS91	quadratic	5	0	0	1	0	1
HS92	quadratic	6	0	0	1	0	1
HS93	general	6	6	0	2	0	2
MADSEN	linear	3	0	0	6	0	6
MATRIX2	quadratic	6	4	0	2	0	2
MINMAXBD	linear	5	0	0	20	0	20
POLAK1	linear	3	0	0	2	0	2
POLAK2	linear	11	0	0	2	0	2
POLAK3	linear	12	0	0	10	0	10
POLAK5	linear	3	0	0	2	0	2
POLAK6	linear	5	0	0	4	0	4
S365	quadratic	7	4	0	5	0	5
S365MOD	quadratic	7	4	0	5	0	5
SNAKE	linear	2	0	0	2	0	2
SPIRAL	linear	3	0	0	2	0	2
SYNTHES1	general	6	0	6	6	4	2
TWOBARS	general	2	0	2	2	0	2
WOMFLET	linear	3	0	0	3	0	3

Table A.26. Properties of small-scale `CUTEst` problems.

Problem	SIF Parameter	Testing Name	d	m
SVANBERG	$N = 10$	SVANBERGN10	10	10
SVANBERG	$N = 50$	SVANBERGN50	50	50
SVANBERG	$N = 100$	SVANBERGN100	100	100
SVANBERG	$N = 500$	SVANBERGN500	500	500
READING4	$N = 2$	READING4N2	2	2
READING4	$N = 50$	READING4N50	50	50
READING4	$N = 100$	READING4N100	100	100
READING4	$N = 500$	READING4N500	500	500
COSHFUN	$M = 3$	COSHFUNM3	10	3
COSHFUN	$M = 8$	COSHFUNM8	25	8
COSHFUN	$M = 14$	COSHFUNM14	43	14
COSHFUN	$M = 20$	COSHFUNM20	61	20

Table A.27. Instances of variable-size `CUTEst` problems.

	KNITRO				COBYLA			
Problem	$\phi(x)$	#feval	CPU	feaserr	$\phi(x)$	#fevals	CPU	feaserr
CB3	2.0000	40	0.023	2.22e-16	2.0000	44	0.157	0.00e+00
CHACONN2	2.0000	40	0.021	0.00e+00	2.0000	48	0.156	0.00e+00
GIGOMEZ3	2.0000	40	0.022	0.00e+00	2.0000	45	0.160	0.00e+00
HS100	680.6301	360	0.160	0.00e+00	680.6300	454	0.272	0.00e+00
DIPIGRI	680.6301	389	0.152	2.84e-14	680.6300	458	0.288	0.00e+00
HS93	135.0760	2247	1.040	4.44e-16	135.0760	3000*	0.873	0.00e+00
HS64	6299.8424	144	0.093	0.00e+00	6299.8400	409	0.244	0.00e+00
POLAK3	5.9330	902	0.414	3.89e-16	5.9330	3139	0.071	0.00e+00
POLAK1	2.7183	60	0.034	0.00e+00	2.7183	326	0.232	0.00e+00
HS104	3.9512	718	0.282	0.00e+00	3.9512	3977	0.131	0.00e+00
HS100MOD	678.6796	477	0.207	0.00e+00	678.6790	385	0.240	0.00e+00
TWOBARS	1.5087	63	0.037	0.00e+00	1.5087	83	0.172	0.00e+00
HS85	-2.2156	117	0.078	7.11e-15	-2.2156	299	0.242	0.00e+00
CB2	1.9522	72	0.041	1.11e-16	1.9522	111	0.173	0.00e+00
CHACONN1	1.9522	55	0.030	0.00e+00	1.9522	120	0.175	0.00e+00
MADSEN	0.6164	73	0.042	0.00e+00	0.6164	112	0.175	0.00e+00
HS66	0.5182	43	0.026	4.44e-16	0.5182	101	0.170	0.00e+00
MINMAXBD	115.7064	420	0.335	6.17e-14	115.7060	924	0.395	0.00e+00
CANTILVR	1.3400	170	0.078	3.61e-16	1.3400	281	0.227	0.00e+00
HS92	1.3627	976	0.988	4.60e-17	1.3627	3000*	0.993	0.00e+00
HALDMADS	0.0001	113	0.073	4.44e-16	0.0001	669	0.341	0.00e+00
HS34	-0.8340	59	0.033	4.44e-16	-0.8340	47	0.149	0.00e+00
HS90	1.3627	570	0.475	0.00e+00	1.3629	2000*	0.066	0.00e+00
SNAKE	0.0000	25	0.029	1.02e-16	0.0000	72	0.165	0.00e+00
HS72	727.6794	108	0.103	1.73e-18	727.6794	1002	0.424	4.77e-18
GIGOMEZ2	1.9522	79	0.048	2.22e-16	1.9522	111	0.169	1.11e-16
SYNTHES1	0.7593	96	0.041	0.00e+00	0.7593	174	0.186	1.54e-23
HS88	1.3627	209	0.156	0.00e+00	1.3627	167	0.203	2.09e-17
HS13	0.9999	140	0.155	2.61e-13	1.0000	86	0.168	1.07e-27
MATRIX2	0.0000	275	0.131	3.60e-17	0.0000	216	0.199	5.20e-18
WOMFLET	0.0000	103	0.070	5.89e-10	0.0000	117	0.181	8.99e-18
S365	0.0000	63	0.032	1.72e-07	0.0000	210	0.215	4.35e-17
HS67	-1162.1187	285	0.198	0.00e+00	-980.1600	7000*	0.885	0.00e+00
CRESC50	0.5952	16704	17.912	0.00e+00	7.9917	50000*	4.863	0.00e+00
CRESC4	0.8719	688	0.363	0.00e+00	44.5901	4000*	0.318	0.00e+00
HS91	1.3627	1229	1.057	9.96e-18	1.1199	2500*	0.583	6.50e-04
DEMBO7	174.7870	3225	1.594	3.61e-16	250.0720	10000*	0.238	7.29e-02
POLAK5	50.0000	65	0.043	0.00e+00	34.4929	1500*	0.571	1.55e+01
HS101	1809.7691	3500*	1.908	1.38e-13	3000.1900	3500*	0.077	1.99e-01
HS89	1.3627	315	0.257	2.17e-17	0.6520	1500*	0.742	5.12e-02
HS102	911.8816	3500*	1.965	4.67e-12	3000.3500	3500*	0.096	3.60e-01
HS103	543.6680	3500*	1.858	2.57e-16	3000.1700	3500*	0.091	1.76e-01
SPIRAL	0.0195	1500*	0.796	1.34e-05	0.0958	1500*	0.508	0.00e+00
POLAK6	-78.6745	474	1.311	8.55e+01	0.0000	2500*	0.765	0.00e+00
S365MOD	0.2500	162	0.204	1.25e+00	0.0301	247	0.211	3.26e-01
BURKEHAN	10.0000	13	0.007	1.01e+02	0.0000	54	0.175	1.00e+00
POLAK2	29.9570	2641	5.119	5.39e+01	-259.2500	5500*	0.794	3.14e+02

Table A.28. Summary of the results for small-scale noiseless constrained `CUTEst` problems. The horizontal bars divide cases (i), (ii), (iii), and (iv).

		$TOL = 10^{-6}$				$TOL = 10^{-3}$				$TOL = 10^{-1}$				
			KNITRO		COBYLA			KNITRO		COBYLA			KNITRO		COBYLA	
problem	ϕ^*	target ϕ	$\phi(x)$	#fevals	$\phi(x)$	#fevals	target ϕ	$\phi(x)$	#fevals	$\phi(x)$	#fevals	target ϕ	$\phi(x)$	#fevals	$\phi(x)$	#fevals
CB3	2.000000	2.00000	2.00000	30	2.00000	25	2.00200	2.00000	30	2.00000	25	2.20000	2.00000	30	2.00351	18
CHACONN2	2.000000	2.00000	2.00000	30	2.00000	32	2.00200	2.00000	30	2.00021	24	2.20000	2.00000	30	2.06187	19
GIGOMEZ3	2.000000	2.00000	2.00000	30	2.00000	39	2.00200	2.00000	30	2.00000	39	2.20000	2.00000	30	2.00000	39
HS100	680.630057	680.63074	680.63013	124	680.63000	247	681.31069	680.75511	75	680.63000	247	748.69306	714.00000	1	714.00000	1
DIPIGRI	680.630057	680.63074	680.63013	124	680.63000	410	681.31069	680.75511	75	680.63100	141	748.69306	714.00000	1	714.00000	1
HS93	135.075964	135.07610	135.07610	871	135.07600	2416	135.21104	135.14058	229	135.21000	270	148.58356	137.06640	1	137.06600	1
HS64	6299.842428	6299.84873	6299.73981	64	6299.84000	335	6306.14227	6302.48265	54	6305.80000	316	6929.82667	6687.88225	30	6684.54000	239
POLAK3	5.933003	5.93301	5.93301	469	5.93300	2637	5.93894	5.93353	349	5.93644	2350	6.52630	5.93353	349	5.94016	2328
POLAK1	2.718282	2.71828	2.71828	50	2.71828	265	2.72100	2.71828	50	2.71834	253	2.99011	2.71828	50	2.83811	250
HS104	3.951163	3.95117	3.95117	290	3.95116	1764	3.95511	3.95148	231	3.95381	730	4.34628	3.95946	193	4.15766	173
HS100MOD	678.679638	678.68032	678.67969	173	678.68000	169	679.35832	679.08174	63	679.18500	50	746.54760	714.00000	1	714.00000	1
TWOBARS	1.508652	1.50865	1.50865	50	1.50865	33	1.51016	1.50865	50	1.50892	21	1.65952	1.50865	50	1.65447	10
HS85	-2.215605	-2.21560	-2.21560	110	-2.21560	299	-2.21339	-2.21560	110	-2.21500	247	-1.99404	-2.21560	110	-2.02180	158
CB2	1.952224	1.95223	1.95222	52	1.95222	57	1.95418	1.95222	52	1.95230	41	2.14745	1.95222	52	1.95230	41
CHACONN1	1.952224	1.95223	1.95222	40	1.95222	55	1.95418	1.95222	40	1.95222	55	2.14745	1.95222	40	2.00282	13
MADSEN	0.616432	0.61643	0.61643	58	0.61643	65	0.61743	0.61643	58	0.61643	65	0.71643	0.61643	58	0.65433	22
HS66	0.518163	0.51816	0.51816	33	0.51816	37	0.51916	0.51866	21	0.51817	28	0.61816	0.58000	1	0.58000	1
MINMAXBD	115.706440	115.70656	115.70644	343	115.70600	800	115.82215	115.70644	343	115.72100	766	127.27708	118.74188	246	115.72100	766
CANTILVR	1.339956	1.33996	1.33994	107	1.33995	189	1.34130	1.33994	107	1.33997	130	1.47395	1.33994	107	1.34758	107
HS92	1.362657	1.36266	1.36266	844	1.36265	2121	1.36402	1.36327	405	1.36310	203	1.49892	1.36327	405	1.36310	203
HALDMADS	0.000122	0.00012	0.00012	97	0.00013	669	0.00112	0.00012	97	0.00057	51	0.10012	0.00012	97	0.00057	51
HS34	-0.834032	-0.83403	-0.83403	54	-0.83400	40	-0.83303	-0.83403	54	-0.83400	25	-0.73403	-0.83186	44	-0.83400	25
HS90	1.362657	1.36266	1.36185	259	1.36287	2000	1.36402	1.36185	259	1.36311	159	1.49892	1.36185	259	1.36311	159
SNAKE	0.000000	0.00000	0.00000	17	0.00000	67	0.00100	0.00000	17	0.00009	61	0.10000	0.00000	17	0.00009	61
HS72	727.679358	727.68009	727.66249	72	727.67973	877	728.40704	727.66249	72	728.36304	856	800.44729	727.66249	72	738.77714	832
GIGOMEZ2	1.952224	1.95223	1.95222	64	1.95223	47	1.95418	1.95222	64	1.95266	32	2.14745	1.95222	64	2.08642	13
SYNTHES1	0.759284	0.75929	0.75928	72	0.75928	79	0.76028	0.75928	72	0.75928	79	0.85928	0.75928	72	0.79837	17
HS88	1.362657	1.36266	1.36249	189	1.36266	138	1.36402	1.36249	189	1.36387	123	1.49892	1.36249	189	1.36387	123
HS13	0.999872	0.99987	0.99987	116	1.00000	86	1.00087	1.00060	88	1.00085	41	1.09987	1.07956	40	1.07745	22
MATRIX2	0.000000	0.00000	0.00000	131	0.00000	187	0.00100	0.00001	107	0.00000	187	0.10000	0.00001	107	0.09622	46
WOMFLET	0.000000	0.00000	0.00000	78	0.00000	104	0.00100	0.00000	78	0.00000	104	0.10000	0.00000	78	0.00000	104
S365	0.000000	0.00000	0.00000	63	0.00000	179	0.00100	0.00000	63	0.00000	179	0.10000	0.00000	63	0.00000	179

Table A.29. Function evaluations to obtain $\phi_k \leq \phi^* + TOL \cdot \max\{1.0, |\phi^*|\}$ for small scale noiseless constrained problems.

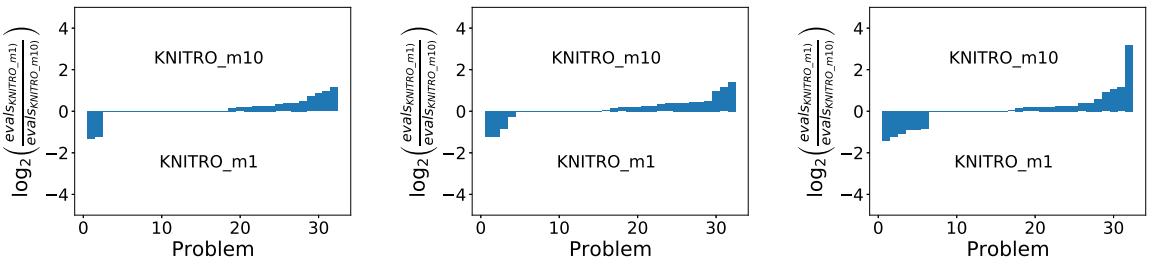

Figure A.14. *Efficiency, Noiseless Case.* Log-ratio profiles comparing KNITRO with an L-BFGS Hessian approximation of memory one and memory 10 when $\epsilon(x) = 0$. The figures measure number of function evaluations to satisfy (2.2.7) for $\tau = 10^{-1}$ (left), 10^{-3} (middle), 10^{-6} (right).

A.3.3.3. Noisy Functions. In this section, we list the final objective value($\phi(x)$), feasibility error(feaserr) and the number of function evaluations (#feval) needed to achieve this for each problem instance, varying the noise level $\sigma_f \in \{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$, in Tables A.30-A.35. Function evaluations marked with a * denote cases where the algorithm reached the maximum number of function evaluations. An 'f' marks the cases where the feasibility error is large compared to the noise level, i.e., $\| \max\{\psi(x), 0\}\|_\infty \geq \sqrt{3\sigma_f}$.

	KNITRO					COBYLA				
σ_f	0.0	1e-07	1e-05	1e-03	1e-01	0.0	1.00e-07	1.00e-05	1.00e-03	1.00e-01
CB3	4.92e-12	1.00e-08	2.00e-06	1.35e-04	1.51e-01	8.36e-12	1.00e-08	6.00e-06	1.10e-03	2.97e-01
CHACONN2	1.72e-12	1.00e-08	1.60e-05	3.80e-04	8.27e-02	1.72e-12	1.00e-08	3.00e-06	1.01e-03	f
GIGOMEZ3	9.80e-13	1.00e-08	2.00e-06	1.35e-04	1.62e-01	9.80e-13	1.00e-08	1.00e-05	9.30e-05	f
HS100	2.05e-12	2.37e-06	2.14e-04	1.34e-02	2.16e+00	5.87e-07	1.63e-06	9.62e-04	3.13e-02	3.65e-02
DIPIGRI	4.09e-12	3.24e-05	1.87e-04	1.25e-02	1.98e+00	2.05e-12	1.63e-06	4.23e-04	3.21e-02	9.14e-01
HS93	3.90e-09	4.34e-03	6.21e-02	2.49e-01	2.69e+00	3.96e-05	1.52e-01	4.88e-01	1.79e+00	1.99e+00
HS64	2.28e-09	4.29e-05	3.10e-03	4.90e+00	8.78e+02	2.28e-09	2.62e-04	7.11e-02	f	f
POLAK3	1.00e-12	6.51e-07	2.13e-04	4.39e-03	f	1.01e-12	5.65e-06	3.58e-04	4.13e-04	f
POLAK1	1.00e-12	1.72e-07	6.17e-06	1.53e-03	2.62e+00	9.90e-13	1.72e-07	1.17e-06	3.40e-02	5.15e+00
HS104	1.05e-11	5.36e-05	1.30e-02	2.99e-02	6.63e-01	1.05e-11	1.55e-03	1.38e-02	1.37e-01	2.74e+00
HS100MOD	1.02e-12	2.01e-05	2.48e-04	2.06e-02	1.34e+00	1.02e-12	1.11e-07	2.13e-04	8.63e-03	1.20e+00
TWOBARS	1.51e-12	4.18e-07	1.14e-05	1.39e-04	9.27e-02	1.51e-12	4.18e-07	1.54e-04	1.56e-03	5.86e-02
HS85	6.00e-14	3.12e-07	3.12e-07	5.72e-03	9.27e-01	4.51e-06	3.12e-07	6.88e-07	4.82e-01	9.59e-01
CB2	3.70e-13	5.06e-07	4.49e-06	5.47e-04	2.05e-01	3.70e-13	4.94e-07	3.49e-06	8.49e-06	f
CHACONN1	8.80e-13	4.94e-07	6.49e-06	3.96e-03	1.34e-01	1.07e-12	4.94e-07	1.51e-06	6.81e-04	2.44e-02
MADSEN	1.00e-12	5.64e-07	7.56e-06	1.11e-04	2.58e-01	1.01e-12	5.64e-07	6.44e-06	2.90e-04	2.66e-01
HS66	8.64e-13	2.74e-07	1.37e-05	6.56e-04	2.53e-02	8.64e-13	2.74e-07	4.27e-06	5.23e-04	9.11e-03
MINMAXBD	1.01e-12	3.21e-07	5.63e-05	3.75e-03	f	2.52e-10	3.21e-07	2.03e-05	1.65e-04	1.31e+01
CANTILVR	4.40e-13	3.61e-07	1.64e-06	1.72e-03	1.91e-01	4.40e-13	6.39e-07	7.16e-05	8.57e-03	f
HS92	1.06e-09	1.16e-02	1.18e-01	f	9.97e-01	1.40e-07	5.02e-03	2.51e-01	2.14e+00	1.07e+00
HALDMADS	9.94e-13	3.71e-07	1.06e-05	7.51e-03	2.53e-01	9.52e-06	3.66e-05	5.66e-05	4.94e-04	f
HS34	2.89e-13	4.45e-07	2.55e-06	4.24e-04	6.93e-01	3.92e-09	4.45e-07	1.24e-05	1.82e-03	8.34e-01
HS90	1.06e-09	1.09e-04	6.99e-01	f	5.73e-01	2.18e-04	7.24e-03	3.13e-01	1.30e+00	1.03e+00
SNAKE	7.96e-13	f	2.28e+00	4.51e+01	8.35e+00	6.74e-07	2.12e-02	3.24e-02	3.56e-02	1.28e+00
HS72	0.00e+00	6.83e-02	2.53e+01	3.78e+02	f	0.00e+00	1.63e-02	f	f	f
GIGOMEZ2	0.00e+00	4.94e-07	1.15e-05	3.21e-04	7.78e-02	0.00e+00	4.94e-07	2.95e-05	9.14e-04	1.04e-01
SYNTHES1	0.00e+00	2.39e-06	7.39e-06	6.80e-03	1.22e+00	4.00e-15	2.61e-06	7.98e-02	8.49e-02	1.10e+00
HS88	1.10e-13	1.29e-04	3.85e-02	4.04e-01	1.32e+00	4.08e-12	1.76e-04	1.82e-01	7.87e-01	1.13e+00
HS13	2.97e-08	1.08e-02	4.54e-02	1.43e-01	6.42e-01	1.28e-04	6.70e-03	2.95e-02	1.15e-01	1.38e+00
MATRIX2	7.93e-16	1.00e-08	1.10e-05	3.63e-03	7.53e-02	3.82e-17	1.00e-08	3.80e-05	2.52e-03	1.19e+00
WOMFLET	7.89e-13	2.90e-05	1.39e-03	8.39e-03	6.25e+00	7.89e-13	1.00e-08	2.00e-06	3.12e-03	3.34e+00
S365	0.00e+00	1.00e-08	1.00e-08	1.00e-08	1.00e-08	8.89e-35	1.00e-08	1.00e-08	1.00e-08	1.00e-08

Table A.30. Optimality gap $|\phi(x_k) - \phi^*|$ for small scale noisy constrained `CUTEst` problems.

	KNITRO					COBYLA				
σ_f	0.0	1e-07	1e-05	1e-03	1e-01	0.0	1.00e-07	1.00e-05	1.00e-03	1.00e-01
CB3	2.22e-16	0.00e+00	0.00e+00	7.50e-05	0.00e+00	0.00e+00	0.00e+00	1.20e-05	0.00e+00	3.71e-02
CHACONN2	0.00e+00	0.00e+00	0.00e+00	1.22e-04	2.54e-01	0.00e+00	0.00e+00	5.00e-06	0.00e+00	1.53e+00
GIGOMEZ3	0.00e+00	0.00e+00	0.00e+00	7.50e-05	0.00e+00	0.00e+00	0.00e+00	0.00e+00	3.32e-04	5.89e-01
HS100	0.00e+00	5.00e-06	2.10e-05	6.18e-04	1.01e-01	0.00e+00	0.00e+00	0.00e+00	0.00e+00	1.67e-01
DIPIGRI	2.84e-14	3.10e-05	3.00e-06	5.53e-04	1.38e-01	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
HS93	4.44e-16	0.00e+00	1.00e-05	1.03e-03	1.13e-01	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
HS64	0.00e+00	0.00e+00	1.00e-05	2.18e-03	4.96e-01	0.00e+00	0.00e+00	0.00e+00	1.82e-01	4.97e+00
POLAK3	3.89e-16	1.00e-06	8.00e-06	1.34e-03	5.56e-01	0.00e+00	0.00e+00	0.00e+00	8.60e-03	8.72e+00
POLAK1	0.00e+00	0.00e+00	0.00e+00	2.54e-04	6.74e-02	0.00e+00	0.00e+00	1.10e-05	9.15e-04	9.40e-02
HS104	0.00e+00	0.00e+00	0.00e+00	1.04e-03	2.81e-01	0.00e+00	0.00e+00	3.00e-06	2.80e-02	3.70e-01
HS100MOD	0.00e+00	0.00e+00	2.80e-05	3.91e-04	1.77e-01	0.00e+00	0.00e+00	0.00e+00	0.00e+00	7.18e-02
TWOBARS	0.00e+00	0.00e+00	9.00e-06	1.28e-03	1.86e-02	0.00e+00	0.00e+00	0.00e+00	1.04e-03	1.40e-01
HS85	7.11e-15	1.00e-06	7.00e-06	1.52e-04	0.00e+00	0.00e+00	0.00e+00	0.00e+00	1.41e-03	0.00e+00
CB2	1.11e-16	0.00e+00	1.40e-05	0.00e+00	1.15e-02	0.00e+00	0.00e+00	6.00e-06	1.02e-03	1.10e+00
CHACONN1	0.00e+00	1.00e-06	1.40e-05	0.00e+00	2.43e-03	0.00e+00	0.00e+00	2.00e-06	7.34e-04	2.38e-02
MADSEN	0.00e+00	0.00e+00	0.00e+00	4.89e-04	0.00e+00	0.00e+00	0.00e+00	1.00e-05	3.03e-04	0.00e+00
HS66	4.44e-16	0.00e+00	3.00e-06	0.00e+00	0.00e+00	0.00e+00	0.00e+00	5.00e-06	1.21e-03	4.57e-02
MINMAXBD	6.17e-14	2.20e-05	1.90e-05	2.96e-04	4.65e+02	0.00e+00	0.00e+00	0.00e+00	1.53e-03	0.00e+00
CANTILVR	3.61e-16	0.00e+00	8.00e-06	0.00e+00	5.74e-02	0.00e+00	0.00e+00	0.00e+00	0.00e+00	2.47e+00
HS92	4.60e-17	0.00e+00	2.58e-04	1.33e-01	1.35e-01	0.00e+00	0.00e+00	6.73e-04	1.04e-03	7.11e-02
HALDMADS	4.44e-16	2.00e-06	1.40e-05	0.00e+00	0.00e+00	0.00e+00	0.00e+00	1.50e-05	9.94e-04	1.82e+01
HS34	4.44e-16	0.00e+00	6.00e-06	0.00e+00	1.15e-01	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
HS90	0.00e+00	0.00e+00	1.20e-05	1.33e-01	1.40e-01	0.00e+00	0.00e+00	1.07e-03	6.30e-05	1.20e-01
SNAKE	1.02e-16	1.70e-03	2.12e-04	2.90e-03	1.87e-01	0.00e+00	1.00e-06	3.00e-06	3.27e-04	3.65e-01
HS72	1.73e-18	0.00e+00	7.31e-04	4.62e-02	5.76e-01	4.77e-18	0.00e+00	7.94e-03	3.47e-01	3.26e+00
GIGOMEZ2	2.22e-16	2.00e-06	0.00e+00	6.68e-04	1.78e-01	1.11e-16	0.00e+00	0.00e+00	1.29e-03	0.00e+00
SYNTHES1	0.00e+00	1.00e-06	5.00e-06	0.00e+00	2.09e-01	1.54e-23	0.00e+00	0.00e+00	0.00e+00	0.00e+00
HS88	0.00e+00	0.00e+00	4.40e-05	2.05e-03	1.26e-01	2.09e-17	0.00e+00	3.57e-04	1.82e-02	1.47e-01
HS13	2.61e-13	0.00e+00	1.20e-05	4.07e-04	6.52e-02	1.07e-27	0.00e+00	3.00e-06	0.00e+00	0.00e+00
MATRIX2	3.60e-17	0.00e+00	9.00e-06	7.30e-04	2.95e-02	5.20e-18	0.00e+00	6.00e-06	3.17e-04	0.00e+00
WOMFLET	5.89e-10	1.50e-05	1.00e-06	4.53e-04	5.05e-02	8.99e-18	0.00e+00	5.00e-06	0.00e+00	0.00e+00
S365	1.72e-07	0.00e+00	2.00e-06	6.43e-04	6.98e-03	4.35e-17	0.00e+00	7.00e-06	1.70e-04	1.72e-02

Table A.31. Feasibility error $\|\max\{\psi(x_k), 0\}\|_\infty$ for small scale noisy constrained `CUTEst` problems.

	$\tau = 10^{-6}$						$\tau = 10^{-2}$					
	KNITRO			COBYLA			KNITRO			COBYLA		
problem	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals
CB3	2.000000	0.00e+00	35	2.000000	0.00e+00	30	1.9999980	6.00e-06	30	2.000030	1.20e-05	20
CHACONN2	1.999998	6.00e-06	30	2.000000	0.00e+00	29	1.9999980	6.00e-06	30	1.999631	5.01e-04	22
GIGOMEZ3	2.000000	0.00e+00	35	2.000000	0.00e+00	24	2.0000000	0.00e+00	35	2.000000	0.00e+00	24
HS100	680.630087	8.00e-06	173	680.630055	1.15e-04	125	680.7482460	0.00e+00	75	680.707367	0.00e+00	87
DIPIGRI	680.630025	3.10e-05	189	680.630055	2.80e-05	205	680.7513980	0.00e+00	75	680.638226	2.42e-04	114
HS93	135.080301	1.00e-06	650	135.092558	2.00e-06	f	135.0937190	1.00e-06	477	135.080622	5.37e-04	535
HS64	6299.795253	2.10e-05	77	6300.065790	1.20e-05	312	6688.2748180	5.70e-05	30	6704.980195	1.24e-04	238
POLAK3	5.933006	1.00e-06	357	5.933005	1.19e-04	1333	5.9332460	1.55e-04	268	5.939929	0.00e+00	1220
POLAK1	2.718281	1.00e-06	50	2.718279	9.00e-06	251	2.7182600	5.50e-05	45	2.743571	7.00e-05	216
HS104	3.951218	0.00e+00	305	3.952662	1.00e-06	f	3.9535870	1.20e-05	198	3.953450	4.63e-04	353
HS100MOD	678.679616	5.42e-04	146	678.679644	7.50e-05	179	678.7677270	4.00e-06	110	678.849653	0.00e+00	83
TWOBARS	1.508653	0.00e+00	55	1.508652	0.00e+00	40	1.5086550	3.00e-06	45	1.508930	0.00e+00	21
HS85	-2.215605	0.00e+00	152	-2.215605	9.00e-06	246	-2.2156220	5.41e-04	145	-2.206122	0.00e+00	233
CB2	1.952224	2.00e-06	57	1.952225	1.00e-06	48	1.9522210	4.00e-06	52	1.960966	0.00e+00	22
CHACONN1	1.952225	1.00e-06	40	1.952225	1.00e-06	37	1.9521680	8.20e-05	35	1.955056	0.00e+00	20
MADSEN	0.616433	0.00e+00	58	0.616432	4.74e-04	32	0.6164330	0.00e+00	58	0.616264	5.11e-04	31
HS66	0.518164	0.00e+00	33	0.518163	0.00e+00	37	0.5186740	0.00e+00	21	0.518026	3.71e-04	26
MINMAXBD	115.706388	1.17e-04	485	115.706134	4.03e-04	783	115.7063880	1.17e-04	485	115.919506	0.00e+00	741
CANTILVR	1.339957	0.00e+00	211	1.339958	3.80e-05	131	1.3415130	0.00e+00	187	1.347590	0.00e+00	107
HS92	1.374297	0.00e+00	f	1.367672	0.00e+00	139	1.3742970	0.00e+00	f	1.366663	2.00e-06	117
HALDMADS	0.000121	6.00e-06	78	0.000159	1.00e-06	f	0.0001210	6.00e-06	78	0.000159	1.00e-06	f
HS34	-0.834033	1.00e-06	54	-0.834032	0.00e+00	34	-0.8320120	0.00e+00	44	-0.834016	0.00e+00	25
HS90	1.362766	0.00e+00	326	1.368155	1.00e-06	f	1.3608930	2.00e-06	206	1.360847	1.10e-05	156
SNAKE	-22.242365	1.70e-03	f	-0.021173	2.00e-06	f	-22.2423650	1.70e-03	f	-0.021173	2.00e-06	f
HS72	727.747619	0.00e+00	f	727.695652	0.00e+00	870	721.1362080	3.33e-04	66	720.999346	3.14e-04	815
GIGOMEZ2	1.952224	1.00e-06	64	1.952225	1.80e-05	46	1.9520790	2.15e-04	59	1.952432	0.00e+00	31
SYNTHES1	0.759272	1.60e-05	64	0.759285	1.00e-06	53	0.7591430	1.95e-04	56	0.785862	1.00e-06	30
HS88	1.362528	0.00e+00	331	1.362528	0.00e+00	114	1.3581270	5.00e-06	240	1.354190	8.00e-06	93
HS13	0.989068	0.00e+00	97	1.006571	0.00e+00	f	1.1183300	0.00e+00	36	1.138030	0.00e+00	19
MATRIX2	0.000000	0.00e+00	153	0.000001	0.00e+00	113	0.0000000	0.00e+00	153	0.000001	0.00e+00	113
WOMFLET	0.000029	1.50e-05	f	0.000006	6.00e-05	121	0.0230460	4.21e-04	81	0.000468	3.02e-04	68
S365	0.000000	5.41e-04	54	0.000000	3.13e-04	48	0.0000000	5.41e-04	54	0.000000	3.13e-04	48

Table A.32. Number of function evaluations to obtain (2.4.5) for noise level $\sigma_f = 10^{-7}$.

	$\tau = 10^{-6}$						$\tau = 10^{-2}$					
	KNITRO			COBYLA			KNITRO			COBYLA		
problem	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals
CB3	2.000003	0.00e+00	47	2.000000	8.00e-06	32	1.999981	2.50e-05	30	2.000031	1.10e-05	20
CHACONN2	2.000006	4.00e-06	45	2.000001	2.79e-04	31	1.999999	1.90e-05	30	1.999624	5.08e-04	22
GIGOMEZ3	2.000003	0.00e+00	47	2.000003	1.26e-03	25	2.000003	0.00e+00	47	2.000003	1.26e-03	25
HS100	680.630284	9.00e-06	160	680.630275	1.47e-04	152	680.849165	0.00e+00	46	680.757842	1.82e-03	56
DIPIGRI	680.630221	7.00e-06	149	680.630243	5.00e-05	164	680.817892	1.54e-03	56	680.718994	3.36e-04	58
HS93	135.138021	1.00e-05	229	135.416906	1.60e-05	f	135.133971	1.16e-03	66	135.416906	1.60e-05	f
HS64	6299.871889	0.00e+00	66	6299.950263	2.61e-03	308	6708.036271	2.43e-03	25	6669.894009	2.09e-03	246
POLAK3	5.933216	8.00e-06	313	5.933217	7.40e-05	2303	5.932429	3.09e-03	270	5.943411	4.15e-03	2186
POLAK1	2.718282	6.00e-06	50	2.718280	1.20e-05	269	2.718182	9.00e-04	45	2.743588	4.30e-05	230
HS104	3.964210	0.00e+00	170	3.963888	1.04e-04	f	3.964996	3.71e-04	109	3.965500	9.61e-04	113
HS100MOD	678.679887	1.30e-05	193	678.679885	5.49e-04	122	678.947334	1.28e-03	73	678.953589	0.00e+00	52
TWOBARS	1.508641	9.00e-06	155	1.508665	1.90e-05	f	1.508463	1.28e-04	41	1.508969	0.00e+00	21
HS85	-2.215605	1.70e-05	106	-2.215605	3.30e-05	243	-2.215628	6.85e-04	99	-2.206170	8.10e-05	235
CB2	1.952220	1.40e-05	60	1.952219	7.00e-06	64	1.957493	6.23e-04	44	1.960991	0.00e+00	22
CHACONN1	1.952218	1.40e-05	58	1.952220	1.40e-05	42	1.952172	6.10e-05	35	1.955070	0.00e+00	20
MADSEN	0.616425	9.00e-06	146	0.616425	1.17e-04	40	0.616422	2.61e-04	49	0.613403	3.27e-03	20
HS66	0.518177	3.00e-06	f	0.518158	1.00e-05	37	0.518164	3.50e-05	32	0.518055	1.87e-03	19
MINMAXBD	115.706237	5.45e-04	926	115.705937	1.72e-03	779	115.705316	2.33e-03	919	122.679517	0.00e+00	719
CANTILVR	1.339958	8.00e-06	149	1.339959	3.30e-05	151	1.338856	2.67e-03	91	1.342286	4.59e-03	111
HS92	1.244708	2.58e-04	f	1.112153	6.73e-04	140	1.244708	2.58e-04	f	1.109888	7.50e-04	68
HALDMADS	0.000133	1.40e-05	101	0.000179	1.40e-05	f	0.000133	1.40e-05	101	0.000135	5.10e-05	77
HS34	-0.834035	6.00e-06	52	-0.834035	3.20e-05	32	-0.833901	0.00e+00	36	-0.834014	0.00e+00	25
HS90	2.061544	1.20e-05	f	1.049278	1.07e-03	118	2.061544	1.20e-05	f	1.049423	1.07e-03	72
SNAKE	-2.275095	2.12e-04	87	-0.032419	3.00e-06	f	-2.269176	2.34e-04	69	-0.032419	3.00e-06	f
HS72	702.418590	7.31e-04	f	528.392603	7.94e-03	f	702.418590	7.31e-04	f	528.392603	7.94e-03	f
GIGOMeZ2	1.952237	0.00e+00	81	1.952209	1.80e-05	f	1.952214	1.60e-03	49	1.952439	0.00e+00	31
SYNTHES1	0.759275	6.00e-06	122	0.759278	2.40e-05	113	0.758501	1.34e-04	56	0.839094	0.00e+00	29
HS88	1.324205	4.40e-05	f	1.180569	3.57e-04	67	1.324205	4.40e-05	f	1.177325	3.68e-04	47
HS13	0.954433	1.20e-05	331	1.029338	5.00e-06	f	1.089502	0.00e+00	36	1.138525	0.00e+00	19
MATRIX2	0.000012	9.00e-06	135	0.000038	6.00e-06	f	0.000012	9.00e-06	135	0.000038	6.00e-06	f
WOMFLET	0.001388	1.00e-06	f	-0.000003	2.33e-03	80	0.017147	1.63e-03	90	0.001884	1.73e-04	68
S365	0.000000	6.32e-04	54	0.000000	5.30e-03	36	0.000000	6.32e-04	54	0.000000	5.30e-03	36

Table A.33. Number of function evaluations to obtain (2.4.5) for noise level $\sigma_f = 10^{-5}$

	$\tau = 10^{-6}$						$\tau = 10^{-2}$					
	KNITRO			COBYLA			KNITRO			COBYLA		
problem	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals
CB3	2.000135	7.50e-05	57	2.001105	0.00e+00	f	1.991529	1.86e-02	25	1.999922	9.37e-04	17
CHACONN2	2.000382	1.20e-04	30	2.000360	1.84e-03	f	1.989816	2.59e-02	25	2.007665	3.71e-02	19
GIGOMEZ3	2.000135	7.50e-05	f	1.999908	3.32e-04	35	2.000135	7.50e-05	f	1.999908	3.32e-04	35
HS100	680.643454	6.18e-04	118	680.653293	2.80e-03	f	680.862278	2.56e-03	47	680.751163	1.19e-02	46
DIPIGRI	680.642602	5.53e-04	146	680.653238	2.48e-03	f	680.858601	0.00e+00	47	680.938844	1.37e-02	46
HS93	135.324499	1.03e-03	126	136.540776	7.65e-03	f	135.323447	3.39e-03	35	136.540776	7.65e-03	f
HS64	6294.941500	2.18e-03	74	6307.172876	1.82e-01	f	6746.496044	5.01e-02	20	6307.172876	1.82e-01	f
POLAK3	5.937398	1.34e-03	f	5.932593	8.61e-03	1645	5.954694	7.66e-03	252	5.909504	3.84e-02	1510
POLAK1	2.719811	2.51e-04	86	2.752280	9.16e-04	f	2.723341	7.78e-04	36	2.736065	3.33e-02	231
HS104	3.981084	1.04e-03	174	4.087923	2.80e-02	f	3.981888	9.32e-04	154	4.087923	2.80e-02	f
HS100MOD	678.700188	3.91e-04	f	678.688238	0.00e+00	99	678.964451	3.27e-03	73	678.817053	2.85e-02	61
TWOBARS	1.508513	1.28e-03	f	1.507091	1.04e-03	27	1.508513	1.28e-03	f	1.507577	1.34e-02	21
HS85	-2.209889	1.52e-04	271	-1.733123	1.41e-03	f	-2.211306	1.94e-02	127	-1.733123	1.41e-03	f
CB2	1.952771	0.00e+00	f	1.952216	1.02e-03	45	1.943877	1.91e-02	51	1.956972	0.00e+00	23
CHACONN1	1.956186	0.00e+00	f	1.952905	7.34e-04	23	1.949855	4.28e-02	46	1.934334	2.62e-02	15
MADSEN	0.616321	4.89e-04	f	0.616142	3.03e-04	35	0.619826	0.00e+00	68	0.612382	1.21e-02	26
HS66	0.518819	0.00e+00	f	0.517640	1.21e-03	48	0.518819	0.00e+00	f	0.518165	1.35e-02	13
MINMAXBD	115.710191	2.96e-04	f	115.706307	4.03e-02	756	115.703385	1.16e-02	538	121.478546	2.68e-02	717
CANTILVR	1.341675	0.00e+00	141	1.347345	3.04e-03	f	1.343901	5.05e-03	84	1.336533	4.03e-02	145
HS92	0.003678	1.33e-01	f	0.507997	2.58e-02	f	0.003678	1.33e-01	f	0.507997	2.58e-02	f
HALDMADS	0.007637	0.00e+00	f	0.000616	9.93e-04	71	0.007637	0.00e+00	f	0.000616	9.93e-04	71
HS34	-0.833608	0.00e+00	71	-0.834809	1.73e-03	f	-0.832197	0.00e+00	36	-0.832210	0.00e+00	22
HS90	0.001271	1.33e-01	f	0.683684	1.11e-02	f	0.001271	1.33e-01	f	0.683684	1.11e-02	f
SNAKE	-45.074608	2.90e-03	130	-0.035633	3.27e-04	f	-44.926682	3.22e-02	60	-0.035633	3.27e-04	f
HS72	349.372500	4.62e-02	f	66.439486	3.47e-01	f	349.372500	4.62e-02	f	66.439486	3.47e-01	f
GIGOMeZ2	1.951903	6.68e-04	f	1.951310	1.29e-03	48	1.951903	6.68e-04	f	1.951561	7.74e-04	36
SYNTHES1	0.766084	0.00e+00	76	0.755256	1.12e-03	f	0.790862	1.53e-03	56	0.844177	0.00e+00	29
HS88	0.958819	2.05e-03	f	0.575949	1.82e-02	45	0.958819	2.05e-03	f	0.576448	1.81e-02	26
HS13	0.857313	4.07e-04	48	1.114815	0.00e+00	f	0.857380	4.06e-04	38	1.114815	0.00e+00	f
MATRIX2	0.003633	7.30e-04	f	0.002521	3.17e-04	68	0.003633	7.30e-04	f	0.002521	3.17e-04	68
WOMFLET	0.008386	4.53e-04	f	0.003122	0.00e+00	108	0.007706	5.15e-02	69	-0.051536	5.38e-02	87
S365	0.000000	4.03e-02	45	0.000000	5.11e-02	32	0.000000	4.03e-02	45	0.000016	4.64e-02	25

Table A.34. Number of function evaluations to obtain (2.4.5) for noise level $\sigma_f = 10^{-3}$.

	$\tau = 10^{-6}$						$\tau = 10^{-2}$					
	KNITRO			COBYLA			KNITRO			COBYLA		
problem	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals	$\phi(x)$	feaserr	#fevals
CB3	2.151366	0.00e+00	200	2.297095	3.71e-02	f	2.151799	0.00e+00	183	2.297095	3.71e-02	f
CHACONN2	1.917274	2.54e-01	f	0.544480	1.53e+00	f	1.917274	2.54e-01	f	0.544480	1.53e+00	f
GIGOMEZ3	2.162155	0.00e+00	f	1.558092	5.89e-01	f	2.162155	0.00e+00	f	1.554036	5.32e-01	20
HS100	682.785696	1.01e-01	f	680.545842	2.09e-01	f	682.785696	1.01e-01	f	680.679716	3.84e-01	25
DIPIGRI	682.612679	1.38e-01	f	681.544064	0.00e+00	49	682.612679	1.38e-01	f	681.282921	7.16e-02	39
HS93	132.388506	1.13e-01	126	135.053154	4.58e-02	f	132.388506	1.13e-01	126	135.053154	4.58e-02	f
HS64	7178.300921	4.96e-01	33	13921.561135	4.97e+00	f	6221.151364	5.05e-01	27	13921.561135	4.97e+00	f
POLAK3	5.768461	5.56e-01	f	-2.341289	8.72e+00	f	5.768461	5.56e-01	f	-2.341289	8.72e+00	f
POLAK1	5.342043	6.74e-02	86	7.870717	9.39e-02	f	5.381476	1.35e-01	62	7.870717	9.39e-02	f
HS104	3.287811	2.81e-01	f	1.208522	3.70e-01	30	3.287811	2.81e-01	f	1.193347	3.97e-01	10
HS100MOD	680.016339	1.77e-01	f	679.878287	7.18e-02	45	680.111489	8.61e-02	53	679.599540	2.77e-01	34
TWOBARS	1.601395	1.86e-02	f	1.450060	1.40e-01	30	1.601395	1.86e-02	f	1.450155	1.40e-01	12
HS85	-1.288983	0.00e+00	30	-1.256810	0.00e+00	f	-1.288983	0.00e+00	30	-1.256810	0.00e+00	f
CB2	2.157319	1.15e-02	f	0.871901	1.10e+00	f	2.157319	1.15e-02	f	0.871901	1.10e+00	f
CHACONN1	2.085730	2.43e-03	f	1.976579	2.38e-02	27	2.085730	2.43e-03	f	1.979692	2.45e-02	12
MADSEN	0.874476	0.00e+00	70	0.882487	0.00e+00	f	0.874516	0.00e+00	33	0.882487	0.00e+00	f
HS66	0.543480	0.00e+00	f	0.509050	4.56e-02	37	0.543480	0.00e+00	f	0.508822	4.63e-02	20
MINMAXBD	-33.388429	4.65e+02	f	112.975518	2.81e+00	f	-33.388429	4.65e+02	f	112.975518	2.81e+00	f
CANTILVR	1.531159	5.74e-02	f	0.934009	2.47e+00	f	1.531159	5.74e-02	f	0.934009	2.47e+00	f
HS92	0.366123	1.35e-01	f	0.293398	7.11e-02	48	0.366123	1.35e-01	f	0.297797	7.09e-02	35
HALDMADS	0.252628	0.00e+00	f	-3.027998	1.82e+01	f	0.252628	0.00e+00	f	-3.027998	1.82e+01	f
HS34	-0.141457	1.15e-01	49	-0.750941	2.99e-01	f	-0.140985	1.15e-01	25	-0.750941	2.99e-01	f
HS90	0.790027	1.40e-01	f	0.331080	1.20e-01	48	0.790027	1.40e-01	f	0.331906	1.20e-01	13
SNAKE	-8.350669	1.87e-01	93	1.284550	3.65e-01	f	-8.350669	1.87e-01	93	1.284550	3.65e-01	f
HS72	67.968573	5.76e-01	f	8.946566	3.26e+00	f	67.968573	5.76e-01	f	8.946566	3.26e+00	f
GIGOMeZ2	1.874401	1.78e-01	78	2.056211	0.00e+00	f	1.874484	1.90e-01	70	2.056211	0.00e+00	f
SYNTHES1	1.977343	2.09e-01	f	1.855036	0.00e+00	11	1.841763	3.65e-01	69	1.855036	0.00e+00	11
HS88	0.047086	1.26e-01	16	0.235414	1.47e-01	f	0.047086	1.26e-01	16	0.235414	1.47e-01	f
HS13	0.357703	6.52e-02	48	2.381364	0.00e+00	f	0.551282	1.71e-02	19	2.381364	0.00e+00	f
MATRIX2	0.075268	2.95e-02	194	1.186858	0.00e+00	f	0.075268	2.95e-02	194	1.186858	0.00e+00	f
WOMFLET	6.254269	5.05e-02	f	3.338088	0.00e+00	48	6.254269	5.05e-02	f	3.336520	0.00e+00	35
S365	0.000000	9.22e-02	45	0.000000	5.72e-02	24	0.000000	9.22e-02	45	-0.001477	1.53e-01	23

Table A.35. Number of function evaluations to obtain (2.4.5) for noise level $\sigma_f = 10^{-1}$.

APPENDIX B

Adaptive Finite-Difference Interval Estimation for Noisy Derivative-Free Optimization

B.1. Finite-Difference Formula Derivation and Tables

We summarize the different standard finite-difference schemes with equidistant points, their theoretical error, optimal steplength, and optimal error in terms of the noise level ϵ_f and local bound on the q-th derivative L_q for a smooth univariate function $\phi : \mathbb{R} \to \mathbb{R}$ in Tables B.1 and B.2. For completeness, we provide a complete derivation of the errors for a generic finite-difference approximation to the d-th order derivative below.

We will use $f : \mathbb{R} \to \mathbb{R}$ to denote the noisy function evaluations $f(t) = \phi(t) + \epsilon(t)$. We will consider two settings for $\epsilon(t)$: (1) we will assume that $\epsilon(t)$ is bounded, i.e., $|\epsilon(t)| \leq \epsilon_f$ for all t; (2) we will assume that $\epsilon(t)$ is a random variable with $\mathbb{E}[\epsilon(t)] = 0$ and $\mathbb{E}[\epsilon(t)^2] = \sigma_f^2$ for all t. The tables vary the number of evaluated points m and is dependent on the local Lipschitz constant $L_q \geq 0$ which bounds the q-th derivative

$$|\phi^{(q)}(t + h_0 s)| \leq L_q$$

for all $s \in [s_1, s_m]$, where q is the order of the remainder term in the Taylor expansion.

In the most general case, given distinct shifts $\{s_j\}_{j=1}^m$ and points $\{t_1, ..., t_m\} = \{t + hs_1, t + hs_2, ..., t + hs_m\}$, one can derive a generic finite-difference method to approximate

the d-th derivative of the form:

$$\phi^{(d)}(t) \approx \frac{\sum_{j=1}^{m} w_j f(t + s_j h)}{h^d} = f^{(d)}(t; h).$$

We will assume without loss of generality that $s_1 < s_2 < ... < s_m$. First, note that $f^{(d)}$ can be decomposed into a noiseless finite-difference formula and its corresponding error:

$$f^{(d)}(t; h) = \frac{\sum_{j=1}^{m} w_j \phi(t + s_j h)}{h^d} + \frac{\sum_{j=1}^{m} w_j \epsilon(t + s_j h)}{h^d}.$$

Considering the noiseless finite-difference term, since the function is smooth, one can write the Lagrange remainder form of the Taylor series expansions for each function evaluation without noise as:

$$\phi(t + h s_j) = \sum_{l=0}^{q-1} \frac{1}{l!} \phi^{(l)}(t) s_j^l + \frac{1}{q!} \phi^{(q)}(\xi_j) s_j^q$$

for $\xi_j \in [t, t + h s_j]$ for $j = 1, ..., m$. Therefore, if the weights w satisfy

$$\frac{1}{d!} \sum_{j=1}^{m} w_j s_j^l = \begin{cases} 0 & \text{for } l \neq d,\ l = 0, 1, ..., q-1 \\ 1 & \text{for } l = d \end{cases}$$

then

$$\frac{\sum_{j=1}^{m} w_j \phi(t + s_j h)}{h^d} = \phi^{(d)}(t) + \frac{h^{q-d}}{q!} \sum_{j=1}^{m} w_j \phi^{(q)}(\xi_j) s_j^q.$$

This can be written compactly by the linear system of equations:

$$V(s)^T w = d! \cdot e_{p-d}$$

where $V(s) \in \mathbb{R}^{m \times q}$ is the Vandermonde matrix defined as

$$V(s) = \begin{bmatrix} s_1^{q-1} & s_1^{q-2} & \dots & s_1^0 \\ s_2^{q-1} & s_2^{q-2} & \dots & s_2^0 \\ \vdots & \vdots & \ddots & \vdots \\ s_m^{q-1} & s_m^{q-2} & \dots & s_m^0 \end{bmatrix}$$

and $e_{p-d} \in \mathbb{R}^p$ is the $(p-d)$-th coordinate vector.

To derive a reasonable bound on the total error, suppose we are given $h_0 > 0$ and a bound on $\phi^{(q)}$

$$|\phi^{(q)}(t + sh_0)| \leq L_q$$

for all $s \in [s_1, s_m]$. If we assume that the error is bounded, i.e., $|\epsilon(t)| \leq \epsilon_f$, then one can then bound the error in the approximation by:

$$|f^{(d)}(t;h) - \phi^{(d)}(t)| \leq \frac{L_q h^{q-d}}{q!} \sum_{j=1}^{m} |w_j s_j^q| + \frac{\|w\|_1 \epsilon_f}{h^d} = \epsilon_g(h)$$

for all $0 < h \leq h_0$. If we assume instead that $\mathrm{Var}(\epsilon(t)) = \sigma_f^2$, then we can similarly show

$$\mathbb{E}[(f^{(d)}(t;h) - \phi^{(d)}(t))^2] \leq \frac{L_q^2 h^{2(q-d)}}{(q!)^2} \sum_{j=1}^{m} w_j^2 s_j^{2q} + \frac{\|w\|_2^2 \sigma_f^2}{h^{2d}} = \sigma_g^2(h)$$

for all $0 < h \leq h_0$.

The above Taylor series analysis is pessimistic in that it requires multiple ξ_j points, and therefore yields a loose bound when applying the triangle inequality. Instead, one can consider the derivation of finite-difference schemes for approximating the *first* derivative

at an interpolation point using Lagrange polynomials, which yields a tighter bound on the error.

As above, suppose we are given distinct points $\{t_1, ..., t_m\} = \{t + hs_1, ..., t + hs_m\}$ and we are interested in approximating $\phi^{(1)}(t)$. Recall that the Lagrange basis polynomials are defined as:

$$\psi_{p,j}(\tilde{t}) = \frac{\prod_{k \neq j}(\tilde{t} - t_k)}{\prod_{k \neq j}(t_j - t_k)} = \frac{\omega_m(\tilde{t})}{\omega_m^{(1)}(t_j)(\tilde{t} - t_j)}, \ \omega_m(\tilde{t}) = \prod_{j=1}^{m}(\tilde{t} - t_j).$$

Then the Lagrange interpolation is defined as:

$$\ell(\tilde{t}) = \sum_{j=1}^{m} \psi_{m,j}(\tilde{t})\phi(t_j).$$

It is well-known that the remainder is

$$\phi(\tilde{t}) - \ell(\tilde{t}) = \frac{\omega_m(\tilde{t})}{m!}\phi^{(m)}(\xi)$$

for some $\xi \in [t_1, t_m]$. Note that the finite-difference formula can simply be obtained by differentiating the Lagrange polynomial

$$\ell^{(1)}(\tilde{t}) = \sum_{j=1}^{m} \psi_{m,j}^{(1)}(\tilde{t})\phi(t_j).$$

Therefore, the finite-difference coefficients are obtained by evaluating $\psi_{m,j}^{(1)}(\tilde{t})$. The error is also obtained by noting

$$\phi^{(1)}(\tilde{t}) = \ell^{(1)}(\tilde{t}) + \frac{\omega_m^{(1)}(\tilde{t})}{m!}\phi^{(m)}(\xi) + \frac{\omega_m(\tilde{t})}{m!}\phi^{(m)}(\xi)\frac{d\xi}{dx}.$$

Since

$$\omega_m^{(1)}(\tilde{t}) = \sum_{j=1}^{m} \prod_{k \neq j} (\tilde{t} - t_k),$$

plugging in $\tilde{t} = t_i$ for any $i = 1, ..., m$, we get the following equality

$$\phi^{(1)}(t_i) = \ell^{(1)}(t_i) + \frac{\omega_m^{(1)}(t_i)}{m!}\phi^{(m)}(\xi) = \ell^{(1)}(t_i) + \prod_{j \neq i}(t_i - t_j)\frac{\phi^{(m)}(\xi)}{m!}.$$

Given $h_0 > 0$ and a bound on $\phi^{(m)}$

$$|\phi^{(m)}(t + h_0 s)| \leq L_m$$

for all $s \in [s_1, s_m]$ and assuming $t = t_i$ is one of the interpolation points, we obtain the bound

$$|\phi^{(1)}(t) - \ell^{(1)}(t)| \leq \frac{L_m h^{m-1}}{m!} \left| \prod_{j \neq i} s_j \right|$$

and if we incorporate the error in the function evaluations, we obtain a error and variance bounds of

$$|f^{(1)}(t; h) - \phi^{(1)}(t)| \leq \frac{L_m h^{m-1}}{m!} \left| \prod_{j \neq i} s_j \right| + \frac{\|w\|_1 \epsilon_f}{h} = \epsilon_g(h)$$

$$\mathbb{E}[(f^{(1)}(t; h) - \phi^{(1)}(t))^2] \leq \frac{L_m^2 h^{2(m-1)}}{(m!)^2} \prod_{j \neq i} s_j^2 + \frac{\|w\|_2^2 \sigma_f^2}{h^2} = \sigma_g^2(h)$$

for all $0 < h \leq h_0$.

Table B.1. Table containing the finite-difference formula, deterministic error bound $|f^{(1)}(t;h) - \phi^{(1)}(t)| \leq \epsilon_g(h)$ for generic h, optimal h^*, and optimal error $\epsilon_g(h^*)$ for forward-difference schemes with number of points $m \in \{2,3,4,5\}$.

m	$f^{(1)}(t;h)$	$\epsilon_g(h)$	h^*	$\epsilon_g(h^*)$
2	$\frac{f(t+h)-f(t)}{h}$	$\frac{L_2 h}{2} + \frac{2\epsilon_f}{h}$	$2\sqrt{\frac{\epsilon_f}{L_2}}$	$2\sqrt{L_2 \epsilon_f}$
3	$\frac{-3f(t)+4f(t+h)-f(t+2h)}{2h}$	$\frac{L_3 h^2}{3} + \frac{4\epsilon_f}{h}$	$\sqrt[3]{\frac{6\epsilon_f}{L_3}}$	$6^{2/3} L_3^{1/3} \epsilon_f^{2/3}$
4	$\frac{-11f(t)+18f(t+h)-9f(t+2h)+2f(t+3h)}{6h}$	$\frac{L_4 h^3}{4} + \frac{20\epsilon_f}{3h}$	$\sqrt[4]{\frac{80\epsilon_f}{9L_4}}$	$\frac{8 \cdot 5^{3/4}}{3\sqrt{3}} L_4^{1/4} \epsilon_f^{3/4}$
5	$\frac{-25f(t)+48f(t+h)-36f(t+2h)+16f(t+3h)-3f(t+4h)}{12h}$	$\frac{L_5 h^4}{5} + \frac{32\epsilon_f}{3h}$	$\sqrt[5]{\frac{40\epsilon_f}{3L_5}}$	$4\left(\frac{5}{3}\right)^{4/5} 2^{2/5} L_5^{1/5} \epsilon_f^{4/5}$

Table B.2. Table containing the finite-difference formula, deterministic error bound $|f^{(1)}(t;h) - \phi^{(1)}(t)| \leq \epsilon_g(h)$ for generic h, optimal h^*, and optimal error $\epsilon_g(h^*)$ for central-difference schemes with number of points $m \in \{2,4,6\}$.

m	$f^{(1)}(t;h)$	$\epsilon_g(h)$	h^*	$\epsilon_g(h^*)$
2	$\frac{f(t+h)-f(t-h)}{2h}$	$\frac{L_3 h^2}{6} + \frac{\epsilon_f}{h}$	$\sqrt[3]{\frac{3\epsilon_f}{L_3}}$	$\frac{3^{2/3}}{2} L_3^{1/3} \epsilon_f^{2/3}$
4	$\frac{f(t-2h)-8f(t-h)+8f(t+h)-f(t+2h)}{12h}$	$\frac{L_5 h^4}{30} + \frac{3\epsilon_f}{2h}$	$\sqrt[5]{\frac{45\epsilon_f}{4L_5}}$	$\frac{1}{4}\left(\frac{3}{2}\right)^{4/5} 5^{4/5} L_5^{1/5} \epsilon_f^{4/5}$
6	$\frac{-f(t-3h)+9f(t-2h)-45f(t-h)+45f(t+h)-9f(t+2h)+f(t+3h)}{60h}$	$\frac{L_7 h^6}{140} + \frac{11\epsilon_f}{6h}$	$\sqrt[7]{\frac{385\epsilon_f}{9L_7}}$	$\frac{77^{6/7}}{12 \cdot 3^{5/7} \cdot \sqrt[7]{5}} L_7^{1/7} \epsilon_f^{6/7}$

Table B.3. Table containing the finite-difference formula, MSE error bound $\mathbb{E}[(f^{(1)}(t;h)-\phi^{(1)}(t))^2] \leq \sigma_g^2(h)$ for generic h, optimal h^*, and optimal error $\sigma_g(h^*)$ for forward-difference schemes with number of points $m \in \{2,3,4,5\}$.

p	$f^{(1)}(t;h)$	$\sigma_g^2(h)$	h^*	$\sigma_g(h^*)$
1	$\frac{f(t+h)-f(t)}{h}$	$\frac{L_2^2h^2}{4}+\frac{2\epsilon_f^2}{h^2}$	$8^{1/4}\sqrt{\frac{\epsilon_f}{L_2}}$	$2^{1/4}\sqrt{L_2\epsilon_f}$
2	$\frac{-3f(t)+4f(t+h)-f(t+2h)}{2h}$	$\frac{L_3^2h^4}{9}+\frac{13\epsilon_f^2}{2h^2}$	$(\frac{3}{2})^{1/3}13^{1/6}\sqrt[3]{\frac{\epsilon_f}{L_3}}$	$\frac{\sqrt[6]{3}\sqrt[3]{13}}{2^{2/3}}L_3^{1/3}\epsilon_f^{2/3}$
3	$\frac{-11f(t)+18f(t+h)-9f(t+2h)+2f(t+3h)}{6h}$	$\frac{L_4^2h^6}{16}+\frac{265\epsilon_f^2}{18h^2}$	$(\frac{2}{3})^{3/8}265^{1/8}\sqrt[4]{\frac{\epsilon_f}{L_4}}$	$\frac{1}{3}\sqrt[8]{\frac{2}{3}}265^{3/8}L_4^{1/4}\epsilon_f^{3/4}$
4	$\frac{-25f(t)+48f(t+h)-36f(t+2h)+16f(t+3h)-3f(t+4h)}{12h}$	$\frac{L_5^2h^8}{25}+\frac{2245\epsilon_f^2}{72h^2}$	$\frac{5^{3/10}449^{1/10}}{\sqrt{2}\sqrt[5]{3}}\sqrt[5]{\frac{\epsilon_f}{L_5}}$	$\frac{5^{7/10}449^{2/5}}{4\cdot3^{4/5}}L_5^{1/5}\epsilon_f^{4/5}$

Table B.4. Table containing the finite-difference formula, MSE error bound $\mathbb{E}[(f^{(1)}(t;h)-\phi^{(1)}(t))^2] \leq \sigma_g^2(h)$ for generic h, optimal h^*, and optimal error $\sigma_g(h^*)$ for central-difference schemes with number of points $m \in \{2,4,6\}$.

p	$f^{(1)}(t;h)$	$\sigma_g^2(h)$	h^*	$\sigma_g(h^*)$
2	$\frac{f(t+h)-f(t-h)}{2h}$	$\frac{L_3^2h^4}{36}+\frac{\epsilon_f^2}{2h^2}$	$\sqrt[3]{3}\sqrt[3]{\frac{\epsilon_f}{L_3}}$	$\frac{\sqrt[6]{3}}{2}L_3^{1/3}\epsilon_f^{2/3}$
4	$\frac{f(t-2h)-8f(t-h)+8f(t+h)-f(t+2h)}{12h}$	$\frac{L_5^2h^8}{900}+\frac{65\epsilon_f^2}{72h^2}$	$(\frac{5}{2})^{3/10}13^{1/10}\sqrt[5]{\frac{\epsilon_f}{L_5}}$	$\frac{5^{7/10}\cdot13^{2/5}}{12\cdot\sqrt[5]{2}}L_5^{1/5}\epsilon_f^{4/5}$
6	$\frac{-f(t-3h)+9f(t-2h)-45f(t-h)+45f(t+h)-9f(t+2h)+f(t+3h)}{60h}$	$\frac{L_7^2h^{12}}{140^2}+\frac{2107\epsilon_f^2}{1800h^2}$	$\frac{7^{2/7}43^{1/14}}{3^{3/14}}\sqrt[7]{\frac{\epsilon_f}{L_7}}$	$\frac{7^{17/14}43^{3/7}}{60\cdot3^{2/7}}L_7^{1/7}\epsilon_f^{6/7}$

B.2. Complete Experimental Results

Here, we present the complete experimental results from Section 3.4.

B.2.1. Robustness to Different Noise Levels

We test our procedure on a simple function $\phi(t) = \cos(t)$ for different noise levels using different schemes listed in Table 3.1. These are shown in Figure B.1. Detailed numerical results, including the number of iterations and relative error, are listed in Table B.5.

Observe that our method is able to consistently achieve low relative error using a similar number of function evaluations across all tested noise levels. This is a desirable property, as it demonstrates that our initial choice of the interval h and our method is consistent over different noise levels.

B.2.2. Affine Invariance

One advantage of our proposed method is that the testing ratio remains unchanged under affine transformations of the function. It is particularly obvious that our procedure is invariant when adding a constant to the function. Hence, we focus on transformations of the form $\phi(t) \to a \cdot \phi(b \cdot t)$ for some $a, b \neq 0$.

To do this, we test Algorithm 3.2 on the function $\phi(t) = a \cdot \sin(b \cdot t)$ at $t = 0$ for various a and b. We fix the noise level to be $\epsilon_f = 10^{-3}$. The results are shown in Figure B.2. Detailed results can be found in Table B.6 and B.7. As seen in Figure B.2, our method is affine-invariant and can output consistently correct results for different a and b.

Table B.5. Detailed results for $\phi(t) = \cos(t)$ with different noise levels; r represents the final testing ratio; h^* is the h that minimizes $\delta_S(h; \phi, t, \epsilon_f)$ reported by `minimize_scalar` function in `scipy.optimize` and could be unreliable.

scheme	$h_\dagger$	h^*	r	`#iters`	`#Evals`	relative error	ϵ_f
FD	2.00e-04	2.72e-04	2.08	1	3	1.86e-05	1.00e-08
FD	6.32e-04	8.59e-04	2.00	1	3	4.21e-05	1.00e-07
FD	3.50e-03	2.73e-03	4.79	4	8	1.17e-03	1.00e-06
FD	6.32e-03	8.64e-03	1.77	1	3	1.72e-03	1.00e-05
FD	2.00e-02	2.76e-02	2.00	1	3	1.69e-03	1.00e-04
FD	6.32e-02	9.05e-02	1.73	1	3	5.07e-04	1.00e-03
FD	5.00e-01	1.73e+00	3.89	3	6	9.86e-02	1.00e-02
FD	6.32e-01	8.26e+00	1.52	1	3	2.97e-01	1.00e-01
CD	3.11e-03	3.29e-03	2.40	1	4	1.89e-06	1.00e-08
CD	6.69e-03	7.09e-03	2.66	1	4	5.39e-06	1.00e-07
CD	1.44e-02	1.53e-02	2.72	1	4	9.35e-06	1.00e-06
CD	3.11e-02	3.29e-02	2.05	1	4	3.34e-04	1.00e-05
CD	6.69e-02	7.09e-02	2.18	1	4	1.33e-03	1.00e-04
CD	1.44e-01	1.53e-01	2.55	1	4	3.32e-03	1.00e-03
CD	3.11e-01	3.30e-01	1.89	1	4	3.84e-02	1.00e-02
CD	6.69e-01	7.74e+00	2.01	1	4	5.71e-02	1.00e-01
FD_3P	3.91e-03	4.14e-03	2.88	1	5	1.01e-05	1.00e-08
FD_3P	8.43e-03	8.92e-03	3.24	1	5	2.01e-05	1.00e-07
FD_3P	1.82e-02	1.92e-02	2.76	1	5	2.05e-04	1.00e-06
FD_3P	3.91e-02	4.11e-02	3.28	1	5	5.82e-04	1.00e-05
FD_3P	8.43e-02	8.77e-02	2.86	1	5	5.65e-03	1.00e-04
FD_3P	1.82e-01	1.86e-01	3.10	1	5	2.29e-02	1.00e-03
FD_3P	3.91e-01	2.99e+00	2.88	1	5	9.96e-02	1.00e-02
FD_3P	2.53e+00	2.12e+01	5.75	2	7	4.17e-01	1.00e-01
FD_4P	2.16e-02	2.04e-02	9.36	5	18	2.14e-06	1.00e-08
FD_4P	4.61e-02	3.67e-02	16.51	4	13	1.14e-05	1.00e-07
FD_4P	8.19e-02	4.76e-01	10.80	4	13	9.43e-05	1.00e-06
FD_4P	1.70e-01	1.25e-01	4.40	5	18	6.15e-04	1.00e-05
FD_4P	2.59e-01	3.28e+00	14.62	4	13	7.80e-04	1.00e-04
FD_4P	3.07e-01	3.28e+00	4.23	1	6	5.46e-03	1.00e-03
FD_4P	5.46e-01	8.78e+00	6.44	1	6	3.19e-02	1.00e-02
FD_4P	2.91e+00	8.79e+00	4.28	2	8	9.55e-01	1.00e-01
CD_4P	4.08e-02	4.22e-02	2.52	1	6	1.16e-07	1.00e-08
CD_4P	6.46e-02	6.69e-02	2.04	1	6	8.34e-07	1.00e-07
CD_4P	1.02e-01	1.06e-01	1.95	1	6	8.81e-06	1.00e-06
CD_4P	1.62e-01	1.68e-01	1.79	1	6	4.29e-05	1.00e-05
CD_4P	2.57e-01	2.67e-01	1.71	1	6	4.00e-04	1.00e-04
CD_4P	4.08e-01	4.25e-01	1.89	1	6	8.32e-04	1.00e-03
CD_4P	8.07e-01	7.97e+00	4.83	4	20	5.87e-03	1.00e-02
CD_4P	1.54e+00	2.06e+01	4.25	3	14	2.34e-01	1.00e-01
L2_CD	3.29e-02	3.07e-02	3.78	4	15	1.00e-04	1.00e-08
L2_CD	4.68e-02	5.46e-02	1.89	1	5	8.55e-05	1.00e-07
L2_CD	1.04e-01	9.71e-02	4.22	4	15	7.28e-04	1.00e-06
L2_CD	1.48e-01	1.73e-01	1.90	1	5	1.06e-03	1.00e-05
L2_CD	3.29e-01	3.07e-01	4.03	4	15	8.28e-03	1.00e-04
L2_CD	5.85e-01	5.49e-01	3.81	4	15	2.84e-02	1.00e-03
L2_CD	1.04e+00	9.87e-01	3.45	4	15	7.95e-02	1.00e-02
L2_CD	2.96e+00	1.55e+01	5.45	2	7	5.40e-01	1.00e-01

Table B.6. Detailed results for $\phi(t) = a \cdot \sin(b \cdot t)$ with $\epsilon_f =$ `1E-3`; r represents the final testing ratio; h^* is the h that minimizes $\delta_S(h; \phi, t, \epsilon_f)$ reported by `minimize_scalar` function in `scipy.optimize` and could be unreliable.

a	b	scheme	$h_\dagger$	h^*	r	`#iters`	`#Evals`	relative error
0.10	0.10	FD	2.53e+00	3.94e+00	1.89	5	8	1.75e-02
0.10	1.00	FD	3.48e-01	3.94e-01	4.34	6	11	5.29e-02
0.10	10.00	FD	2.77e-02	3.94e-02	2.68	4	8	2.01e-05
1.00	0.10	FD	1.39e+00	1.82e+00	2.61	7	12	1.30e-02
1.00	1.00	FD	1.58e-01	1.82e-01	4.91	3	6	3.01e-03
1.00	10.00	FD	1.58e-02	1.82e-02	4.58	2	4	6.97e-03
10.00	0.10	FD	6.32e-01	8.44e-01	3.23	4	7	4.76e-04
10.00	1.00	FD	6.32e-02	8.44e-02	3.38	1	3	2.83e-04
10.00	10.00	FD	6.92e-03	8.44e-03	3.34	5	9	2.92e-03
0.10	0.10	CD	3.89e+00	3.12e+00	5.47	4	10	2.51e-02
0.10	1.00	CD	2.88e-01	3.12e-01	2.30	3	10	1.38e-02
0.10	10.00	CD	3.20e-02	3.12e-02	3.13	4	14	1.70e-02
1.00	0.10	CD	1.30e+00	1.44e+00	2.17	3	8	2.81e-03
1.00	1.00	CD	1.44e-01	1.44e-01	2.97	1	4	3.46e-03
1.00	10.00	CD	1.60e-02	1.44e-02	4.06	3	10	4.27e-03
10.00	0.10	CD	6.49e-01	6.70e-01	2.73	5	16	7.02e-04
10.00	1.00	CD	7.21e-02	6.70e-02	3.74	4	14	8.66e-04
10.00	10.00	CD	5.34e-03	6.70e-03	1.52	4	12	4.75e-04
0.10	0.10	FD_3P	4.91e+00	8.14e+01	2.93	4	11	2.13e-02
0.10	1.00	FD_3P	5.45e-01	8.14e+00	2.45	2	7	6.48e-02
0.10	10.00	FD_3P	7.57e-02	8.14e-01	4.90	5	15	1.61e-01
1.00	0.10	FD_3P	1.64e+00	2.14e+01	2.25	3	9	8.79e-03
1.00	1.00	FD_3P	1.82e-01	2.14e+00	3.63	1	5	1.86e-04
1.00	10.00	FD_3P	2.02e-02	1.83e-02	4.43	3	9	1.15e-02
10.00	0.10	FD_3P	8.18e-01	8.45e-01	3.48	5	13	1.71e-03
10.00	1.00	FD_3P	9.09e-02	8.45e-02	4.58	4	11	1.77e-03
10.00	10.00	FD_3P	1.01e-02	8.45e-03	6.67	6	15	1.00e-03
0.10	0.10	FD_4P	9.33e+00	8.44e+01	4.95	9	31	1.63e-01
0.10	1.00	FD_4P	9.79e-01	8.44e+00	9.64	8	32	1.50e-01
0.10	10.00	FD_4P	1.96e+00	1.78e+01	7.71	7	29	9.60e-01
1.00	0.10	FD_4P	2.76e+00	3.59e+00	4.47	3	11	3.59e-03
1.00	1.00	FD_4P	3.07e-01	3.59e-01	6.55	1	6	8.16e-03
1.00	10.00	FD_4P	6.62e-01	5.88e+00	10.15	8	35	9.49e-01
10.00	0.10	FD_4P	1.84e+00	2.25e+00	6.57	4	14	3.68e-04
10.00	1.00	FD_4P	2.05e-01	2.71e+00	10.68	3	10	2.90e-04
10.00	10.00	FD_4P	3.07e-01	4.62e+00	16.47	1	6	8.38e-01
0.10	0.10	CD_4P	6.52e+00	7.97e+01	2.12	5	14	5.73e-03
0.10	1.00	CD_4P	6.11e-01	7.97e+00	1.57	3	14	4.45e-03
0.10	10.00	CD_4P	7.64e-02	7.97e-01	4.32	5	18	1.06e-02
1.00	0.10	CD_4P	4.08e+00	4.10e+00	2.30	7	26	9.02e-04
1.00	1.00	CD_4P	4.08e-01	4.10e-01	2.30	1	6	9.02e-04
1.00	10.00	CD_4P	3.82e-02	4.10e-02	1.68	6	20	6.98e-04
10.00	0.10	CD_4P	2.45e+00	2.58e+00	1.89	5	18	1.18e-04
10.00	1.00	CD_4P	2.55e-01	2.58e-01	2.31	4	20	1.39e-04
10.00	10.00	CD_4P	2.55e-02	2.58e-02	2.31	5	14	1.39e-04

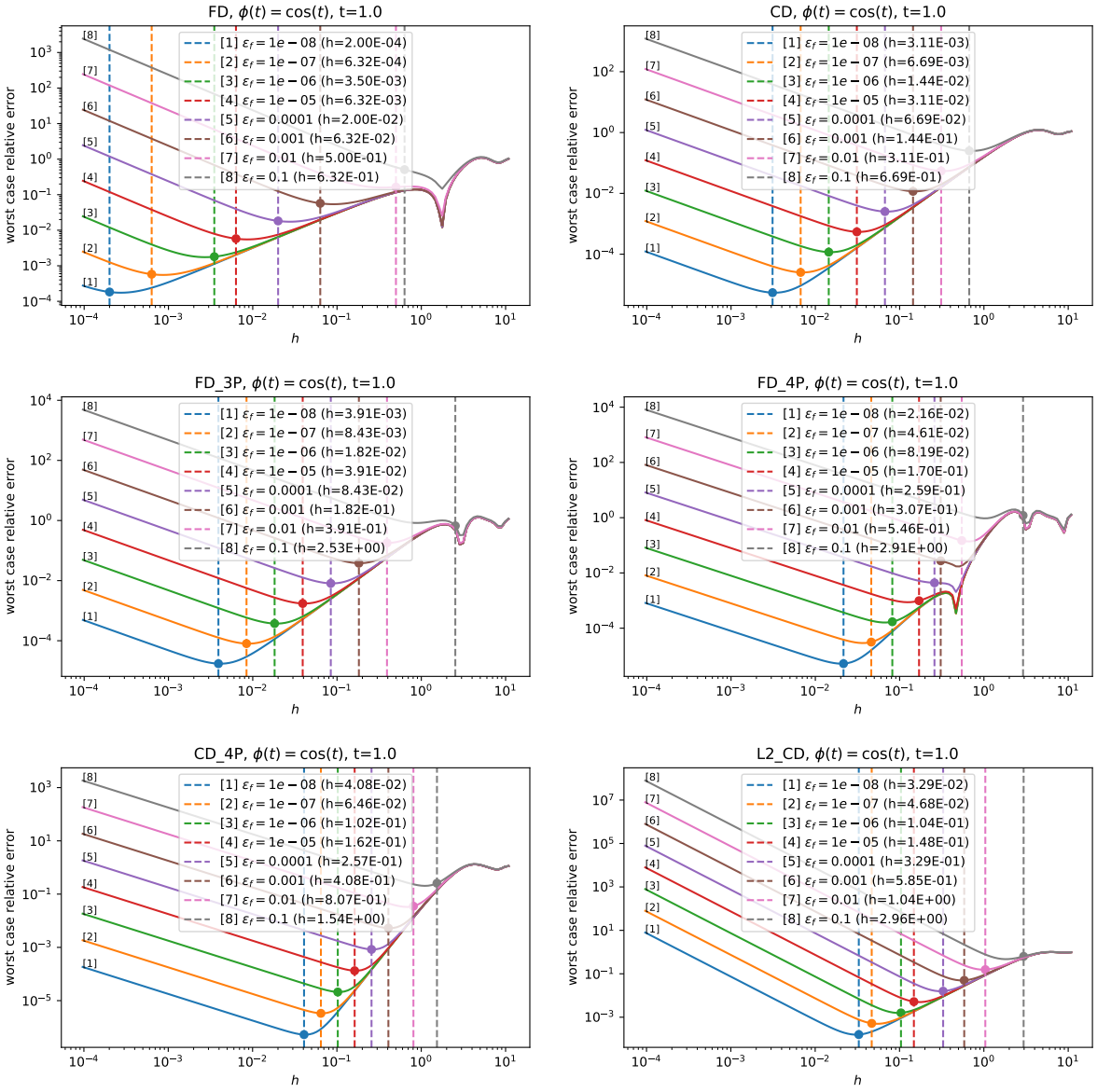

Figure B.1. Worst case relative error $\delta_S(h; \phi, t, \epsilon_f)$ against h on function $\phi(t) = \cos(t)$ with different noise levels; the vertical dashed line represents the $h_\dagger$ output by Algorithm 3.2.

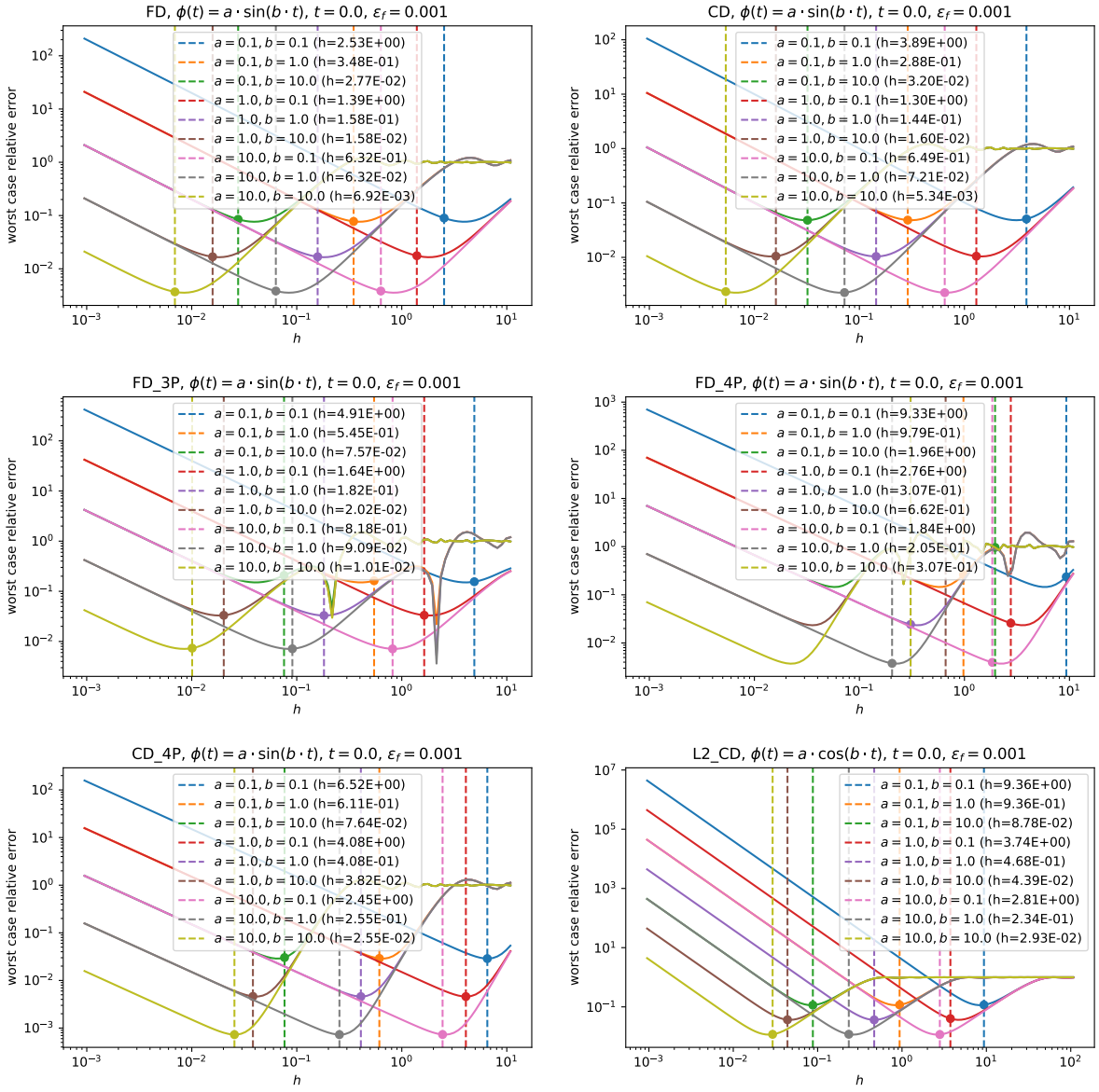

Figure B.2. Worst case relative error $\delta_S(h; \phi, t, \epsilon_f)$ against h on function $\phi(t) = a \cdot \sin(b \cdot t)$ for different a and b; the vertical dashed line represents the $h_\dagger$ output by Algorithm 3.2.

B.2.3. Difficult and Special Examples

Here, we present the full table of results for the examples listed in Section 3.4 in Table B.8 with $\epsilon_f = 10^{-3}$. For reference, the considered problems are:

Table B.7. Detailed results for $\phi(t) = a \cdot \sin(b \cdot t)$ with $\epsilon_f =$ `1E-3`; r represents the final testing ratio; h^* is the h that minimizes $\delta_S(h; \phi, t, \epsilon_f)$ reported by `minimize_scalar` function in `scipy.optimize` and could be unreliable.

a	b	scheme	$h_\dagger$	h^*	r	`#iters`	`#Evals`	relative error
0.10	0.10	`L2_CD`	9.36e+00	8.42e+00	4.39	8	23	5.97e-02
0.10	1.00	`L2_CD`	9.36e-01	8.42e-01	4.79	2	7	4.28e-02
0.10	10.00	`L2_CD`	8.78e-02	8.99e-01	3.28	5	15	5.32e-02
1.00	0.10	`L2_CD`	3.74e+00	4.70e+00	1.99	4	11	9.49e-03
1.00	1.00	`L2_CD`	4.68e-01	4.70e-01	2.57	1	5	2.13e-02
1.00	10.00	`L2_CD`	4.39e-02	4.70e-02	2.15	6	17	1.70e-02
10.00	0.10	`L2_CD`	2.81e+00	2.64e+00	3.59	5	15	7.63e-03
10.00	1.00	`L2_CD`	2.34e-01	2.64e-01	2.62	2	7	6.94e-04
10.00	10.00	`L2_CD`	2.93e-02	2.64e-02	5.02	5	13	3.94e-03

(1) $\phi(t) = (e^t - 1)^2$, at $t = -8$.

(2) $\phi(t) = e^{100t}$, at $t = 0.01$.

(3) $\phi(t) = t^4 + 3t^2 - 10t$, at $t = 0.99999$.

(4) $\phi(t) = 10000t^3 + 0.01t^2 + 5t$, at $t = 10^{-9}$.

B.2.4. Comparison with Moré-Wild Heuristic

We compare our adaptive forward-difference procedure against the Moré-Wild heuristic [71], as described in Section 3.2.1.

First, observe that if function ϕ has (near) central symmetry at t, then Moré-Wild heuristic is very likely to fail. To demonstrate this, we test on $\phi(t) = \sin(t)$ with various value of t close to 0 and different noise levels ϵ_f. The results are summarized in Table B.9.

Next, we test our adaptive procedure and the Moré-Wild heuristic on $\phi(t) = a \cdot (\exp(b \cdot t) - 1)$ at $t = 0$, with a fixed noise level: $\epsilon_f =$ `1E-3`. We summarize our result in Table B.10. Notice that Moré-Wild heuristic may not be able to find a suitable estimation for h, in which case a failure is declared. In such cases, we will report the result as "$--$".

Table B.8. Detailed results for special examples, with $\epsilon_f =$ `1E-3`; r represents the final testing ratio; h^* is the h that minimizes $\delta_S(h; \phi, t, \epsilon_f)$ reported by `minimize_scalar` function in `scipy.optimize` and could be unreliable.

$\phi(t)$	scheme	$h_\dagger$	h^*	r	`#iters`	`#Evals`	relative error
$\left(e^t - 1.0\right)^2$	`FD`	1.01e+00	1.46e+00	4.49	3	5	1.02e+00
$\left(e^t - 1.0\right)^2$	`CD`	1.30e+00	1.53e+00	3.38	3	8	5.73e-02
$\left(e^t - 1.0\right)^2$	`FD_3P`	8.18e-01	3.82e+02	2.28	5	13	6.63e-01
$\left(e^t - 1.0\right)^2$	`FD_4P`	9.21e-01	3.82e+02	4.14	2	8	4.15e+00
$\left(e^t - 1.0\right)^2$	`CD_4P`	1.43e+00	3.82e+02	1.84	5	22	1.11e+00
$\left(e^t - 1.0\right)^2$	`L2_CD`	2.34e+00	8.68e+00	3.03	6	19	3.90e-01
e^{100t}	`FD`	4.32e-04	3.79e-04	3.72	7	11	2.74e-02
e^{100t}	`CD`	1.19e-03	1.03e-03	4.29	7	20	3.09e-03
e^{100t}	`FD_3P`	1.12e-03	3.82e+02	3.08	8	19	6.81e-03
e^{100t}	`FD_4P`	1.90e-03	3.82e+02	6.54	8	23	4.97e-03
e^{100t}	`CD_4P`	3.18e-03	3.82e+02	2.15	8	20	4.60e-04
e^{100t}	`L2_CD`	3.66e-03	3.64e-03	3.01	8	19	1.18e-02
$t^4 + 3t^2 - 10t$	`FD`	1.58e-02	1.48e-02	3.55	2	4	7.97e+02
$t^4 + 3t^2 - 10t$	`CD`	4.81e-02	5.00e-02	2.94	2	6	2.15e+01
$t^4 + 3t^2 - 10t$	`FD_3P`	6.06e-02	6.16e-02	3.64	2	7	2.38e+02
$t^4 + 3t^2 - 10t$	`FD_4P`	1.54e-01	1.39e-01	11.90	4	13	1.93e+02
$t^4 + 3t^2 - 10t$	`CD_4P`	9.39e+02	4.87e+03	1.62	16	48	2.71e-03
$t^4 + 3t^2 - 10t$	`L2_CD`	2.34e-01	2.11e-01	4.53	2	7	5.38e-03
$10000t^3 + 0.01t^2 + 5t$	`FD`	3.95e-03	4.64e-03	4.36	3	5	5.46e-02
$10000t^3 + 0.01t^2 + 5t$	`CD`	3.56e-03	3.68e-03	2.63	6	18	4.02e-02
$10000t^3 + 0.01t^2 + 5t$	`FD_3P`	4.49e-03	4.64e-03	3.65	6	15	1.17e-02
$10000t^3 + 0.01t^2 + 5t$	`FD_4P`	6.72e+02	3.20e+03	11.72	8	22	2.37e-06
$10000t^3 + 0.01t^2 + 5t$	`CD_4P`	8.35e+02	1.03e+04	1.95	12	28	1.99e-07
$10000t^3 + 0.01t^2 + 5t$	`L2_CD`	9.59e+02	2.84e+03	1.95	12	27	7.49e-08

We can see that when the Moré-Wild heuristic does not declare a failure, it usually outputs an interval h that is quite close to our procedure and produces similar relative error as ours. However, there are many cases where Moré-Wild heuristic fails, while our procedure works very robustly in all cases.

B.2.5. Finite-Difference L-BFGS

We present the total number of function evaluations and final optimality gap $\phi(x) - \phi^*$ used by each method in Tables B.11–B.18.

Table B.9. Comparison between the Moré-Wild heuristic against our adaptive procedure on function $\phi(t) = \sin(t)$ with various ϵ_f and t. We use "$--$" to report the cases where Moré-Wild heuristic fails. Subscript "MW" labels the results corresponding to Moré-Wild heuristic, and subscript "ada" labels the results corresponding to our adaptive procedure; δ is the relative error, and $\overline{\delta}$ is the worst-case relative error.

ϵ_f	t	h_{MW}	δ_{MW}	$\overline{\delta}_{\text{MW}}$	h_{ada}	δ_{ada}	$\overline{\delta}_{\text{ada}}$
1.00e-08	1.00e-08	$--$	$--$	$--$	3.20e-03	0.000	0.000
1.00e-08	1.00e-06	$--$	$--$	$--$	3.20e-03	0.000	0.000
1.00e-08	1.00e-04	1.68e-02	0.000	0.000	3.20e-03	0.000	0.000
1.00e-08	1.00e-02	1.68e-03	0.000	0.000	2.00e-03	0.000	0.000
1.00e-08	0.00e+00	$--$	$--$	$--$	3.20e-03	0.000	0.000
1.00e-06	1.00e-08	$--$	$--$	$--$	1.40e-02	0.000	0.000
1.00e-06	1.00e-06	$--$	$--$	$--$	1.40e-02	0.000	0.000
1.00e-06	1.00e-04	$--$	$--$	$--$	1.40e-02	0.000	0.000
1.00e-06	1.00e-02	1.69e-02	0.000	0.000	1.40e-02	0.000	0.000
1.00e-06	0.00e+00	$--$	$--$	$--$	1.40e-02	0.000	0.000
1.00e-04	1.00e-08	$--$	$--$	$--$	5.00e-02	0.001	0.004
1.00e-04	1.00e-06	$--$	$--$	$--$	6.50e-02	0.003	0.004
1.00e-04	1.00e-04	$--$	$--$	$--$	5.00e-02	0.001	0.004
1.00e-04	1.00e-02	$--$	$--$	$--$	5.00e-02	0.001	0.005
1.00e-04	0.00e+00	$--$	$--$	$--$	6.50e-02	0.000	0.004
1.00e-02	1.00e-08	$--$	$--$	$--$	2.00e-01	0.058	0.107
1.00e-02	1.00e-06	5.20e-01	0.067	0.083	3.50e-01	0.018	0.077
1.00e-02	1.00e-04	$--$	$--$	$--$	3.50e-01	0.019	0.077
1.00e-02	1.00e-02	$--$	$--$	$--$	3.50e-01	0.029	0.079
1.00e-02	0.00e+00	6.10e-01	0.085	0.094	3.50e-01	0.032	0.077

In general, our adaptive procedure is more robust to different noise levels. Our method only fails when the initial choice of h is not sufficiently small to initially identify the local behavior of the function. This can be seen, for example, with the `BOX2` example. On the other hand, Moré and Wild's heuristic frequently fails when the noise level is large (for example, with $\epsilon_f = 10^{-1}$). This is due both to the case where $\phi(x) \approx 0$ and hence (3.2.11) fails, as well as the case where two iterations are insufficient to find an h that satisfies their conditions. In both cases, we denote a failure case with *. As expected, using a fixed interval is always efficient, but may perform poorly when the Hessian in the function changes, as described in Section 3.4.

Table B.10. Comparison between the Moré-Wild heuristic against our adaptive procedure on function $\phi(t) = a \cdot (\exp(b \cdot t) - 1)$ with $\epsilon_f = \texttt{1E-3}$ at $t = 0$. We use "$--$" to report the cases where Moré-Wild heuristic fails. Subscript "MW" labels the results corresponding to Moré-Wild heuristic, and subscript "ada" labels the results corresponding to our adaptive procedure; δ is the relative error, and $\overline{\delta}$ is the worst-case relative error.

a	b	h_{MW}	δ_{MW}	$\overline{\delta}_{\text{MW}}$	h_{ada}	δ_{ada}	$\overline{\delta}_{\text{ada}}$
0.01	0.01	$--$	$--$	$--$	4.05e+01	0.135	0.727
0.01	0.10	$--$	$--$	$--$	4.05e+00	0.145	0.727
0.01	1.00	$--$	$--$	$--$	4.43e-01	0.275	0.710
0.01	10.00	4.68e-02	0.177	0.702	3.95e-02	0.084	0.732
0.01	100.00	$--$	$--$	$--$	3.95e-03	0.123	0.732
0.10	0.01	$--$	$--$	$--$	1.62e+01	0.032	0.209
0.10	0.10	$--$	$--$	$--$	1.77e+00	0.074	0.207
0.10	1.00	1.64e-01	0.072	0.209	1.58e-01	0.095	0.210
0.10	10.00	1.62e-02	0.096	0.209	1.58e-02	0.055	0.210
0.10	100.00	$--$	$--$	$--$	1.73e-03	0.078	0.207
1.00	0.01	$--$	$--$	$--$	7.08e+00	0.041	0.065
1.00	0.10	$--$	$--$	$--$	6.32e-01	0.034	0.064
1.00	1.00	5.26e-02	0.029	0.065	6.32e-02	0.036	0.064
1.00	10.00	5.28e-03	0.012	0.065	6.92e-03	0.014	0.064
1.00	100.00	$--$	$--$	$--$	6.18e-04	0.009	0.064
10.00	0.01	$--$	$--$	$--$	2.53e+00	0.012	0.021
10.00	0.10	1.76e-01	0.011	0.020	2.53e-01	0.009	0.021
10.00	1.00	1.68e-02	0.006	0.020	1.58e-02	0.005	0.021
10.00	10.00	1.68e-03	0.002	0.020	2.47e-03	0.014	0.021
10.00	100.00	$--$	$--$	$--$	2.47e-04	0.010	0.021
100.00	0.01	$--$	$--$	$--$	6.32e-01	0.003	0.006
100.00	0.10	5.39e-02	0.002	0.006	6.32e-02	0.004	0.006
100.00	1.00	5.31e-03	0.001	0.006	6.92e-03	0.001	0.006
100.00	10.00	5.31e-04	0.000	0.006	6.18e-04	0.001	0.006
100.00	100.00	$--$	$--$	$--$	6.18e-05	0.001	0.006

Table B.11. Total number of function evaluations used and final accuracy achieved by forward-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Moré-Wild**		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-1}	1373	7.651	1457	4.777×10^{-1}	559	4.950×10^{-1}
ALLINITU	4	1×10^{-1}	656	2.767	972	3.708×10^{-1}	686	1.003×10^{-1}
ARWHEAD	100	1×10^{-1}	2513	2.027×10^{-1}	8784	3.040×10^{-1}	2811	2.458×10^{-1}
BARD	3	1×10^{-1}	650	8.311×10^{-1}	602	7.581×10^{-2}	670	3.942×10^{-1}
BDQRTIC	100	1×10^{-1}	2601	6.278	9213	5.902	5246	4.438
BIGGS3	3	1×10^{-1}	650	1.195	726	1.348	674	1.545
BIGGS5	5	1×10^{-1}	662	1.294	1457	1.319	696	1.359
BIGGS6	6	1×10^{-1}	668	4.026×10^{-1}	853	5.483×10^{-1}	693	6.766×10^{-1}
BOX2	2	1×10^{-1}	644	4.345×10^{-2}	709	4.095×10^{-2}	1975	3.406^*
BOX3	3	1×10^{-1}	650	4.843×10^{-2}	1385	1.887×10^{-1}	676	7.527×10^{-2}
BRKMCC	2	1×10^{-1}	644	4.391×10^{-2}	709	5.850×10^{-3}	658	1.674×10^{-1}
BROWNAL	100	1×10^{-1}	2513	5.097×10^{-2}	10800	4.318×10^{-2}	3293	1.250×10^{-2}
BROWNDEN	4	1×10^{-1}	656	1.159×10^{-1}	1364	1.568×10^{-1}	1328	1.465×10^{-1}
CLIFF	2	1×10^{-1}	644	2.902×10^{2}	861	1.180×10^{1}	1593	3.385×10^{-1}
CRAGGLVY	100	1×10^{-1}	8011	1.850×10^{2}	207136	$5.048 \times 10^{2*}$	11498	1.906×10^{1}
CUBE	2	1×10^{-1}	644	4.352×10^{-2}	6340	4.227^*	659	8.670×10^{-2}
DENSCHND	3	1×10^{-1}	650	2.254×10^{2}	2062	2.627×10^{-1}	1018	9.226×10^{-3}
DENSCHNE	3	1×10^{-1}	650	1.151	2062	1.059	542	1.018
DIXMAANH	300	1×10^{-1}	3713	1.382×10^{1}	22922	1.392×10^{1}	7809	1.280×10^{1}
DQRTIC	100	1×10^{-1}	2513	1.743×10^{2}	26299	4.930	10912	4.944×10^{-1}
EDENSCH	36	1×10^{-1}	2129	1.674	4995	1.642	4624	3.352
EIGENALS	110	1×10^{-1}	2609	5.105×10^{1}	63118	1.496×10^{1}	11630	1.236×10^{1}
EIGENBLS	110	1×10^{-1}	5103	6.717	8306	1.292×10^{1}	5098	6.533
EIGENCLS	30	1×10^{-1}	1452	1.401	10899	1.218×10^{1}	4088	1.907
ENGVAL1	100	1×10^{-1}	2295	7.316×10^{-1}	610646	$2.114 \times 10^{2*}$	4956	1.186
EXPFIT	2	1×10^{-1}	644	6.682×10^{-1}	31080	$1.270 \times 10^{2*}$	659	6.881×10^{-2}
FLETCBV3	100	1×10^{-1}	2549	1.785×10^{5}	9488	1.785×10^{5}	21950	8.454×10^{3}
FLETCHBV	100	1×10^{-1}	40856	-1.175×10^{9}	135927	1.154×10^{9}	166981	5.696×10^{9}
FREUROTH	100	1×10^{-1}	2295	7.097×10^{1}	9658	5.756×10^{1}	4008	5.912×10^{1}
GENROSE	100	1×10^{-1}	4326	1.421×10^{2}	36745	1.321×10^{2}	5347	1.364×10^{2}
GULF	3	1×10^{-1}	650	6.713	1385	6.820	670	6.664
HAIRY	2	1×10^{-1}	644	9.325×10^{1}	1328	$4.889 \times 10^{2*}$	1008	7.985×10^{-2}
HELIX	3	1×10^{-1}	650	7.601	745	7.591	671	7.656
NCB20B	100	1×10^{-1}	2295	4.793×10^{-1}	5531	2.000×10^{-1}	5393	1.280×10^{-1}
NONDIA	100	1×10^{-1}	2513	5.546×10^{-1}	10800	4.717×10^{-1}	5076	3.770×10^{-1}
NONDQUAR	100	1×10^{-1}	2613	8.483×10^{-1}	33658	7.457	5707	3.571×10^{-1}
OSBORNEA	5	1×10^{-1}	662	1.748×10^{-1}	52825	$1.142 \times 10^{2*}$	2869	1.722×10^{-1}
OSBORNEB	11	1×10^{-1}	698	1.178	1014	3.021	769	1.851
PENALTY1	100	1×10^{-1}	8895	1.008×10^{2}	20136	1.131	12532	1.268
PFIT1LS	3	1×10^{-1}	17316	1.346×10^{2}	745	4.139	675	7.796
PFIT2LS	3	1×10^{-1}	20616	2.684×10^{2}	1385	2.350×10^{1}	1991	4.006
PFIT3LS	3	1×10^{-1}	20088	9.024×10^{2}	2126	2.933	1988	2.171
PFIT4LS	3	1×10^{-1}	21702	2.771×10^{3}	3839	2.704	1989	9.426
QUARTC	100	1×10^{-1}	2513	1.743×10^{2}	26299	4.930	10912	4.944×10^{-1}
SINEVAL	2	1×10^{-1}	644	5.576	566	4.712	664	5.554
SINQUAD	100	1×10^{-1}	4326	1.254×10^{1}	28209	3.331×10^{1}	5281	9.760
SISSER	2	1×10^{-1}	644	3.393×10^{-2}	2026	1.611×10^{-1}	664	7.452×10^{-3}
SPARSQUR	100	1×10^{-1}	4326	1.365	506948	$6.928 \times 10^{1*}$	5592	1.209×10^{-1}
TOINTGSS	100	1×10^{-1}	2513	2.071×10^{1}	8784	1.467×10^{1}	4133	9.678
TQUARTIC	100	1×10^{-1}	2513	1.108	62958	3.128^*	3391	7.331×10^{-1}
TRIDIA	100	1×10^{-1}	2513	8.478×10^{1}	18300	1.691×10^{1}	5677	3.935×10^{1}
WATSON	31	1×10^{-1}	1881	7.753×10^{-1}	4702	1.993	2174	2.873
WOODS	100	1×10^{-1}	10414	8.618×10^{1}	16987	2.907×10^{2}	5255	5.533×10^{1}
ZANGWIL2	2	1×10^{-1}	644	2.686×10^{-2}	1349	3.307×10^{-2}	657	3.852×10^{-2}

Table B.12. Total number of function evaluations used and final accuracy achieved by forward-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Moré-Wild**		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-3}	1008	3.772×10^{-1}	817	3.313×10^{-1}	560	3.286×10^{-1}
ALLINITU	4	1×10^{-3}	656	2.354×10^{-3}	563	1.858×10^{-3}	686	1.172×10^{-3}
ARWHEAD	100	1×10^{-3}	2513	4.510×10^{-2}	7331	4.407×10^{-2}	4951	4.160×10^{-2}
BARD	3	1×10^{-3}	650	2.093×10^{-3}	602	3.467×10^{-3}	538	3.273×10^{-3}
BDQRTIC	100	1×10^{-3}	6017	9.588×10^{-2}	8733	1.151×10^{-1}	8551	9.804×10^{-2}
BIGGS3	3	1×10^{-3}	650	8.010×10^{-4}	712	1.375×10^{-2}	672	2.297×10^{-2}
BIGGS5	5	1×10^{-3}	662	1.813×10^{-1}	1457	1.261×10^{-1}	680	1.453×10^{-1}
BIGGS6	6	1×10^{-3}	668	2.904×10^{-1}	853	2.907×10^{-1}	563	2.909×10^{-1}
BOX2	2	1×10^{-3}	644	1.964×10^{-4}	709	2.624×10^{-6}	661	1.882×10^{-5}
BOX3	3	1×10^{-3}	650	9.556×10^{-4}	726	4.567×10^{-4}	672	3.568×10^{-4}
BRKMCC	2	1×10^{-3}	644	3.552×10^{-3}	566	3.872×10^{-3}	661	8.029×10^{-4}
BROWNAL	100	1×10^{-3}	2513	6.869×10^{-4}	10800	4.648×10^{-4}	3293	1.670×10^{-4}
BROWNDEN	4	1×10^{-3}	656	1.183×10^{-3}	706	1.912×10^{-3}	1327	8.187×10^{-4}
CLIFF	2	1×10^{-3}	644	2.902×10^{2}	28980	8.220×10^{-1}	1008	3.059×10^{-4}
CRAGGLVY	100	1×10^{-3}	7993	4.852	8619	1.046	7272	9.025×10^{-1}
CUBE	2	1×10^{-3}	644	4.201×10^{-2}	709	4.951×10^{-2}	659	4.381×10^{-2}
DENSCHND	3	1×10^{-3}	631	4.109	1385	1.170×10^{-3}	1018	9.852×10^{-4}
DENSCHNE	3	1×10^{-3}	650	9.994×10^{-1}	745	1.000	672	1.002
DIXMAANH	300	1×10^{-3}	7774	1.041	37580	7.456×10^{-2}	15794	9.375×10^{-2}
DQRTIC	100	1×10^{-3}	6609	4.832	21149	9.935×10^{-3}	11600	1.713×10^{-2}
EDENSCH	36	1×10^{-3}	1194	5.382×10^{-2}	6085	2.188×10^{-2}	3234	3.748×10^{-2}
EIGENALS	110	1×10^{-3}	4850	3.394×10^{-1}	22676	8.848×10^{-2}	8753	7.456×10^{-2}
EIGENBLS	110	1×10^{-3}	1932	2.084	21095	1.631	6825	1.740
EIGENCLS	30	1×10^{-3}	1875	7.554×10^{-2}	8449	7.032×10^{-2}	3187	8.686×10^{-2}
ENGVAL1	100	1×10^{-3}	2295	8.024×10^{-2}	9658	2.733×10^{-2}	4841	4.366×10^{-2}
EXPFIT	2	1×10^{-3}	644	6.733×10^{-3}	566	3.801×10^{-4}	663	3.592×10^{-4}
FLETCBV3	100	1×10^{-3}	2549	1.785×10^{5}	9488	1.785×10^{5}	29889	-1.181×10^{2}
FLETCHBV	100	1×10^{-3}	33825	-7.812×10^{8}	141125	6.878×10^{7}	62282	-6.875×10^{8}
FREUROTH	100	1×10^{-3}	4861	2.366×10^{-1}	9725	2.041×10^{-2}	8226	4.143×10^{-2}
GENROSE	100	1×10^{-3}	5779	1.109×10^{2}	17379	1.108×10^{2}	7561	1.110×10^{2}
GULF	3	1×10^{-3}	650	6.626	599	6.622	672	6.622
HAIRY	2	1×10^{-3}	644	7.979×10^{-3}	566	7.030×10^{-4}	853	3.304×10^{-3}
HELIX	3	1×10^{-3}	650	4.079×10^{-3}	1091	2.495×10^{-4}	671	3.746×10^{-4}
NCB20B	100	1×10^{-3}	2295	1.400×10^{-2}	8984	5.856×10^{-3}	6650	4.462×10^{-3}
NONDIA	100	1×10^{-3}	2513	4.911×10^{-1}	13295	4.862×10^{-1}	8550	4.612×10^{-1}
NONDQUAR	100	1×10^{-3}	5534	9.503×10^{-2}	29253	1.247×10^{-2}	8653	3.602×10^{-2}
OSBORNEA	5	1×10^{-3}	662	1.525×10^{-1}	54694	2.093	553	1.567×10^{-1}
OSBORNEB	11	1×10^{-3}	555	3.697×10^{-1}	2314	3.170×10^{-1}	766	3.115×10^{-1}
PENALTY1	100	1×10^{-3}	4326	2.200×10^{1}	14928	1.345×10^{-3}	12554	2.741×10^{-4}
PFIT1LS	3	1×10^{-3}	17244	7.787	2062	6.076×10^{-2}	1312	4.829×10^{-2}
PFIT2LS	3	1×10^{-3}	19169	1.147×10^{2}	2062	5.424×10^{-2}	1310	8.222×10^{-2}
PFIT3LS	3	1×10^{-3}	21645	6.633×10^{2}	2062	2.615×10^{-2}	1993	4.635×10^{-2}
PFIT4LS	3	1×10^{-3}	22928	2.305×10^{3}	2631	2.308×10^{-1}	1992	2.053×10^{-1}
QUARTC	100	1×10^{-3}	6609	4.832	21149	9.935×10^{-3}	11600	1.713×10^{-2}
SINEVAL	2	1×10^{-3}	644	1.557	2026	2.780×10^{-1}	1299	1.695×10^{-1}
SINQUAD	100	1×10^{-3}	4326	3.852×10^{-2}	19928	1.067×10^{-1}	5543	7.637×10^{-2}
SISSER	2	1×10^{-3}	644	2.069×10^{-3}	566	1.051×10^{-3}	664	2.094×10^{-4}
SPARSQUR	100	1×10^{-3}	6609	3.112×10^{-2}	14021	8.064×10^{-2}	6625	3.022×10^{-3}
TOINTGSS	100	1×10^{-3}	2513	2.198×10^{-1}	5071	1.026×10^{-1}	3730	6.133×10^{-2}
TQUARTIC	100	1×10^{-3}	2513	7.227×10^{-1}	7529	7.241×10^{-1}	4090	7.220×10^{-1}
TRIDIA	100	1×10^{-3}	7921	3.607×10^{-1}	28195	3.539×10^{-1}	9887	4.283×10^{-1}
WATSON	31	1×10^{-3}	1881	1.862×10^{-1}	10162	2.201×10^{-2}	2919	1.754×10^{-1}
WOODS	100	1×10^{-3}	11923	7.268×10^{-1}	10536	6.643	5493	6.472
ZANGWIL2	2	1×10^{-3}	644	8.620×10^{-4}	566	8.797×10^{-4}	657	6.881×10^{-4}

Table B.13. Total number of function evaluations used and final accuracy achieved by forward-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Moré-Wild**		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-5}	662	3.023×10^{-1}	1880	2.674×10^{-4}	2014	4.084×10^{-4}
ALLINITU	4	1×10^{-5}	656	1.835×10^{-5}	706	1.146×10^{-5}	682	6.468×10^{-6}
ARWHEAD	100	1×10^{-5}	3832	1.827×10^{-4}	11900	3.262×10^{-4}	7570	5.117×10^{-4}
BARD	3	1×10^{-5}	650	1.894×10^{-3}	602	1.878×10^{-3}	674	1.903×10^{-3}
BDQRTIC	100	1×10^{-5}	10225	3.227×10^{-3}	14968	2.305×10^{-4}	9860	5.248×10^{-4}
BIGGS3	3	1×10^{-5}	631	2.826×10^{-4}	712	2.404×10^{-4}	670	1.581×10^{-4}
BIGGS5	5	1×10^{-5}	1008	2.170×10^{-2}	3911	1.064×10^{-4}	2030	9.355×10^{-3}
BIGGS6	6	1×10^{-5}	1014	1.863×10^{-3}	2170	9.419×10^{-5}	1348	4.434×10^{-4}
BOX2	2	1×10^{-5}	644	3.493×10^{-6}	1349	8.092×10^{-6}	658	2.480×10^{-6}
BOX3	3	1×10^{-5}	650	7.493×10^{-6}	726	4.612×10^{-6}	672	1.114×10^{-5}
BRKMCC	2	1×10^{-5}	644	3.927×10^{-7}	566	1.244×10^{-5}	661	3.079×10^{-5}
BROWNAL	100	1×10^{-5}	2513	1.963×10^{-5}	9567	1.629×10^{-5}	4851	1.487×10^{-5}
BROWNDEN	4	1×10^{-5}	656	3.971×10^{-6}	1145	1.268×10^{-5}	1324	1.062×10^{-5}
CLIFF	2	1×10^{-5}	644	2.902×10^{2}	1349	2.275×10^{-4}	1008	2.238×10^{-4}
CRAGGLVY	100	1×10^{-5}	7993	8.012×10^{-2}	13711	8.641×10^{-3}	10935	1.884×10^{-2}
CUBE	2	1×10^{-5}	644	4.069×10^{-2}	709	1.148×10^{-2}	662	4.524×10^{-3}
DENSCHND	3	1×10^{-5}	996	1.720×10^{-1}	936	1.141×10^{-5}	1016	2.311×10^{-5}
DENSCHNE	3	1×10^{-5}	650	9.993×10^{-1}	563	9.993×10^{-1}	672	9.993×10^{-1}
DIXMAANH	300	1×10^{-5}	19344	1.035×10^{-2}	38056	5.615×10^{-3}	15708	8.080×10^{-3}
DQRTIC	100	1×10^{-5}	3486	2.687×10^{-1}	16434	1.667×10^{-4}	14109	1.864×10^{-4}
EDENSCH	36	1×10^{-5}	1194	4.513×10^{-4}	4840	5.362×10^{-4}	4624	7.831×10^{-4}
EIGENALS	110	1×10^{-5}	4921	1.929×10^{-2}	18980	1.337×10^{-2}	10030	1.353×10^{-2}
EIGENBLS	110	1×10^{-5}	3178	1.556	11368	1.553	46254	1.018×10^{-2}
EIGENCLS	30	1×10^{-5}	4370	5.560×10^{-4}	9380	8.302×10^{-4}	6373	5.415×10^{-4}
ENGVAL1	100	1×10^{-5}	2601	2.285×10^{-4}	8619	2.302×10^{-4}	7134	2.310×10^{-4}
EXPFIT	2	1×10^{-5}	644	4.595×10^{-5}	563	6.982×10^{-6}	666	6.814×10^{-6}
FLETCBV3	100	1×10^{-5}	4326	1.785×10^{5}	44623	1.785×10^{5}	100756	-1.304×10^{2}
FLETCHBV	100	1×10^{-5}	113423	3.562×10^{9}	315886	5.660×10^{7}	213807	1.765×10^{9}
FREUROTH	100	1×10^{-5}	4326	2.559×10^{-3}	10097	5.248×10^{-4}	7166	6.820×10^{-4}
GENROSE	100	1×10^{-5}	27714	8.446×10^{-3}	86415	5.674×10^{-3}	56271	1.067×10^{-2}
GULF	3	1×10^{-5}	650	6.596×10^{-3}	1091	2.275×10^{-3}	675	3.771×10^{-3}
HAIRY	2	1×10^{-5}	644	1.129×10^{-4}	1328	4.688×10^{-6}	1008	8.137×10^{-6}
HELIX	3	1×10^{-5}	650	7.422×10^{-4}	1385	8.376×10^{-5}	650	4.881×10^{-5}
NCB20B	100	1×10^{-5}	4790	4.231×10^{-4}	13270	3.565×10^{-4}	9691	3.061×10^{-4}
NONDIA	100	1×10^{-5}	4326	2.174×10^{-4}	21377	1.572×10^{-2}	8423	1.396×10^{-2}
NONDQUAR	100	1×10^{-5}	11923	9.225×10^{-3}	33549	1.510×10^{-3}	14104	1.836×10^{-3}
OSBORNEA	5	1×10^{-5}	853	7.139×10^{-4}	62478	5.603×10^{-1}	1334	1.215×10^{-3}
OSBORNEB	11	1×10^{-5}	1137	1.051×10^{-1}	4506	2.381×10^{-3}	2936	3.325×10^{-3}
PENALTY1	100	1×10^{-5}	4326	1.113×10^{-1}	20475	1.893×10^{-4}	12804	1.877×10^{-4}
PFIT1LS	3	1×10^{-5}	17338	1.609×10^{-2}	2062	2.921×10^{-6}	1312	2.320×10^{-5}
PFIT2LS	3	1×10^{-5}	18947	3.652×10^{-2}	2062	3.200×10^{-3}	1310	2.463×10^{-3}
PFIT3LS	3	1×10^{-5}	21038	2.034×10^{-1}	2062	2.364×10^{-2}	1999	2.822×10^{-2}
PFIT4LS	3	1×10^{-5}	22064	2.451×10^{-1}	2631	1.004×10^{-1}	1990	1.177×10^{-1}
QUARTC	100	1×10^{-5}	3486	2.687×10^{-1}	16434	1.667×10^{-4}	14109	1.864×10^{-4}
SINEVAL	2	1×10^{-5}	625	4.640×10^{-3}	2026	3.676×10^{-3}	1977	1.667×10^{-4}
SINQUAD	100	1×10^{-5}	2613	1.417×10^{-3}	11616	1.581×10^{-4}	5545	2.677×10^{-4}
SISSER	2	1×10^{-5}	644	7.894×10^{-6}	709	2.995×10^{-7}	658	3.218×10^{-6}
SPARSQUR	100	1×10^{-5}	8365	9.873×10^{-4}	14478	7.713×10^{-5}	8320	3.739×10^{-5}
TOINTGSS	100	1×10^{-5}	2513	2.193×10^{-3}	10274	6.054×10^{-4}	3729	1.167×10^{-3}
TQUARTIC	100	1×10^{-5}	4571	2.622×10^{-1}	9184	1.534×10^{-1}	10072	1.199×10^{-1}
TRIDIA	100	1×10^{-5}	9428	1.900×10^{-3}	45233	3.442×10^{-3}	19815	2.348×10^{-3}
WATSON	31	1×10^{-5}	3072	3.088×10^{-3}	14287	8.549×10^{-3}	6393	6.386×10^{-3}
WOODS	100	1×10^{-5}	7964	8.185×10^{-3}	25179	5.822×10^{-3}	13779	3.840×10^{-3}
ZANGWIL2	2	1×10^{-5}	644	8.737×10^{-6}	670	1.685×10^{-5}	657	7.597×10^{-6}

Table B.14. Total number of function evaluations used and final accuracy achieved by forward-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Moré-Wild**		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-7}	662	9.768×10^{-5}	2985	1.199×10^{-5}	1334	1.001×10^{-5}
ALLINITU	4	1×10^{-7}	656	2.172×10^{-7}	706	1.061×10^{-7}	543	7.839×10^{-8}
ARWHEAD	100	1×10^{-7}	4326	1.643×10^{-6}	11900	5.117×10^{-7}	5842	1.049×10^{-6}
BARD	3	1×10^{-7}	650	2.376×10^{-5}	706	7.175×10^{-7}	660	1.273×10^{-7}
BDQRTIC	100	1×10^{-7}	8011	1.802×10^{-5}	14968	5.277×10^{-6}	12535	4.346×10^{-6}
BIGGS3	3	1×10^{-7}	650	2.348×10^{-7}	726	3.990×10^{-7}	1018	1.386×10^{-6}
BIGGS5	5	1×10^{-7}	1008	6.378×10^{-6}	1614	1.637×10^{-6}	2869	3.891×10^{-5}
BIGGS6	6	1×10^{-7}	1014	-5.616×10^{-3}	2724	-5.595×10^{-3}	2023	-5.647×10^{-3}
BOX2	2	1×10^{-7}	644	1.726×10^{-6}	690	1.547×10^{-7}	658	1.865×10^{-6}
BOX3	3	1×10^{-7}	650	7.911×10^{-7}	602	6.639×10^{-7}	862	5.213×10^{-7}
BRKMCC	2	1×10^{-7}	644	2.465×10^{-7}	670	4.240×10^{-7}	661	9.632×10^{-7}
BROWNAL	100	1×10^{-7}	4326	2.571×10^{-7}	9567	1.334×10^{-5}	7259	5.981×10^{-7}
BROWNDEN	4	1×10^{-7}	656	3.507×10^{-7}	1145	1.214×10^{-7}	1324	1.468×10^{-7}
CLIFF	2	1×10^{-7}	644	2.902×10^{2}	861	2.198×10^{-4}	1300	2.192×10^{-4}
CRAGGLVY	100	1×10^{-7}	4678	1.570×10^{-2}	18164	1.153×10^{-4}	14124	7.253×10^{-5}
CUBE	2	1×10^{-7}	990	1.798×10^{-4}	1349	1.188×10^{-4}	662	1.404×10^{-4}
DENSCHND	3	1×10^{-7}	650	3.918×10^{-2}	1385	3.805×10^{-8}	1311	1.209×10^{-7}
DENSCHNE	3	1×10^{-7}	650	9.993×10^{-1}	1091	1.427×10^{-7}	1015	8.898×10^{-8}
DIXMAANH	300	1×10^{-7}	62344	1.773×10^{-4}	63812	2.194×10^{-5}	37104	2.753×10^{-5}
DQRTIC	100	1×10^{-7}	4571	5.119×10^{-3}	20136	3.509×10^{-7}	17121	4.377×10^{-6}
EDENSCH	36	1×10^{-7}	2165	2.262×10^{-6}	3168	3.776×10^{-6}	4624	3.756×10^{-6}
EIGENALS	110	1×10^{-7}	23459	1.291×10^{-3}	132863	2.204×10^{-4}	43487	2.893×10^{-4}
EIGENBLS	110	1×10^{-7}	38955	1.169×10^{-3}	182840	9.604×10^{-4}	92870	9.672×10^{-4}
EIGENCLS	30	1×10^{-7}	3906	1.173×10^{-5}	12408	9.139×10^{-6}	6345	8.639×10^{-6}
ENGVAL1	100	1×10^{-7}	4326	1.734×10^{-6}	8997	3.737×10^{-6}	6527	2.550×10^{-6}
EXPFIT	2	1×10^{-7}	644	3.707×10^{-7}	709	5.737×10^{-8}	666	4.705×10^{-8}
FLETCBV3	100	1×10^{-7}	39622	4.451×10^{2}	281142	1.666×10^{2}	208557	-5.161×10^{1}
FLETCHBV	100	1×10^{-7}	116172	-1.393×10^{9}	311603	2.365×10^{9}	212997	5.073×10^{9}
FREUROTH	100	1×10^{-7}	4678	2.155×10^{-5}	10097	8.099×10^{-6}	7165	6.707×10^{-6}
GENROSE	100	1×10^{-7}	27731	1.016×10^{-4}	83521	6.277×10^{-5}	56975	8.924×10^{-5}
GULF	3	1×10^{-7}	650	4.478×10^{-3}	1163	3.909×10^{-3}	1992	1.893×10^{-5}
HAIRY	2	1×10^{-7}	644	1.478×10^{-6}	670	1.107×10^{-7}	1008	2.080×10^{-7}
HELIX	3	1×10^{-7}	650	7.990×10^{-6}	1091	1.355×10^{-5}	674	1.967×10^{-6}
NCB20B	100	1×10^{-7}	12717	2.174×10^{-5}	34691	2.432×10^{-5}	21290	2.303×10^{-5}
NONDIA	100	1×10^{-7}	5779	2.912×10^{-6}	24808	1.134×10^{-5}	5826	8.625×10^{-4}
NONDQUAR	100	1×10^{-7}	8438	1.256×10^{-3}	102555	1.348×10^{-4}	44583	1.199×10^{-4}
OSBORNEA	5	1×10^{-7}	662	8.747×10^{-5}	47678	1.721×10^{-1}	2014	2.290×10^{-5}
OSBORNEB	11	1×10^{-7}	2015	1.342×10^{-5}	4408	1.260×10^{-5}	3626	2.063×10^{-5}
PENALTY1	100	1×10^{-7}	4326	6.722×10^{-4}	15829	1.869×10^{-4}	13984	1.870×10^{-4}
PFIT1LS	3	1×10^{-7}	15490	5.585×10^{-4}	2062	1.259×10^{-6}	1021	7.981×10^{-6}
PFIT2LS	3	1×10^{-7}	18475	3.194×10^{-2}	2126	1.834×10^{-3}	1314	1.441×10^{-3}
PFIT3LS	3	1×10^{-7}	19911	4.062×10^{-2}	2062	2.782×10^{-2}	1989	3.161×10^{-2}
PFIT4LS	3	1×10^{-7}	22339	2.817×10^{-1}	2631	1.171×10^{-1}	2842	1.668×10^{-1}
QUARTC	100	1×10^{-7}	4571	5.119×10^{-3}	20136	3.509×10^{-7}	17121	4.377×10^{-6}
SINEVAL	2	1×10^{-7}	625	7.770×10^{-4}	2026	9.623×10^{-7}	1300	5.182×10^{-4}
SINQUAD	100	1×10^{-7}	3400	1.629×10^{-5}	11616	1.584×10^{-6}	5543	6.333×10^{-6}
SISSER	2	1×10^{-7}	644	2.687×10^{-7}	709	1.365×10^{-9}	661	6.235×10^{-8}
SPARSQUR	100	1×10^{-7}	4861	2.668×10^{-5}	26138	1.350×10^{-7}	12562	2.022×10^{-7}
TOINTGSS	100	1×10^{-7}	2513	2.192×10^{-5}	10651	6.014×10^{-6}	3729	1.168×10^{-5}
TQUARTIC	100	1×10^{-7}	8895	1.645×10^{-2}	14021	8.050×10^{-4}	10062	1.413×10^{-3}
TRIDIA	100	1×10^{-7}	10619	2.499×10^{-5}	47586	2.839×10^{-5}	21947	3.486×10^{-5}
WATSON	31	1×10^{-7}	4264	1.161×10^{-3}	13658	1.172×10^{-3}	4585	1.854×10^{-3}
WOODS	100	1×10^{-7}	6609	5.611×10^{-5}	25179	4.354×10^{-5}	10111	9.748×10^{-5}
ZANGWIL2	2	1×10^{-7}	644	8.738×10^{-8}	670	1.683×10^{-7}	657	7.832×10^{-8}

Table B.15. Total number of function evaluations used and final accuracy achieved by central-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-1}	602	4.455×10^{-1}	3074	4.451×10^{-1}
ALLINITU	4	1×10^{-1}	848	7.698×10^{-1}	1371	3.378×10^{-3}
ARWHEAD	100	1×10^{-1}	4001	4.137×10^{-2}	5021	5.157×10^{-2}
BARD	3	1×10^{-1}	669	2.616×10^{-3}	947	1.535×10^{-2}
BDQRTIC	100	1×10^{-1}	8594	2.047×10^{-1}	11555	1.918×10^{-1}
BIGGS3	3	1×10^{-1}	669	2.607×10^{-1}	1150	1.154×10^{-2}
BIGGS5	5	1×10^{-1}	693	8.829×10^{-2}	1125	3.740×10^{-2}
BIGGS6	6	1×10^{-1}	705	2.966×10^{-1}	1419	2.907×10^{-1}
BOX2	2	1×10^{-1}	657	1.013×10^{-2}	560	6.435×10^{-3}
BOX3	3	1×10^{-1}	669	4.700×10^{-2}	594	1.444×10^{-3}
BRKMCC	2	1×10^{-1}	657	4.767×10^{-4}	2418	1.983×10^{-4}
BROWNAL	100	1×10^{-1}	7175	1.321×10^{-5}	12140	1.331×10^{-7}
BROWNDEN	4	1×10^{-1}	681	4.778×10^{-4}	2247	1.814×10^{-3}
CLIFF	2	1×10^{-1}	657	2.902×10^{2}	4819	2.413×10^{-2}
CRAGGLVY	100	1×10^{-1}	11144	2.765	23987	6.089×10^{-1}
CUBE	2	1×10^{-1}	657	4.421×10^{-2}	3101	4.462×10^{-2}
DENSCHND	3	1×10^{-1}	669	5.418×10^{-3}	2310	8.228×10^{-3}
DENSCHNE	3	1×10^{-1}	669	1.015	576	1.025
DIXMAANH	300	1×10^{-1}	32019	7.301×10^{-1}	41575	4.225×10^{-2}
DQRTIC	100	1×10^{-1}	10891	4.910×10^{-1}	28135	7.032×10^{-4}
EDENSCH	36	1×10^{-1}	4217	5.638×10^{-2}	6835	7.687×10^{-2}
EIGENALS	110	1×10^{-1}	7295	2.049	20145	1.099×10^{-1}
EIGENBLS	110	1×10^{-1}	12096	2.326	13672	1.801
EIGENCLS	30	1×10^{-1}	3161	6.304×10^{-1}	8124	3.930×10^{-1}
ENGVAL1	100	1×10^{-1}	6704	1.473×10^{-1}	11258	2.414×10^{-1}
EXPFIT	2	1×10^{-1}	7418	2.456×10^{1}	5915	2.093×10^{-2}
FLETCBV3	100	1×10^{-1}	2546	1.785×10^{5}	136921	-1.561×10^{2}
FLETCHBV	100	1×10^{-1}	138282	6.330×10^{9}	380580	-1.079×10^{9}
FREUROTH	100	1×10^{-1}	6216	2.650×10^{-1}	14495	4.930×10^{-2}
GENROSE	100	1×10^{-1}	7084	1.116×10^{2}	28786	1.117×10^{2}
GULF	3	1×10^{-1}	669	6.621	2124	6.728
HAIRY	2	1×10^{-1}	657	1.261×10^{-4}	1483	6.896×10^{-4}
HELIX	3	1×10^{-1}	505	8.544×10^{2}	4900	2.476×10^{1}
NCB20B	100	1×10^{-1}	3028	2.230	12309	4.906×10^{-2}
NONDIA	100	1×10^{-1}	4782	4.941×10^{-1}	9226	4.929×10^{-1}
NONDQUAR	100	1×10^{-1}	7210	1.974×10^{-1}	14490	7.337×10^{-2}
OSBORNEA	5	1×10^{-1}	693	1.530×10^{-1}	2345	8.790×10^{-1}
OSBORNEB	11	1×10^{-1}	746	6.496×10^{-1}	3119	1.366
PENALTY1	100	1×10^{-1}	7210	3.404×10^{-2}	21785	1.847×10^{-4}
PFIT1LS	3	1×10^{-1}	17843	1.276×10^{1}	3706	8.023×10^{-1}
PFIT2LS	3	1×10^{-1}	3466	1.453×10^{2}	2827	5.360×10^{-1}
PFIT3LS	3	1×10^{-1}	26780	8.283×10^{2}	3807	1.767×10^{-1}
PFIT4LS	3	1×10^{-1}	27602	2.416×10^{3}	3727	3.979×10^{-1}
QUARTC	100	1×10^{-1}	10891	4.910×10^{-1}	28135	7.032×10^{-4}
SINEVAL	2	1×10^{-1}	501	5.547	3266	7.918
SINQUAD	100	1×10^{-1}	5279	2.818×10^{-1}	11289	3.892
SISSER	2	1×10^{-1}	657	9.267×10^{-3}	491	2.900×10^{-5}
SPARSQUR	100	1×10^{-1}	5345	6.181×10^{-2}	18495	1.016×10^{-2}
TOINTGSS	100	1×10^{-1}	5391	1.021×10^{-2}	8324	4.000×10^{-9}
TQUARTIC	100	1×10^{-1}	2605	8.336×10^{-1}	12297	7.326×10^{-1}
TRIDIA	100	1×10^{-1}	10830	1.695×10^{-1}	88893	5.839×10^{-13}
WATSON	31	1×10^{-1}	2880	2.177×10^{-1}	5954	1.693×10^{-1}
WOODS	100	1×10^{-1}	5279	6.439	21136	5.545
ZANGWIL2	2	1×10^{-1}	657	4.902×10^{-4}	1351	-9.999×10^{-11}

Table B.16. Total number of function evaluations used and final accuracy achieved by central-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-3}	594	4.451×10^{-1}	2306	4.451×10^{-1}
ALLINITU	4	1×10^{-3}	574	3.325×10^{-3}	1377	1.847×10^{-5}
ARWHEAD	100	1×10^{-3}	7175	5.590×10^{-4}	8379	6.125×10^{-4}
BARD	3	1×10^{-3}	669	1.905×10^{-3}	713	1.883×10^{-3}
BDQRTIC	100	1×10^{-3}	11220	3.667×10^{-3}	31078	4.314×10^{-4}
BIGGS3	3	1×10^{-3}	669	1.009×10^{-3}	1363	1.314×10^{-4}
BIGGS5	5	1×10^{-3}	1333	1.649×10^{-2}	1921	1.418×10^{-2}
BIGGS6	6	1×10^{-3}	705	5.181×10^{-2}	1844	-3.749×10^{-3}
BOX2	2	1×10^{-3}	657	1.287×10^{-5}	560	1.740×10^{-4}
BOX3	3	1×10^{-3}	669	3.555×10^{-5}	1351	5.017×10^{-5}
BRKMCC	2	1×10^{-3}	657	2.500×10^{-6}	556	3.658×10^{-7}
BROWNAL	100	1×10^{-3}	4782	8.022×10^{-6}	12030	2.446×10^{-8}
BROWNDEN	4	1×10^{-3}	681	2.721×10^{-6}	1525	1.803×10^{-7}
CLIFF	2	1×10^{-3}	657	2.902×10^{2}	3590	2.497×10^{-4}
CRAGGLVY	100	1×10^{-3}	11976	1.745×10^{-2}	30115	2.086×10^{-3}
CUBE	2	1×10^{-3}	657	4.239×10^{-2}	2418	4.580×10^{-2}
DENSCHND	3	1×10^{-3}	669	4.714×10^{-3}	1969	6.472×10^{-4}
DENSCHNE	3	1×10^{-3}	669	9.993×10^{-1}	1002	9.994×10^{-1}
DIXMAANH	300	1×10^{-3}	28496	1.428×10^{-2}	48610	3.693×10^{-3}
DQRTIC	100	1×10^{-3}	11015	1.313×10^{-2}	25201	2.425×10^{-5}
EDENSCH	36	1×10^{-3}	2128	2.700×10^{-4}	4599	2.101×10^{-4}
EIGENALS	110	1×10^{-3}	8516	4.539×10^{-2}	27468	1.303×10^{-2}
EIGENBLS	110	1×10^{-3}	47792	3.108×10^{-2}	25441	1.551
EIGENCLS	30	1×10^{-3}	4087	2.953×10^{-3}	10419	1.545×10^{-3}
ENGVAL1	100	1×10^{-3}	7210	1.658×10^{-4}	14954	4.131×10^{-4}
EXPFIT	2	1×10^{-3}	9839	3.021	1013	2.602×10^{-4}
FLETCBV3	100	1×10^{-3}	2546	1.785×10^{5}	198233	-1.561×10^{2}
FLETCHBV	100	1×10^{-3}	107444	-1.473×10^{9}	239215	3.847×10^{9}
FREUROTH	100	1×10^{-3}	7175	4.683×10^{-4}	14972	1.047×10^{-4}
GENROSE	100	1×10^{-3}	54819	2.202×10^{-3}	124290	9.891×10^{-4}
GULF	3	1×10^{-3}	1309	2.995×10^{-3}	1782	4.249×10^{-3}
HAIRY	2	1×10^{-3}	418	5.279×10^{-6}	4113	1.608×10^{-7}
HELIX	3	1×10^{-3}	432	8.522×10^{2}	2867	2.475×10^{1}
NCB20B	100	1×10^{-3}	5918	1.353×10^{-1}	14781	1.608×10^{-3}
NONDIA	100	1×10^{-3}	6475	1.027×10^{-2}	10213	1.263×10^{-2}
NONDQUAR	100	1×10^{-3}	7084	1.973×10^{-2}	28713	3.712×10^{-3}
OSBORNEA	5	1×10^{-3}	1333	2.881×10^{-3}	2237	8.790×10^{-1}
OSBORNEB	11	1×10^{-3}	2082	2.869×10^{-1}	3565	7.271×10^{-2}
PENALTY1	100	1×10^{-3}	7210	1.928×10^{-4}	20312	1.917×10^{-4}
PFIT1LS	3	1×10^{-3}	24764	1.500×10^{1}	2027	1.183×10^{-3}
PFIT2LS	3	1×10^{-3}	28415	1.973×10^{2}	2030	4.006×10^{-3}
PFIT3LS	3	1×10^{-3}	31502	1.018×10^{3}	3796	3.066×10^{-2}
PFIT4LS	3	1×10^{-3}	31607	3.352×10^{3}	3891	9.033×10^{-2}
QUARTC	100	1×10^{-3}	11015	1.313×10^{-2}	25201	2.425×10^{-5}
SINEVAL	2	1×10^{-3}	588	5.327	3436	4.873×10^{-5}
SINQUAD	100	1×10^{-3}	5391	6.603×10^{-4}	14970	1.277×10^{-4}
SISSER	2	1×10^{-3}	657	9.026×10^{-6}	684	5.497×10^{-5}
SPARSQUR	100	1×10^{-3}	5542	2.341×10^{-3}	19979	2.489×10^{-5}
TOINTGSS	100	1×10^{-3}	2896	1.267×10^{-5}	9505	4.000×10^{-9}
TQUARTIC	100	1×10^{-3}	19220	2.492×10^{-1}	8491	7.221×10^{-1}
TRIDIA	100	1×10^{-3}	17514	3.205×10^{-5}	92231	4.283×10^{-15}
WATSON	31	1×10^{-3}	3173	2.462×10^{-2}	9834	2.186×10^{-3}
WOODS	100	1×10^{-3}	11220	3.320×10^{-3}	23074	6.287×10^{-4}
ZANGWIL2	2	1×10^{-3}	657	9.871×10^{-7}	598	-9.999×10^{-11}

Table B.17. Total number of function evaluations used and final accuracy achieved by central-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-5}	612	4.451×10^{-1}	2558	3.293×10^{-1}
ALLINITU	4	1×10^{-5}	681	7.277×10^{-6}	2035	7.767×10^{-9}
ARWHEAD	100	1×10^{-5}	7487	4.138×10^{-6}	7477	1.108×10^{-6}
BARD	3	1×10^{-5}	669	1.867×10^{-3}	1500	1.678×10^{-7}
BDQRTIC	100	1×10^{-5}	11220	1.539×10^{-6}	19106	4.834×10^{-6}
BIGGS3	3	1×10^{-5}	669	4.019×10^{-6}	1088	3.251×10^{-7}
BIGGS5	5	1×10^{-5}	1756	4.920×10^{-5}	3617	1.465×10^{-5}
BIGGS6	6	1×10^{-5}	2358	-5.228×10^{-3}	4142	-5.625×10^{-3}
BOX2	2	1×10^{-5}	657	2.137×10^{-6}	1327	1.773×10^{-6}
BOX3	3	1×10^{-5}	669	6.746×10^{-7}	1349	7.643×10^{-7}
BRKMCC	2	1×10^{-5}	437	5.850×10^{-10}	558	1.300×10^{-9}
BROWNAL	100	1×10^{-5}	3508	1.384×10^{-7}	19759	1.759×10^{-16}
BROWNDEN	4	1×10^{-5}	681	7.291×10^{-9}	2060	-1.455×10^{-11}
CLIFF	2	1×10^{-5}	657	2.902×10^{2}	4442	2.272×10^{-4}
CRAGGLVY	100	1×10^{-5}	17145	3.635×10^{-5}	32645	5.568×10^{-6}
CUBE	2	1×10^{-5}	657	2.696×10^{-7}	2510	3.013×10^{-6}
DENSCHND	3	1×10^{-5}	1015	3.165×10^{-4}	1878	2.382×10^{-6}
DENSCHNE	3	1×10^{-5}	669	9.993×10^{-1}	1073	7.559×10^{-8}
DIXMAANH	300	1×10^{-5}	45747	8.540×10^{-5}	235332	1.565×10^{-10}
DQRTIC	100	1×10^{-5}	13738	1.501×10^{-4}	28793	1.977×10^{-6}
EDENSCH	36	1×10^{-5}	2382	3.038×10^{-7}	6843	3.233×10^{-7}
EIGENALS	110	1×10^{-5}	27694	1.556×10^{-3}	93131	2.888×10^{-4}
EIGENBLS	110	1×10^{-5}	86034	9.991×10^{-4}	167257	1.114×10^{-3}
EIGENCLS	30	1×10^{-5}	5297	3.243×10^{-5}	15418	5.660×10^{-6}
ENGVAL1	100	1×10^{-5}	5244	1.653×10^{-6}	12432	7.819×10^{-7}
EXPFIT	2	1×10^{-5}	2953	1.015×10^{-2}	1097	3.382×10^{-7}
FLETCBV3	100	1×10^{-5}	61879	-7.404×10^{1}	405665	-1.561×10^{2}
FLETCHBV	100	1×10^{-5}	69727	1.786×10^{9}	168055	-3.029×10^{8}
FREUROTH	100	1×10^{-5}	6618	1.671×10^{-6}	17360	1.873×10^{-7}
GENROSE	100	1×10^{-5}	53552	5.526×10^{-6}	119843	3.104×10^{-6}
GULF	3	1×10^{-5}	669	4.155×10^{-3}	2094	1.869×10^{-6}
HAIRY	2	1×10^{-5}	657	3.882×10^{-9}	2015	2.389×10^{-11}
HELIX	3	1×10^{-5}	445	8.521×10^{2}	5026	2.475×10^{1}
NCB20B	100	1×10^{-5}	14030	1.578×10^{-3}	37332	7.086×10^{-5}
NONDIA	100	1×10^{-5}	7210	1.054×10^{-5}	13740	1.839×10^{-5}
NONDQUAR	100	1×10^{-5}	18545	9.490×10^{-4}	55137	2.686×10^{-4}
OSBORNEA	5	1×10^{-5}	693	9.012×10^{-4}	2295	8.790×10^{-1}
OSBORNEB	11	1×10^{-5}	3861	4.164×10^{-4}	5596	2.698×10^{-5}
PENALTY1	100	1×10^{-5}	9496	1.868×10^{-4}	22443	1.875×10^{-4}
PFIT1LS	3	1×10^{-5}	26200	5.375	2883	8.388×10^{-6}
PFIT2LS	3	1×10^{-5}	29030	1.010×10^{2}	2008	2.499×10^{-3}
PFIT3LS	3	1×10^{-5}	30940	6.034×10^{2}	2339	4.427×10^{-2}
PFIT4LS	3	1×10^{-5}	31645	2.140×10^{3}	6937	7.936×10^{-2}
QUARTC	100	1×10^{-5}	13738	1.501×10^{-4}	28793	1.977×10^{-6}
SINEVAL	2	1×10^{-5}	18967	3.956	3089	2.075×10^{-7}
SINQUAD	100	1×10^{-5}	5127	1.373×10^{-6}	16276	4.398×10^{-7}
SISSER	2	1×10^{-5}	657	3.088×10^{-7}	1337	5.212×10^{-11}
SPARSQUR	100	1×10^{-5}	7464	2.220×10^{-5}	19032	5.543×10^{-6}
TOINTGSS	100	1×10^{-5}	8221	2.253×10^{-8}	8385	4.000×10^{-9}
TQUARTIC	100	1×10^{-5}	12669	8.689×10^{-4}	15299	3.231×10^{-4}
TRIDIA	100	1×10^{-5}	20802	5.305×10^{-7}	107193	1.824×10^{-17}
WATSON	31	1×10^{-5}	7718	1.087×10^{-3}	16253	1.209×10^{-3}
WOODS	100	1×10^{-5}	9234	2.150×10^{-5}	24007	3.869×10^{-6}
ZANGWIL2	2	1×10^{-5}	657	2.019×10^{-9}	1347	-1.000×10^{-10}

Table B.18. Total number of function evaluations used and final accuracy achieved by central-difference L-BFGS method with different choices of the finite-difference interval.

			Fixed Interval		**Adaptive**	
Problem	n	ϵ_f	#Evals	$\phi(x) - \phi^*$	#Evals	$\phi(x) - \phi^*$
AIRCRFTB	5	1×10^{-7}	1585	6.912×10^{-12}	2929	2.059×10^{-22}
ALLINITU	4	1×10^{-7}	463	1.644×10^{-8}	1493	3.211×10^{-10}
ARWHEAD	100	1×10^{-7}	5391	1.192×10^{-8}	8298	2.636×10^{-9}
BARD	3	1×10^{-7}	669	2.756×10^{-9}	1357	4.989×10^{-9}
BDQRTIC	100	1×10^{-7}	11220	1.885×10^{-8}	27324	6.486×10^{-9}
BIGGS3	3	1×10^{-7}	669	1.115×10^{-8}	1357	5.031×10^{-10}
BIGGS5	5	1×10^{-7}	1530	2.281×10^{-8}	3911	4.135×10^{-9}
BIGGS6	6	1×10^{-7}	2022	-5.642×10^{-3}	5582	-5.649×10^{-3}
BOX2	2	1×10^{-7}	657	3.579×10^{-9}	678	1.419×10^{-9}
BOX3	3	1×10^{-7}	669	6.337×10^{-7}	1092	6.848×10^{-10}
BRKMCC	2	1×10^{-7}	657	2.043×10^{-10}	680	2.005×10^{-10}
BROWNAL	100	1×10^{-7}	3799	2.783×10^{-8}	24944	6.904×10^{-19}
BROWNDEN	4	1×10^{-7}	681	-3.638×10^{-10}	1695	-3.929×10^{-10}
CLIFF	2	1×10^{-7}	657	5.621×10^{-1}	10094	2.902×10^{2}
CRAGGLVY	100	1×10^{-7}	18747	5.581×10^{-7}	39527	3.028×10^{-8}
CUBE	2	1×10^{-7}	657	1.452×10^{-7}	1905	7.199×10^{-9}
DENSCHND	3	1×10^{-7}	1309	1.440×10^{-5}	2057	8.977×10^{-8}
DENSCHNE	3	1×10^{-7}	669	3.754×10^{-11}	2034	2.051×10^{-10}
DIXMAANH	300	1×10^{-7}	52791	4.060×10^{-7}	394134	0.000
DQRTIC	100	1×10^{-7}	14367	2.836×10^{-6}	29677	2.399×10^{-9}
EDENSCH	36	1×10^{-7}	2434	1.106×10^{-9}	6835	8.345×10^{-10}
EIGENALS	110	1×10^{-7}	86646	7.971×10^{-6}	194737	2.282×10^{-6}
EIGENBLS	110	1×10^{-7}	194875	1.129×10^{-6}	257660	9.239×10^{-4}
EIGENCLS	30	1×10^{-7}	7754	1.060×10^{-7}	17966	7.350×10^{-8}
ENGVAL1	100	1×10^{-7}	7210	5.117×10^{-9}	13746	7.956×10^{-9}
EXPFIT	2	1×10^{-7}	3032	6.553×10^{-5}	1341	8.708×10^{-10}
FLETCBV3	100	1×10^{-7}	101241	-2.737×10^{1}	693140	-8.437×10^{1}
FLETCHBV	100	1×10^{-7}	106670	1.635×10^{9}	410157	-1.297×10^{9}
FREUROTH	100	1×10^{-7}	6382	-3.050×10^{-9}	14775	-5.215×10^{-9}
GENROSE	100	1×10^{-7}	53767	2.215×10^{-8}	118006	2.538×10^{-8}
GULF	3	1×10^{-7}	1504	3.522×10^{-7}	2098	5.298×10^{-8}
HAIRY	2	1×10^{-7}	657	1.723×10^{-12}	1511	0.000
HELIX	3	1×10^{-7}	413	8.521×10^{2}	2825	5.755×10^{-13}
NCB20B	100	1×10^{-7}	31569	2.837×10^{-5}	73674	1.549×10^{-5}
NONDIA	100	1×10^{-7}	6618	6.417×10^{-8}	14890	6.216×10^{-8}
NONDQUAR	100	1×10^{-7}	77500	5.554×10^{-5}	239505	1.867×10^{-5}
OSBORNEA	5	1×10^{-7}	1333	2.245×10^{-5}	2162	1.562×10^{-1}
OSBORNEB	11	1×10^{-7}	2933	1.612×10^{-8}	6229	6.560×10^{-9}
PENALTY1	100	1×10^{-7}	8565	1.868×10^{-4}	76226	2.507×10^{-6}
PFIT1LS	3	1×10^{-7}	26224	4.862×10^{-4}	4269	2.294×10^{-5}
PFIT2LS	3	1×10^{-7}	28717	1.295×10^{-2}	14332	2.756×10^{-5}
PFIT3LS	3	1×10^{-7}	28777	5.848×10^{-2}	22899	1.676×10^{-5}
PFIT4LS	3	1×10^{-7}	29320	2.991×10^{-1}	25011	1.775×10^{-5}
QUARTC	100	1×10^{-7}	14367	2.836×10^{-6}	29677	2.399×10^{-9}
SINEVAL	2	1×10^{-7}	1722	1.177×10^{-11}	1879	1.601×10^{-10}
SINQUAD	100	1×10^{-7}	5544	5.534×10^{-9}	16166	3.215×10^{-9}
SISSER	2	1×10^{-7}	657	2.663×10^{-8}	703	6.479×10^{-9}
SPARSQUR	100	1×10^{-7}	10891	3.396×10^{-7}	24709	2.323×10^{-9}
TOINTGSS	100	1×10^{-7}	4087	4.042×10^{-9}	6671	4.000×10^{-9}
TQUARTIC	100	1×10^{-7}	9496	3.400×10^{-7}	19028	3.207×10^{-7}
TRIDIA	100	1×10^{-7}	30055	2.966×10^{-10}	114846	1.242×10^{-19}
WATSON	31	1×10^{-7}	8296	1.016×10^{-4}	21230	1.002×10^{-4}
WOODS	100	1×10^{-7}	8594	5.693×10^{-7}	20454	7.795×10^{-8}
ZANGWIL2	2	1×10^{-7}	657	-9.543×10^{-11}	1345	-1.000×10^{-10}

APPENDIX C

A Feasible Nonlinear Programming Approach for Constrained Derivative-Free Optimization

C.1. Numerical Results for Feasible Initial Points

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	3	5	0.29	1.000	1.309×10^{-14}	136	191	0.28	20	104	0.09	9.983×10^{-1}	6.579×10^{-10}
HS22	2	2	2	4	0.04	1.000	0.000	10	25	0.03	1	8	0.01	1.000	0.000
HS23	2	5	4	6	0.04	2.000	0.000	12	16	0.03	6	28	0.03	2.000	0.000
HS26	3	1	53	56	12.96	7.103×10^{-10}	2.220×10^{-16}	5245	39137	12.85	34	244	0.1	7.434×10^{-12}	6.661×10^{-16}
HS32	3	2	4	7	0.04	1.000	0.000	10	16	0.03	2	15	0.01	1.000	0.000
HS34	3	2	6	9	0.1	-8.340×10^{-1}	1.776×10^{-15}	32	70	0.08	7	44	0.02	-8.340×10^{-1}	0.000
HS40	4	3	22	26	0.36	-2.500×10^{-1}	1.110×10^{-16}	56	225	0.26	5	44	0.01	-2.500×10^{-1}	1.110×10^{-16}
HS44	4	6	3	8	0.04	-1.000(*)	0.000	12	23	0.03	5	36	0.03	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.67	1.989×10^{-7}	3.781×10^{-11}	1391	7878	3.47	72	696	0.41	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.17	1.056	2.398×10^{-14}	126	423	1.1	16	121	0.04	7.669×10^{-17}	4.441×10^{-16}
HS64	3	1	43	46	1.04	6.300×10^{3}	0.000	300	662	0.95	12	65	0.04	6.300×10^{3}	0.000
HS66	3	2	2	6	0.03	5.182×10^{-1}(*)	3.563×10^{-12}	7	12	0.02	7	43	0.03	5.182×10^{-1}	0.000
HS67	3	14	45	49	1.7	-1.162×10^{3}(*)	0.000	472	1408	1.59	14	75	0.03	-1.162×10^{3}	0.000
HS72	4	2	7	11	0.26	7.277×10^{2}	2.819×10^{-18}	44	99	0.21	8	55	0.03	7.277×10^{2}	4.337×10^{-19}
HS75	4	5	16	20	0.49	5.174×10^{3}	5.684×10^{-14}	97	180	0.43	6	43	0.04	5.174×10^{3}	8.527×10^{-14}
HS85	5	21	31	36	1.46	-2.216	0.000	357	903	1.31	11	85	0.06	-2.216	1.421×10^{-14}
HS87	6	4	9	15	0.74	8.997×10^{3}	2.274×10^{-13}	80	287	0.71	14	223	0.07	8.997×10^{3}	2.416×10^{-13}
HS88	2	1	5	7	0.4	1.363	0.000	92	318	0.39	27	156	0.08	1.363	2.893×10^{-17}
HS89	3	1	10	13	1.42	1.363	0.000	246	889	1.39	5	35	0.04	7.283×10^{-8}(†)	1.332×10^{-1}
HS90	4	1	12	16	4.22	1.363	0.000	738	3531	4.18	33	285	0.15	1.363	6.600×10^{-18}
HS93	6	2	42	48	2.9	1.351×10^{2}	4.800×10^{-12}	517	2433	2.61	26	295	0.13	1.351×10^{2}	0.000
HS98	6	4	3	10	0.05	3.136(*)	1.776×10^{-15}	5	8	0.02	4	40	0.02	3.136	1.776×10^{-15}
HS100	7	4	106	113	29.46	6.806×10^{2}	0.000	3280	22448	28.72	51	587	0.21	6.806×10^{2}	7.161×10^{-15}
HS101	7	5	103	110	19.42	1.810×10^{3}	3.955×10^{-16}	2616	15697	18.39	35	354	0.19	1.810×10^{3}	0.000
HS102	7	5	103	110	15.11	9.119×10^{2}	3.816×10^{-16}	1985	10240	13.86	21	254	0.13	9.119×10^{2}	2.498×10^{-16}
HS103	7	5	56	63	5.5	5.437×10^{2}	1.318×10^{-16}	825	4072	4.59	20	238	0.1	5.437×10^{2}	2.359×10^{-16}
HS104	8	5	245	253	43.5	3.951	9.714×10^{-17}	6813	37268	41.25	30	359	0.1	3.951	2.082×10^{-16}
LOADBAL	31	31	1	15499	0.93	1.547(***)	1.798×10^{308}	0	1	0.0	40	1380	0.21	4.529×10^{-1}	2.665×10^{-15}
OPTPRLOC	30	30	9	39	1.0	-1.642×10^{1}	1.990×10^{-13}	54	127	0.45	16	583	0.13	-1.642×10^{1}	1.421×10^{-13}
CB3	3	3	1	4	0.01	2.000	0.000	2	3	0.0	2	15	0.01	2.000	0.000
CRESC50	6	100	1000	1006	215.01	5.939×10^{-1}(**)	5.482×10^{-13}	26160	96691	210.07	438	5787	1.76	5.946×10^{-1}	1.188×10^{-14}
DEMBO7	16	20	19	35	2.37	1.748×10^{2}	4.073×10^{-15}	302	795	2.07	17	354	0.16	1.749×10^{2}	6.661×10^{-16}
DNIEPER	57	24	24	81	7.72	1.874×10^{4}	5.684×10^{-14}	273	1094	3.25	8	534	0.11	1.874×10^{4}	5.684×10^{-14}
EXPFITA	5	22	1	10999	0.6	2.999×10^{1}(***)	1.798×10^{308}	0	1	0.0	19	140	0.05	1.137×10^{-3}	1.421×10^{-14}
HIMMELBI	100	12	114	215	72.23	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12586	67.95	72	7487	0.72	-1.736×10^{3}	1.066×10^{-14}
SYNTHES1	6	6	1	2999	0.21	1.000×10^{1}(***)	1.798×10^{308}	0	1	0.0	9	81	0.03	7.593×10^{-1}	1.110×10^{-16}
TWOBARS	2	2	7	9	0.1	1.509	0.000	24	108	0.09	9	57	0.03	1.509	4.441×10^{-16}
DIPIGRI	7	4	116	123	85.85	6.806×10^{2}	3.581×10^{-11}	7461	26264	84.59	48	581	0.18	6.806×10^{2}	0.000

Table C.1. Noiseless Problems with Feasible x_0; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	2	4	0.2	1.000	1.310×10^{-14}	75	104	0.19	8	36	0.02	1.080	0.000
HS22	2	2	1	3	0.03	1.000	1.789×10^{-12}	9	23	0.02	0	3	0.0	1.000	1.843×10^{-14}
HS23	2	5	3	5	0.05	2.000	0.000	11	14	0.03	3	16	0.01	2.040	0.000
HS26	3	1	4	7	0.09	1.996×10^{-2}	9.592×10^{-14}	32	66	0.07	14	87	0.04	4.252×10^{-7}	9.633×10^{-7}
HS32	3	2	4	7	0.09	1.000	0.000	10	16	0.04	2	15	0.02	1.000	0.000
HS34	3	2	5	8	0.09	-7.820×10^{-1}	1.066×10^{-14}	31	68	0.08	4	29	0.02	-8.266×10^{-1}	0.000
HS40	4	3	1	5	0.02	-2.454×10^{-1}	2.554×10^{-15}	5	15	0.01	0	4	0.0	-2.454×10^{-1}	7.334×10^{-9}
HS44	4	6	3	8	0.09	-1.000(*)	0.000	12	23	0.07	4	30	0.02	-1.500×10^{1}	0.000
HS47	5	3	6	11	0.14	5.639×10^{-2}	1.466×10^{-12}	46	118	0.12	17	134	0.04	4.638×10^{-8}	3.167×10^{-9}
HS50	5	3	19	24	1.26	1.056	2.398×10^{-14}	126	423	1.19	9	71	0.03	4.941×10^{-2}	4.441×10^{-16}
HS64	3	1	1	4	0.05	6.813×10^{3}	4.742×10^{-11}	14	72	0.05	0	3	0.0	6.813×10^{3}	0.000
HS66	3	2	1	4	0.02	5.800×10^{-1}	3.563×10^{-12}	6	10	0.01	0	3	0.0	5.800×10^{-1}	0.000
HS67	3	14	31	34	1.91	-1.053×10^{3}	0.000	402	1319	1.82	8	45	0.02	-1.157×10^{3}	0.000
HS72	4	2	1	5	0.01	7.407×10^{2}	0.000	2	5	0.01	0	4	0.0	7.407×10^{2}	0.000
HS75	4	5	1	5	0.02	5.350×10^{3}	1.259×10^{-8}	4	8	0.01	5	37	0.03	5.174×10^{3}	7.739×10^{-8}
HS85	5	21	29	34	1.22	-2.061	1.421×10^{-14}	348	880	1.04	10	78	0.05	-2.216	2.389×10^{-8}
HS87	6	4	1	7	0.07	8.997×10^{3}	2.274×10^{-13}	9	24	0.06	0	5	0.0	8.997×10^{3}	2.956×10^{-12}
HS88	2	1	1	3	0.07	1.385	2.158×10^{-14}	19	41	0.07	0	3	0.0	1.385	0.000
HS89	3	1	2	5	0.29	1.365	1.123×10^{-16}	57	205	0.28	5	35	0.03	7.283×10^{-8}(†)	1.332×10^{-1}
HS90	4	1	2	6	0.53	1.446	6.661×10^{-16}	86	454	0.52	24	228	0.11	1.362	5.472×10^{-7}
HS93	6	2	1	7	0.11	1.371×10^{2}	2.946×10^{-11}	10	24	0.11	0	5	0.0	1.371×10^{2}	0.000
HS98	6	4	2	8	0.03	3.136	0.000	4	6	0.02	3	32	0.02	3.136	0.000
HS100	7	4	1	8	0.12	7.140×10^{2}	0.000	22	79	0.12	0	5	0.0	7.140×10^{2}	0.000
HS101	7	5	26	33	5.24	1.982×10^{3}	3.053×10^{-16}	740	4383	5.04	9	106	0.04	1.848×10^{3}	0.000
HS102	7	5	34	41	4.55	9.948×10^{2}	2.776×10^{-17}	641	3320	4.29	11	159	0.05	9.191×10^{2}	0.000
HS103	7	5	15	22	1.69	5.623×10^{2}	8.913×10^{-12}	276	1263	1.63	16	202	0.06	5.437×10^{2}	1.040×10^{-7}
HS104	8	5	1	9	0.11	4.200	1.459×10^{-11}	14	41	0.11	0	6	0.0	4.200	0.000
LOADBAL	31	31	1	15499	0.97	1.547(***)	1.798×10^{308}	0	1	0.0	6	231	0.04	5.252×10^{-1}	1.710×10^{-14}
OPTPRLOC	30	30	5	35	0.46	-1.567×10^{1}	2.220×10^{-16}	44	104	0.4	2	103	0.04	-1.587×10^{1}	6.661×10^{-16}
CB3	3	3	1	4	0.01	2.000	0.000	2	3	0.01	1	10	0.01	2.000	7.270×10^{-13}
CRESC50	6	100	16	22	2.44	6.057×10^{-1}	9.032×10^{-11}	344	1528	2.38	58	700	0.32	6.155×10^{-1}	0.000
DEMBO7	16	20	13	29	1.39	1.879×10^{2}	5.917×10^{-8}	198	507	1.19	15	318	0.1	1.749×10^{2}	1.869×10^{-13}
DNIEPER	57	24	4	61	1.82	1.941×10^{4}	4.264×10^{-11}	96	457	1.35	2	177	0.03	1.874×10^{4}	1.800×10^{-7}
EXPFITA	5	22	1	10999	0.61	2.999×10^{1}(***)	1.798×10^{308}	0	1	0.0	7	56	0.02	9.889×10^{-2}	1.354×10^{-14}
HIMMELBI	100	12	114	215	70.47	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12586	68.05	3	408	0.09	-1.643×10^{3}	0.000
SYNTHES1	6	6	1	2999	0.22	1.000×10^{1}(***)	1.798×10^{308}	0	1	0.0	7	65	0.02	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	2	4	0.05	1.523	7.885×10^{-9}	13	38	0.04	7	49	0.05	1.509	2.179×10^{-9}
DIPIGRI	7	4	1	8	0.14	7.140×10^{2}	0.000	22	79	0.13	0	5	0.0	7.140×10^{2}	0.000

Table C.2. Noiseless Problems with Feasible x_0. $\tau = 10^{-1}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	2	4	0.18	1.000	1.310×10^{-14}	75	104	0.17	18	88	0.08	1.001	0.000
HS22	2	2	1	3	0.03	1.000	1.789×10^{-12}	9	23	0.02	0	3	0.0	1.000	1.843×10^{-14}
HS23	2	5	3	5	0.04	2.000	0.000	11	14	0.03	4	20	0.02	2.000	0.000
HS26	3	1	36	39	1.85	1.737×10^{-4}	1.040×10^{-12}	754	4338	1.77	14	87	0.03	4.252×10^{-7}	9.633×10^{-7}
HS32	3	2	4	7	0.05	1.000	0.000	10	16	0.03	2	15	0.02	1.000	0.000
HS34	3	2	6	9	0.11	-8.340×10^{-1}	1.776×10^{-15}	32	70	0.08	6	39	0.01	-8.340×10^{-1}	5.446×10^{-9}
HS40	4	3	2	6	0.03	-2.491×10^{-1}	2.189×10^{-9}	8	21	0.02	3	32	0.02	-2.500×10^{-1}	2.040×10^{-9}
HS44	4	6	3	8	0.05	-1.000(*)	0.000	12	23	0.03	4	30	0.03	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.41	1.989×10^{-7}	3.781×10^{-11}	1391	7878	3.33	72	696	0.35	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.02	1.056	2.398×10^{-14}	126	423	0.96	11	85	0.03	1.339×10^{-6}	8.882×10^{-16}
HS64	3	1	13	16	0.46	6.303×10^{3}	1.443×10^{-15}	126	399	0.44	7	40	0.02	6.300×10^{3}	7.051×10^{-10}
HS66	3	2	2	5	0.02	5.182×10^{-1}	3.563×10^{-12}	7	12	0.02	3	21	0.02	5.186×10^{-1}	0.000
HS67	3	14	38	41	1.51	-1.162×10^{3}	0.000	448	1377	1.42	9	50	0.02	-1.161×10^{3}	0.000
HS72	4	2	5	9	0.17	7.284×10^{2}	0.000	38	90	0.16	3	24	0.02	7.277×10^{2}	1.663×10^{-7}
HS75	4	5	16	20	0.32	5.174×10^{3}	5.684×10^{-14}	97	180	0.28	5	37	0.04	5.174×10^{3}	7.739×10^{-8}
HS85	5	21	31	36	0.95	-2.216	0.000	357	903	0.87	10	78	0.05	-2.216	2.389×10^{-8}
HS87	6	4	1	7	0.07	8.997×10^{3}	2.274×10^{-13}	9	24	0.06	0	5	0.0	8.997×10^{3}	2.956×10^{-12}
HS88	2	1	2	4	0.2	1.363	9.216×10^{-16}	41	139	0.19	23	140	0.06	1.362	4.623×10^{-7}
HS89	3	1	4	7	0.49	1.363	3.663×10^{-16}	101	374	0.48	5	35	0.04	7.283×10^{-8}(†)	1.332×10^{-1}
HS90	4	1	6	10	1.54	1.363	5.260×10^{-17}	277	1541	1.52	24	228	0.14	1.362	5.472×10^{-7}
HS93	6	2	21	27	1.5	1.352×10^{2}	4.695×10^{-11}	333	1706	1.43	8	87	0.04	1.351×10^{2}	0.000
HS98	6	4	2	8	0.04	3.136	0.000	4	6	0.01	3	32	0.02	3.136	0.000
HS100	7	4	51	58	15.18	6.813×10^{2}	1.388×10^{-17}	1609	11182	15.01	3	55	0.02	6.811×10^{2}	0.000
HS101	7	5	44	51	7.81	1.812×10^{3}	3.006×10^{-10}	1239	7244	7.65	18	198	0.1	1.810×10^{3}	2.139×10^{-8}
HS102	7	5	69	76	10.13	9.125×10^{2}	4.272×10^{-13}	1483	7663	9.89	15	200	0.06	9.119×10^{2}	5.354×10^{-7}
HS103	7	5	37	44	3.94	5.437×10^{2}	7.980×10^{-16}	671	3465	3.8	16	202	0.08	5.437×10^{2}	1.040×10^{-7}
HS104	8	5	47	55	6.18	3.952	5.218×10^{-15}	1210	6047	5.99	20	259	0.05	3.951	1.537×10^{-7}
LOADBAL	31	31	1	15499	0.9	1.547(***)	1.798×10^{308}	0	1	0.0	9	330	0.05	4.531×10^{-1}	1.421×10^{-14}
OPTPRLOC	30	30	6	36	0.44	-1.642×10^{1}	0.000	47	111	0.39	3	136	0.06	-1.641×10^{1}	0.000
CB3	3	3	1	4	0.01	2.000	0.000	2	3	0.01	1	10	0.0	2.000	7.270×10^{-13}
CRESC50	6	100	68	74	18.0	5.942×10^{-1}	0.000	2416	11257	17.76	438	5787	1.81	5.946×10^{-1}	1.188×10^{-14}
DEMBO7	16	20	18	34	2.11	1.749×10^{2}	1.067×10^{-10}	301	793	2.03	15	318	0.1	1.749×10^{2}	1.869×10^{-13}
DNIEPER	57	24	5	62	1.42	1.874×10^{4}	1.405×10^{-9}	100	465	1.35	2	177	0.06	1.874×10^{4}	1.800×10^{-7}
EXPFITA	5	22	1	10999	0.58	2.999×10^{1}(***)	1.798×10^{308}	0	1	0.0	12	91	0.03	1.355×10^{-3}	6.750×10^{-14}
HIMMELBI	100	12	114	215	68.58	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12586	67.1	12	1326	0.26	-1.734×10^{3}	5.684×10^{-14}
SYNTHES1	6	6	1	2999	0.21	1.000×10^{1}(***)	1.798×10^{308}	0	1	0.0	7	65	0.02	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	3	5	0.06	1.509	4.742×10^{-11}	16	55	0.05	7	49	0.04	1.509	2.179×10^{-9}
DIPIGRI	7	4	51	58	11.33	6.812×10^{2}	6.939×10^{-18}	1491	9509	11.16	3	55	0.01	6.811×10^{2}	0.000

Table C.3. Noiseless Problems with Feasible x_0. $\tau = 10^{-3}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	3	5	0.29	1.000	1.309×10^{-14}	136	191	0.28	19	100	0.1	9.983×10^{-1}	6.579×10^{-10}
HS22	2	2	1	3	0.03	1.000	1.789×10^{-12}	9	23	0.02	0	3	0.0	1.000	1.843×10^{-14}
HS23	2	5	3	5	0.03	2.000	0.000	11	14	0.02	5	24	0.02	2.000	0.000
HS26	3	1	38	41	8.14	8.872×10^{-7}	4.441×10^{-16}	3356	26976	8.07	14	87	0.03	4.252×10^{-7}	9.633×10^{-7}
HS32	3	2	4	7	0.04	1.000	0.000	10	16	0.03	2	15	0.02	1.000	0.000
HS34	3	2	6	9	0.11	-8.340×10^{-1}	1.776×10^{-15}	32	70	0.08	6	39	0.01	-8.340×10^{-1}	5.446×10^{-9}
HS40	4	3	4	8	0.06	-2.500×10^{-1}	1.058×10^{-12}	13	32	0.04	3	32	0.03	-2.500×10^{-1}	2.040×10^{-9}
HS44	4	6	3	8	0.05	-1.000(*)	0.000	12	23	0.04	4	30	0.02	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.42	1.989×10^{-7}	3.781×10^{-11}	1391	7878	3.35	72	696	0.38	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.11	1.056	2.398×10^{-14}	126	423	1.06	11	85	0.04	1.339×10^{-6}	8.882×10^{-16}
HS64	3	1	17	20	0.55	6.300×10^{3}	0.000	156	435	0.52	7	40	0.02	6.300×10^{3}	7.051×10^{-10}
HS66	3	2	2	5	0.03	5.182×10^{-1}	3.563×10^{-12}	7	12	0.02	4	27	0.02	5.182×10^{-1}	0.000
HS67	3	14	39	42	1.45	-1.162×10^{3}	0.000	450	1380	1.38	10	55	0.03	-1.162×10^{3}	0.000
HS72	4	2	6	10	0.24	7.277×10^{2}	6.263×10^{-14}	43	97	0.22	4	31	0.02	7.277×10^{2}	1.572×10^{-10}
HS75	4	5	16	20	0.31	5.174×10^{3}	5.684×10^{-14}	97	180	0.27	5	37	0.03	5.174×10^{3}	7.739×10^{-8}
HS85	5	21	31	36	0.92	-2.216	0.000	357	903	0.83	10	78	0.05	-2.216	2.389×10^{-8}
HS87	6	4	2	8	0.11	8.997×10^{3}	7.844×10^{-12}	15	43	0.1	3	33	0.03	8.997×10^{3}	2.396×10^{-8}
HS88	2	1	4	6	0.31	1.363	1.301×10^{-14}	69	218	0.3	23	140	0.08	1.362	4.623×10^{-7}
HS89	3	1	5	8	0.55	1.363	0.000	113	432	0.54	5	35	0.04	7.283×10^{-8}(†)	1.332×10^{-1}
HS90	4	1	11	15	3.6	1.363	8.118×10^{-18}	718	3450	3.57	24	228	0.1	1.362	5.472×10^{-7}
HS93	6	2	25	31	1.57	1.351×10^{2}	3.331×10^{-16}	371	1842	1.5	8	87	0.05	1.351×10^{2}	0.000
HS98	6	4	2	8	0.03	3.136	0.000	4	6	0.02	3	32	0.02	3.136	0.000
HS100	7	4	89	96	24.86	6.806×10^{2}	0.000	2860	19399	24.54	14	212	0.07	6.806×10^{2}	0.000
HS101	7	5	71	78	12.74	1.810×10^{3}	2.776×10^{-16}	2004	11919	12.48	18	198	0.08	1.810×10^{3}	2.139×10^{-8}
HS102	7	5	78	85	10.45	9.119×10^{2}	0.000	1601	8225	10.11	15	200	0.07	9.119×10^{2}	5.354×10^{-7}
HS103	7	5	39	46	4.22	5.437×10^{2}	6.106×10^{-16}	685	3515	4.09	16	202	0.09	5.437×10^{2}	1.040×10^{-7}
HS104	8	5	80	88	13.31	3.951	0.000	2262	12381	13.0	20	259	0.04	3.951	1.537×10^{-7}
LOADBAL	31	31	1	15499	0.88	1.547(***)	1.798×10^{308}	0	1	0.0	17	594	0.09	4.529×10^{-1}	1.483×10^{-14}
OPTPRLOC	30	30	8	38	0.52	-1.642×10^{1}	4.504×10^{-10}	53	125	0.44	13	487	0.11	-1.642×10^{1}	4.092×10^{-8}
CB3	3	3	1	4	0.01	2.000	0.000	2	3	0.01	1	10	0.0	2.000	7.270×10^{-13}
CRESC50	6	100	1000	1006	214.64	5.939×10^{-1}(**)	5.482×10^{-13}	26160	96691	211.17	438	5787	1.92	5.946×10^{-1}	1.188×10^{-14}
DEMBO7	16	20	19	35	2.06	1.748×10^{2}	4.073×10^{-15}	302	795	1.97	17	354	0.14	1.749×10^{2}	6.661×10^{-16}
DNIEPER	57	24	5	62	1.4	1.874×10^{4}	1.405×10^{-9}	100	465	1.36	2	177	0.06	1.874×10^{4}	1.800×10^{-7}
EXPFITA	5	22	1	10999	0.63	2.999×10^{1}(***)	1.798×10^{308}	0	1	0.0	13	98	0.03	1.137×10^{-3}	6.047×10^{-12}
HIMMELBI	100	12	114	215	68.51	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12586	67.29	31	3273	0.41	-1.736×10^{3}	1.155×10^{-14}
SYNTHES1	6	6	1	2999	0.21	1.000×10^{1}(***)	1.798×10^{308}	0	1	0.0	7	65	0.02	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	3	5	0.06	1.509	4.742×10^{-11}	16	55	0.05	7	49	0.03	1.509	2.179×10^{-9}
DIPIGRI	7	4	87	94	78.71	6.806×10^{2}	1.265×10^{-13}	7019	23030	78.4	14	212	0.06	6.806×10^{2}	0.000

Table C.4. Noiseless Problems with Feasible x_0. $\tau = 10^{-5}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	3	5	0.29	1.000	1.309×10^{-14}	136	191	0.28	19	100	0.1	9.983×10^{-1}	6.579×10^{-10}
HS22	2	2	1	3	0.03	1.000	1.789×10^{-12}	9	23	0.02	0	3	0.0	1.000	1.843×10^{-14}
HS23	2	5	3	5	0.04	2.000	0.000	11	14	0.03	5	24	0.03	2.000	0.000
HS26	3	1	43	46	12.4	2.878×10^{-8}	1.110×10^{-16}	5194	38850	12.32	28	203	0.08	2.719×10^{-11}	8.518×10^{-7}
HS32	3	2	4	7	0.05	1.000	0.000	10	16	0.03	2	15	0.02	1.000	0.000
HS34	3	2	6	9	0.09	-8.340×10^{-1}	1.776×10^{-15}	32	70	0.07	6	39	0.02	-8.340×10^{-1}	5.446×10^{-9}
HS40	4	3	4	8	0.05	-2.500×10^{-1}	1.058×10^{-12}	13	32	0.04	3	32	0.05	-2.500×10^{-1}	2.040×10^{-9}
HS44	4	6	3	8	0.05	-1.000(*)	0.000	12	23	0.03	4	30	0.02	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.44	1.989×10^{-7}	3.781×10^{-11}	1391	7878	3.36	72	696	0.36	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.12	1.056	2.398×10^{-14}	126	423	1.06	13	100	0.03	2.358×10^{-8}	4.441×10^{-16}
HS64	3	1	17	20	0.55	6.300×10^{3}	0.000	156	435	0.52	7	40	0.02	6.300×10^{3}	7.051×10^{-10}
HS66	3	2	2	5	0.03	5.182×10^{-1}	3.563×10^{-12}	7	12	0.02	5	33	0.02	5.182×10^{-1}	5.154×10^{-11}
HS67	3	14	40	43	1.46	-1.162×10^{3}	2.274×10^{-12}	458	1389	1.36	11	60	0.03	-1.162×10^{3}	1.364×10^{-12}
HS72	4	2	7	11	0.21	7.277×10^{2}	2.819×10^{-18}	44	99	0.2	4	31	0.02	7.277×10^{2}	1.572×10^{-10}
HS75	4	5	16	20	0.31	5.174×10^{3}	5.684×10^{-14}	97	180	0.28	5	37	0.04	5.174×10^{3}	7.739×10^{-8}
HS85	5	21	31	36	1.41	-2.216	0.000	357	903	1.3	10	78	0.03	-2.216	2.389×10^{-8}
HS87	6	4	9	15	0.7	8.997×10^{3}	2.274×10^{-13}	80	287	0.67	6	73	0.03	8.997×10^{3}	1.152×10^{-8}
HS88	2	1	4	6	0.31	1.363	1.301×10^{-14}	69	218	0.3	23	140	0.07	1.362	4.623×10^{-7}
HS89	3	1	6	9	0.68	1.363	5.535×10^{-16}	142	532	0.66	5	35	0.04	7.283×10^{-8}(†)	1.332×10^{-1}
HS90	4	1	11	15	3.6	1.363	8.118×10^{-18}	718	3450	3.57	24	228	0.1	1.362	5.472×10^{-7}
HS93	6	2	36	42	2.18	1.351×10^{2}	0.000	483	2291	2.06	9	96	0.05	1.351×10^{2}	0.000
HS98	6	4	2	8	0.02	3.136	0.000	4	6	0.02	3	32	0.01	3.136	0.000
HS100	7	4	106	113	28.46	6.806×10^{2}	0.000	3280	22448	28.08	28	362	0.19	6.806×10^{2}	5.093×10^{-11}
HS101	7	5	90	97	15.48	1.810×10^{3}	2.742×10^{-12}	2377	14118	15.15	20	216	0.1	1.810×10^{3}	1.032×10^{-8}
HS102	7	5	89	96	12.09	9.119×10^{2}	2.012×10^{-16}	1847	9679	11.74	15	200	0.08	9.119×10^{2}	5.354×10^{-7}
HS103	7	5	42	49	4.08	5.437×10^{2}	4.302×10^{-16}	713	3618	3.92	16	202	0.08	5.437×10^{2}	1.040×10^{-7}
HS104	8	5	88	96	13.71	3.951	4.441×10^{-16}	2400	13125	13.38	20	259	0.05	3.951	1.537×10^{-7}
LOADBAL	31	31	1	15499	0.89	1.547(***)	1.798×10^{308}	0	1	0.0	18	627	0.09	4.529×10^{-1}	8.549×10^{-15}
OPTPRLOC	30	30	8	38	0.51	-1.642×10^{1}	4.504×10^{-10}	53	125	0.43	13	487	0.11	-1.642×10^{1}	4.092×10^{-8}
CB3	3	3	1	4	0.01	2.000	0.000	2	3	0.0	1	10	0.0	2.000	7.270×10^{-13}
CRESC50	6	100	1000	1006	214.17	5.939×10^{-1}(**)	5.482×10^{-13}	26160	96691	210.8	438	5787	1.87	5.946×10^{-1}	1.188×10^{-14}
DEMBO7	16	20	19	35	2.11	1.748×10^{2}	4.073×10^{-15}	302	795	2.04	17	354	0.15	1.749×10^{2}	6.661×10^{-16}
DNIEPER	57	24	23	80	3.43	1.874×10^{4}	5.684×10^{-14}	272	1088	3.27	6	416	0.08	1.874×10^{4}	5.623×10^{-7}
EXPFITA	5	22	1	10999	0.6	2.999×10^{1}(***)	1.798×10^{308}	0	1	0.0	14	105	0.04	1.137×10^{-3}	1.776×10^{-14}
HIMMELBI	100	12	114	215	69.11	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12586	68.01	41	4300	0.52	-1.736×10^{3}	7.105×10^{-15}
SYNTHES1	6	6	1	2999	0.23	1.000×10^{1}(***)	1.798×10^{308}	0	1	0.0	7	65	0.02	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	5	7	0.1	1.509	0.000	21	84	0.08	7	49	0.03	1.509	2.179×10^{-9}
DIPIGRI	7	4	96	103	80.88	6.806×10^{2}	4.547×10^{-13}	7235	24605	80.56	28	365	0.1	6.806×10^{2}	7.237×10^{-11}

Table C.5. Noiseless Problems with Feasible x_0. $\tau = 10^{-7}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

C.2. Numerical Results for Infeasible Initial Point

We now include the cost of obtaining an initial feasible point for `FIBO`. In other words, the CPU time and number of constraint evaluations associated with obtaining a feasible starting point are counted. Simple comparison with the previous table indicates that only a few constraint evaluations are needed in order to obtain a feasible solution. As for `KNITRO`, we instead run from the given initial point x_0, which is potentially infeasible.

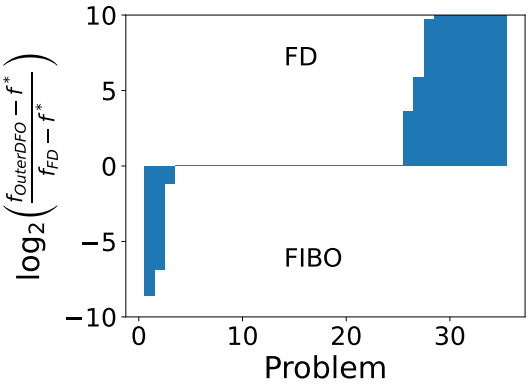

Figure C.1. Infeasible x_0. Log-ratio Plot for Comparing the Final Accuracy (4.3.1) of `FIBO` and `FD`.

As indicated by Figures C.1-C.3, the conclusions remain the same as in Section 4.3 and the cost of obtaining a feasible starting point is quite negligible.

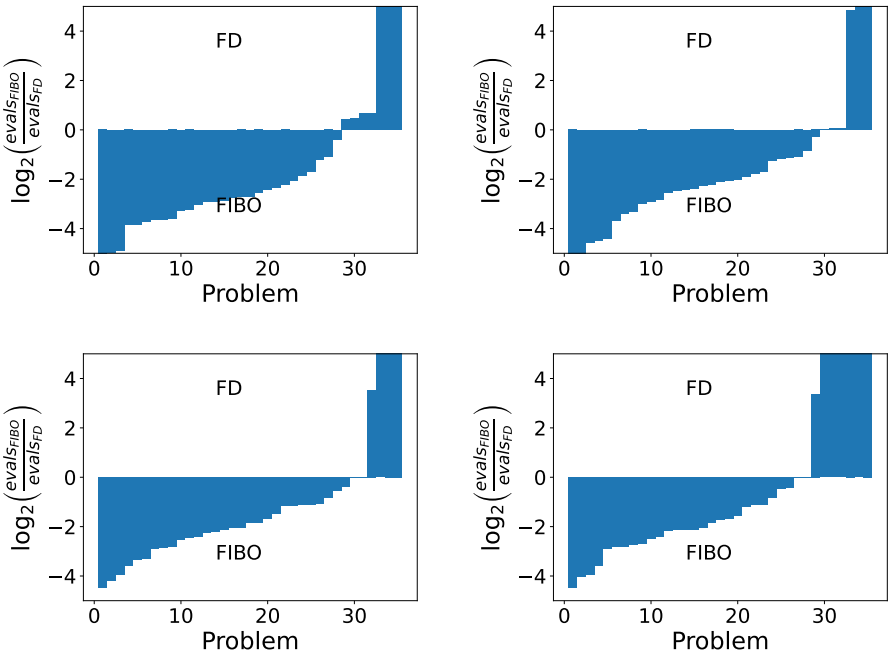

Figure C.2. Infeasible x_0. Log-ratio plot comparing `FIBO` and `FD` in terms of the number of function evaluations to satisfy (4.3.2) for $\tau = 10^{-1}$ (upper left), 10^{-3} (upper right), 10^{-5} (bottom left), 10^{-7} (bottom right).

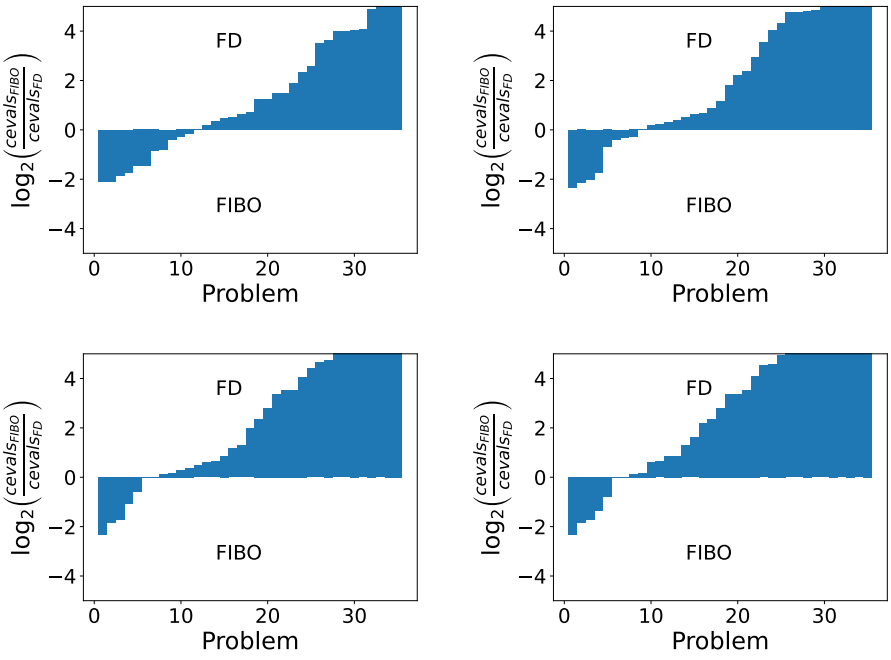

Figure C.3. Infeasible x_0. Log-ratio plot comparing `FIBO` and `FD` in terms of the number of constraint evaluations to satisfy (4.3.2) for $\tau = 10^{-1}$ (upper left), 10^{-3} (upper right), 10^{-5} (bottom left), 10^{-7} (bottom right).

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	3	5	0.32	1.000	1.309×10^{-14}	136	193	0.28	20	106	0.09	9.983×10^{-1}	6.579×10^{-10}
HS22	2	2	2	4	0.05	1.000	0.000	10	27	0.03	5	24	0.01	1.000	0.000
HS23	2	5	4	6	0.05	2.000	0.000	12	18	0.03	7	32	0.01	2.000	0.000
HS26	3	1	53	56	13.0	7.103×10^{-10}	2.220×10^{-16}	5245	39138	12.85	34	244	0.11	7.434×10^{-12}	6.661×10^{-16}
HS32	3	2	4	7	0.05	1.000	0.000	10	17	0.03	2	15	0.01	1.000	0.000
HS34	3	2	6	9	0.12	-8.340×10^{-1}	1.776×10^{-15}	32	71	0.08	7	44	0.02	-8.340×10^{-1}	0.000
HS40	4	3	22	26	0.37	-2.500×10^{-1}	1.110×10^{-16}	56	229	0.26	7	51	0.02	-2.500×10^{-1}	1.665×10^{-16}
HS44	4	6	3	8	0.06	-1.000(*)	0.000	12	24	0.03	5	36	0.01	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.67	1.989×10^{-7}	3.781×10^{-11}	1391	7879	3.47	72	696	0.35	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.18	1.056	2.398×10^{-14}	126	424	1.1	16	121	0.04	7.669×10^{-17}	4.441×10^{-16}
HS64	3	1	43	46	1.07	6.300×10^{3}	0.000	300	674	0.95	16	89	0.04	6.300×10^{3}	1.110×10^{-16}
HS66	3	2	2	6	0.04	5.182×10^{-1}(*)	3.563×10^{-12}	7	13	0.02	7	43	0.01	5.182×10^{-1}	0.000
HS67	3	14	45	49	1.71	-1.162×10^{3}(*)	0.000	472	1409	1.59	14	75	0.03	-1.162×10^{3}	0.000
HS72	4	2	7	11	0.32	7.277×10^{2}	2.819×10^{-18}	44	111	0.21	15	96	0.05	7.277×10^{2}	1.301×10^{-18}
HS75	4	5	16	20	0.54	5.174×10^{3}	5.684×10^{-14}	97	185	0.43	7	48	0.03	5.174×10^{3}	2.842×10^{-14}
HS85	5	21	31	36	1.47	-2.216	0.000	357	904	1.31	11	85	0.03	-2.216	1.421×10^{-14}
HS87	6	4	9	15	0.76	8.997×10^{3}	2.274×10^{-13}	80	290	0.71	14	216	0.07	8.997×10^{3}	2.274×10^{-13}
HS88	2	1	5	7	0.43	1.363	0.000	92	329	0.39	19	106	0.06	1.363	0.000
HS89	3	1	10	13	1.52	1.363	0.000	246	1033	1.39	28	172	0.09	1.363	9.636×10^{-18}
HS90	4	1	12	16	4.26	1.363	0.000	738	3541	4.18	48	426	0.24	1.363	0.000
HS93	6	2	42	48	2.9	1.351×10^{2}	4.800×10^{-12}	517	2434	2.61	26	295	0.08	1.351×10^{2}	0.000
HS98	6	4	3	10	0.09	3.136(*)	1.776×10^{-15}	5	13	0.02	6	56	0.01	3.136	0.000
HS100	7	4	106	113	29.47	6.806×10^{2}	0.000	3280	22449	28.72	51	587	0.16	6.806×10^{2}	7.161×10^{-15}
HS101	7	5	103	110	19.45	1.810×10^{3}	3.955×10^{-16}	2616	15705	18.39	56	582	0.22	1.810×10^{3}	0.000
HS102	7	5	103	110	15.14	9.119×10^{2}	3.816×10^{-16}	1985	10248	13.86	37	386	0.13	9.119×10^{2}	2.498×10^{-16}
HS103	7	5	56	63	5.53	5.437×10^{2}	1.318×10^{-16}	825	4079	4.59	28	284	0.08	5.437×10^{2}	2.220×10^{-16}
HS104	8	5	245	253	43.52	3.951	9.714×10^{-17}	6813	37274	41.25	17	190	0.04	3.951	1.943×10^{-16}
LOADBAL	31	31	1	15499	0.94	1.547(***)	1.798×10^{308}	0	2	0.0	40	1380	0.19	4.529×10^{-1}	2.665×10^{-15}
OPTPRLOC	30	30	9	39	1.01	-1.642×10^{1}	1.990×10^{-13}	54	130	0.45	12	456	0.06	-1.642×10^{1}	1.279×10^{-13}
CB3	3	3	1	4	0.03	2.000	0.000	2	9	0.0	7	40	0.01	2.000	0.000
CRESC50	6	100	1000	1006	215.05	5.939×10^{-1}(**)	5.482×10^{-13}	26160	96697	210.07	456	6023	2.46	5.948×10^{-1}	4.547×10^{-13}
DEMBO7	16	20	19	35	2.42	1.748×10^{2}	4.073×10^{-15}	302	800	2.07	45	893	0.2	1.748×10^{2}	3.608×10^{-16}
DNIEPER	57	24	24	81	7.75	1.874×10^{4}	5.684×10^{-14}	273	1097	3.25	6	418	0.07	1.874×10^{4}	5.684×10^{-14}
EXPFITA	5	22	1	10999	0.62	2.999×10^{1}(***)	1.798×10^{308}	0	2	0.0	19	140	0.05	1.137×10^{-3}	1.421×10^{-14}
HIMMELBI	100	12	114	215	72.25	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12588	67.95	96	9950	1.01	-1.736×10^{3}	2.842×10^{-14}
SYNTHES1	6	6	1	2999	0.24	1.000×10^{1}(***)	1.798×10^{308}	0	2	0.0	9	81	0.02	7.593×10^{-1}	1.110×10^{-16}
TWOBARS	2	2	7	9	0.13	1.509	0.000	24	113	0.09	11	57	0.02	1.509	2.220×10^{-16}
DIPIGRI	7	4	116	123	85.87	6.806×10^{2}	3.581×10^{-11}	7461	26265	84.59	48	581	0.16	6.806×10^{2}	0.000

Table C.6. Noiseless Problems with Infeasible x_0; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	2	4	0.2	1.000	1.310×10^{-14}	75	106	0.19	8	38	0.02	1.080	0.000
HS22	2	2	1	3	0.04	1.000	1.789×10^{-12}	9	25	0.02	3	16	0.01	1.000	1.231×10^{-6}
HS23	2	5	3	5	0.07	2.000	0.000	11	16	0.03	3	16	0.01	2.075	0.000
HS26	3	1	4	7	0.09	1.996×10^{-2}	9.592×10^{-14}	32	67	0.07	14	87	0.03	4.252×10^{-7}	9.633×10^{-7}
HS32	3	2	4	7	0.1	1.000	0.000	10	17	0.04	2	15	0.01	1.000	0.000
HS34	3	2	5	8	0.11	-7.820×10^{-1}	1.066×10^{-14}	31	69	0.08	4	29	0.01	-8.266×10^{-1}	0.000
HS40	4	3	1	5	0.03	-2.454×10^{-1}	2.554×10^{-15}	5	19	0.01	4	33	0.01	-2.500×10^{-1}	5.162×10^{-8}
HS44	4	6	3	8	0.1	-1.000(*)	0.000	12	24	0.07	4	30	0.01	-1.500×10^{1}	0.000
HS47	5	3	6	11	0.15	5.639×10^{-2}	1.466×10^{-12}	46	119	0.12	17	134	0.04	4.638×10^{-8}	3.167×10^{-9}
HS50	5	3	19	24	1.26	1.056	2.398×10^{-14}	126	423	1.19	9	71	0.03	4.941×10^{-2}	4.441×10^{-16}
HS64	3	1	1	4	0.09	6.813×10^{3}	4.742×10^{-11}	14	84	0.05	5	30	0.02	6.705×10^{3}	3.942×10^{-5}
HS66	3	2	1	4	0.05	5.800×10^{-1}	3.563×10^{-12}	6	11	0.01	0	3	0.0	5.800×10^{-1}	0.000
HS67	3	14	31	34	1.92	-1.053×10^{3}	0.000	402	1320	1.82	8	45	0.02	-1.157×10^{3}	0.000
HS72	4	2	1	5	0.03	7.407×10^{2}	0.000	2	17	0.01	11	72	0.04	7.277×10^{2}	8.766×10^{-7}
HS75	4	5	1	5	0.06	5.350×10^{3}	1.259×10^{-8}	4	13	0.01	5	36	0.03	5.174×10^{3}	1.880×10^{-4}
HS85	5	21	29	34	1.23	-2.061	1.421×10^{-14}	348	881	1.04	10	78	0.03	-2.216	2.389×10^{-8}
HS87	6	4	1	7	0.1	8.997×10^{3}	2.274×10^{-13}	9	27	0.06	3	33	0.02	8.997×10^{3}	3.775×10^{-8}
HS88	2	1	1	3	0.13	1.385	2.158×10^{-14}	19	52	0.07	16	94	0.05	1.363	1.074×10^{-8}
HS89	3	1	2	5	0.41	1.365	1.123×10^{-16}	57	349	0.28	23	147	0.07	1.362	7.400×10^{-7}
HS90	4	1	2	6	0.56	1.446	6.661×10^{-16}	86	464	0.52	36	334	0.15	1.363	1.142×10^{-7}
HS93	6	2	1	7	0.13	1.371×10^{2}	2.946×10^{-11}	10	25	0.11	0	5	0.0	1.371×10^{2}	0.000
HS98	6	4	2	8	0.06	3.136	0.000	4	11	0.02	4	40	0.01	3.214	0.000
HS100	7	4	1	8	0.14	7.140×10^{2}	0.000	22	80	0.12	0	5	0.0	7.140×10^{2}	0.000
HS101	7	5	26	33	5.28	1.982×10^{3}	3.053×10^{-16}	740	4391	5.04	26	269	0.1	1.810×10^{3}	2.249×10^{-5}
HS102	7	5	34	41	4.59	9.948×10^{2}	2.776×10^{-17}	641	3328	4.29	25	272	0.09	9.129×10^{2}	5.866×10^{-5}
HS103	7	5	15	22	1.72	5.623×10^{2}	8.913×10^{-12}	276	1270	1.63	20	212	0.07	5.434×10^{2}	3.104×10^{-4}
HS104	8	5	1	9	0.15	4.200	1.459×10^{-11}	14	47	0.11	11	130	0.03	3.951	5.769×10^{-7}
LOADBAL	31	31	1	15499	0.99	1.547(***)	1.798×10^{308}	0	2	0.0	6	231	0.04	5.252×10^{-1}	1.710×10^{-14}
OPTPRLOC	30	30	5	35	0.48	-1.567×10^{1}	2.220×10^{-16}	44	107	0.4	12	456	0.06	-1.642×10^{1}	1.279×10^{-13}
CB3	3	3	1	4	0.02	2.000	0.000	2	9	0.01	5	30	0.01	2.000	3.017×10^{-6}
CRESC50	6	100	16	22	2.49	6.057×10^{-1}	9.032×10^{-11}	344	1534	2.38	10	91	0.06	6.567×10^{-1}	9.586×10^{-4}
DEMBO7	16	20	13	29	1.44	1.879×10^{2}	5.917×10^{-8}	198	512	1.19	16	358	0.12	1.748×10^{2}	1.074×10^{-5}
DNIEPER	57	24	4	61	1.86	1.941×10^{4}	4.264×10^{-11}	96	460	1.35	5	359	0.07	1.874×10^{4}	3.740×10^{-10}
EXPFITA	5	22	1	10999	0.62	2.999×10^{1}(***)	1.798×10^{308}	0	2	0.0	7	56	0.02	9.889×10^{-2}	1.354×10^{-14}
HIMMELBI	100	12	114	215	70.51	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12588	68.05	3	408	0.05	-1.625×10^{3}	0.000
SYNTHES1	6	6	1	2999	0.23	1.000×10^{1}(***)	1.798×10^{308}	0	2	0.0	7	65	0.02	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	2	4	0.07	1.523	7.885×10^{-9}	13	43	0.04	4	26	0.01	1.558	0.000
DIPIGRI	7	4	1	8	0.15	7.140×10^{2}	0.000	22	80	0.13	0	5	0.0	7.140×10^{2}	0.000

Table C.7. Noiseless Problems with Infeasible x_0. $\tau = 10^{-1}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	2	4	0.19	1.000	1.310×10^{-14}	75	106	0.17	18	90	0.08	1.001	0.000
HS22	2	2	1	3	0.04	1.000	1.789×10^{-12}	9	25	0.02	3	16	0.01	1.000	1.231×10^{-6}
HS23	2	5	3	5	0.05	2.000	0.000	11	16	0.03	4	20	0.01	2.001	0.000
HS26	3	1	36	39	1.86	1.737×10^{-4}	1.040×10^{-12}	754	4339	1.77	14	87	0.03	4.252×10^{-7}	9.633×10^{-7}
HS32	3	2	4	7	0.06	1.000	0.000	10	17	0.03	2	15	0.01	1.000	0.000
HS34	3	2	6	9	0.11	-8.340×10^{-1}	1.776×10^{-15}	32	71	0.08	6	39	0.01	-8.340×10^{-1}	5.446×10^{-9}
HS40	4	3	2	6	0.05	-2.491×10^{-1}	2.189×10^{-9}	8	25	0.02	4	33	0.01	-2.500×10^{-1}	5.162×10^{-8}
HS44	4	6	3	8	0.05	-1.000(*)	0.000	12	24	0.03	4	30	0.01	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.41	1.989×10^{-7}	3.781×10^{-11}	1391	7878	3.33	72	696	0.36	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.02	1.056	2.398×10^{-14}	126	423	0.96	11	85	0.03	1.339×10^{-6}	8.882×10^{-16}
HS64	3	1	13	16	0.5	6.303×10^{3}	1.443×10^{-15}	126	411	0.44	9	54	0.02	6.306×10^{3}	0.000
HS66	3	2	2	5	0.03	5.182×10^{-1}	3.563×10^{-12}	7	13	0.02	3	21	0.01	5.186×10^{-1}	0.000
HS67	3	14	38	41	1.51	-1.162×10^{3}	0.000	448	1378	1.42	9	50	0.02	-1.161×10^{3}	0.000
HS72	4	2	5	9	0.2	7.284×10^{2}	0.000	38	102	0.16	11	72	0.05	7.277×10^{2}	8.766×10^{-7}
HS75	4	5	16	20	0.35	5.174×10^{3}	5.684×10^{-14}	97	185	0.28	5	36	0.03	5.174×10^{3}	1.880×10^{-4}
HS85	5	21	31	36	0.96	-2.216	0.000	357	904	0.87	10	78	0.03	-2.216	2.389×10^{-8}
HS87	6	4	1	7	0.09	8.997×10^{3}	2.274×10^{-13}	9	27	0.06	3	33	0.01	8.997×10^{3}	3.775×10^{-8}
HS88	2	1	2	4	0.23	1.363	9.216×10^{-16}	41	150	0.19	16	94	0.05	1.363	1.074×10^{-8}
HS89	3	1	4	7	0.6	1.363	3.663×10^{-16}	101	518	0.48	23	147	0.07	1.362	7.400×10^{-7}
HS90	4	1	6	10	1.57	1.363	5.260×10^{-17}	277	1551	1.52	36	334	0.16	1.363	1.142×10^{-7}
HS93	6	2	21	27	1.51	1.352×10^{2}	4.695×10^{-11}	333	1707	1.43	8	87	0.03	1.351×10^{2}	0.000
HS98	6	4	2	8	0.06	3.136	0.000	4	11	0.01	6	56	0.01	3.136	0.000
HS100	7	4	51	58	15.19	6.813×10^{2}	1.388×10^{-17}	1609	11183	15.01	3	55	0.02	6.811×10^{2}	0.000
HS101	7	5	44	51	7.84	1.812×10^{3}	3.006×10^{-10}	1239	7252	7.65	26	269	0.1	1.810×10^{3}	2.249×10^{-5}
HS102	7	5	69	76	10.15	9.125×10^{2}	4.272×10^{-13}	1483	7671	9.89	26	282	0.09	9.120×10^{2}	3.705×10^{-6}
HS103	7	5	37	44	3.97	5.437×10^{2}	7.980×10^{-16}	671	3472	3.8	20	212	0.07	5.434×10^{2}	3.104×10^{-4}
HS104	8	5	47	55	6.2	3.952	5.218×10^{-15}	1210	6053	5.99	11	130	0.03	3.951	5.769×10^{-7}
LOADBAL	31	31	1	15499	0.92	1.547(***)	1.798×10^{308}	0	2	0.0	9	330	0.05	4.531×10^{-1}	1.421×10^{-14}
OPTPRLOC	30	30	6	36	0.45	-1.642×10^{1}	0.000	47	114	0.39	12	456	0.06	-1.642×10^{1}	1.279×10^{-13}
CB3	3	3	1	4	0.03	2.000	0.000	2	9	0.01	5	30	0.01	2.000	3.017×10^{-6}
CRESC50	6	100	68	74	18.03	5.942×10^{-1}	0.000	2416	11263	17.76	456	6023	2.47	5.948×10^{-1}	4.547×10^{-13}
DEMBO7	16	20	18	34	2.17	1.749×10^{2}	1.067×10^{-10}	301	798	2.03	16	358	0.11	1.748×10^{2}	1.074×10^{-5}
DNIEPER	57	24	5	62	1.45	1.874×10^{4}	1.405×10^{-9}	100	468	1.35	5	359	0.07	1.874×10^{4}	3.740×10^{-10}
EXPFITA	5	22	1	10999	0.59	2.999×10^{1}(***)	1.798×10^{308}	0	2	0.0	12	91	0.03	1.355×10^{-3}	6.750×10^{-14}
HIMMELBI	100	12	114	215	68.61	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12588	67.1	16	1734	0.19	-1.734×10^{3}	3.553×10^{-15}
SYNTHES1	6	6	1	2999	0.22	1.000×10^{1}(***)	1.798×10^{308}	0	2	0.0	7	65	0.02	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	3	5	0.08	1.509	4.742×10^{-11}	16	60	0.05	9	49	0.02	1.509	8.844×10^{-8}
DIPIGRI	7	4	51	58	11.34	6.812×10^{2}	6.939×10^{-18}	1491	9510	11.16	3	55	0.01	6.811×10^{2}	0.000

Table C.8. Noiseless Problems with Infeasible x_0. $\tau = 10^{-3}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	3	5	0.29	1.000	1.309×10^{-14}	136	191	0.28	19	102	0.09	9.983×10^{-1}	6.579×10^{-10}
HS22	2	2	1	3	0.04	1.000	1.789×10^{-12}	9	25	0.02	3	16	0.01	1.000	1.231×10^{-6}
HS23	2	5	3	5	0.04	2.000	0.000	11	16	0.02	5	24	0.01	2.000	0.000
HS26	3	1	38	41	8.15	8.872×10^{-7}	4.441×10^{-16}	3356	26977	8.07	14	87	0.03	4.252×10^{-7}	9.633×10^{-7}
HS32	3	2	4	7	0.05	1.000	0.000	10	17	0.03	2	15	0.01	1.000	0.000
HS34	3	2	6	9	0.11	-8.340×10^{-1}	1.776×10^{-15}	32	71	0.08	6	39	0.01	-8.340×10^{-1}	5.446×10^{-9}
HS40	4	3	4	8	0.08	-2.500×10^{-1}	1.058×10^{-12}	13	36	0.04	4	33	0.01	-2.500×10^{-1}	5.162×10^{-8}
HS44	4	6	3	8	0.05	-1.000(*)	0.000	12	24	0.04	4	30	0.01	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.42	1.989×10^{-7}	3.781×10^{-11}	1391	7878	3.35	72	696	0.36	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.11	1.056	2.398×10^{-14}	126	423	1.06	11	85	0.03	1.339×10^{-6}	8.882×10^{-16}
HS64	3	1	17	20	0.58	6.300×10^{3}	0.000	156	447	0.52	11	64	0.03	6.300×10^{3}	6.916×10^{-6}
HS66	3	2	2	5	0.03	5.182×10^{-1}	3.563×10^{-12}	7	13	0.02	4	27	0.01	5.182×10^{-1}	0.000
HS67	3	14	39	42	1.46	-1.162×10^{3}	0.000	450	1381	1.38	10	55	0.03	-1.162×10^{3}	0.000
HS72	4	2	6	10	0.27	7.277×10^{2}	6.263×10^{-14}	43	109	0.22	11	72	0.04	7.277×10^{2}	8.766×10^{-7}
HS75	4	5	16	20	0.34	5.174×10^{3}	5.684×10^{-14}	97	185	0.27	5	36	0.03	5.174×10^{3}	1.880×10^{-4}
HS85	5	21	31	36	0.94	-2.216	0.000	357	904	0.83	10	78	0.03	-2.216	2.389×10^{-8}
HS87	6	4	2	8	0.13	8.997×10^{3}	7.844×10^{-12}	15	46	0.1	3	33	0.02	8.997×10^{3}	3.775×10^{-8}
HS88	2	1	4	6	0.34	1.363	1.301×10^{-14}	69	229	0.3	16	94	0.05	1.363	1.074×10^{-8}
HS89	3	1	5	8	0.64	1.363	0.000	113	576	0.54	23	147	0.08	1.362	7.400×10^{-7}
HS90	4	1	11	15	3.63	1.363	8.118×10^{-18}	718	3460	3.57	36	334	0.18	1.363	1.142×10^{-7}
HS93	6	2	25	31	1.58	1.351×10^{2}	3.331×10^{-16}	371	1843	1.5	8	87	0.03	1.351×10^{2}	0.000
HS98	6	4	2	8	0.05	3.136	0.000	4	11	0.02	6	56	0.01	3.136	0.000
HS100	7	4	89	96	24.87	6.806×10^{2}	0.000	2860	19400	24.54	14	212	0.07	6.806×10^{2}	0.000
HS101	7	5	71	78	12.77	1.810×10^{3}	2.776×10^{-16}	2004	11927	12.48	27	280	0.1	1.810×10^{3}	2.398×10^{-5}
HS102	7	5	78	85	10.48	9.119×10^{2}	0.000	1601	8233	10.11	28	305	0.09	9.119×10^{2}	2.176×10^{-6}
HS103	7	5	39	46	4.26	5.437×10^{2}	6.106×10^{-16}	685	3522	4.09	20	212	0.07	5.434×10^{2}	3.104×10^{-4}
HS104	8	5	80	88	13.33	3.951	0.000	2262	12387	13.0	11	130	0.03	3.951	5.769×10^{-7}
LOADBAL	31	31	1	15499	0.89	1.547(***)	1.798×10^{308}	0	2	0.0	17	594	0.09	4.529×10^{-1}	1.483×10^{-14}
OPTPRLOC	30	30	8	38	0.54	-1.642×10^{1}	4.504×10^{-10}	53	128	0.44	12	456	0.06	-1.642×10^{1}	1.279×10^{-13}
CB3	3	3	1	4	0.03	2.000	0.000	2	9	0.01	5	30	0.01	2.000	3.017×10^{-6}
CRESC50	6	100	1000	1006	214.67	5.939×10^{-1}(**)	5.482×10^{-13}	26160	96697	211.17	456	6023	2.37	5.948×10^{-1}	4.547×10^{-13}
DEMBO7	16	20	19	35	2.11	1.748×10^{2}	4.073×10^{-15}	302	800	1.97	16	358	0.1	1.748×10^{2}	1.074×10^{-5}
DNIEPER	57	24	5	62	1.43	1.874×10^{4}	1.405×10^{-9}	100	468	1.36	5	359	0.07	1.874×10^{4}	3.740×10^{-10}
EXPFITA	5	22	1	10999	0.63	2.999×10^{1}(***)	1.798×10^{308}	0	2	0.0	13	98	0.03	1.137×10^{-3}	6.047×10^{-12}
HIMMELBI	100	12	114	215	68.54	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12588	67.29	41	4291	0.45	-1.736×10^{3}	1.421×10^{-14}
SYNTHES1	6	6	1	2999	0.21	1.000×10^{1}(***)	1.798×10^{308}	0	2	0.0	7	65	0.01	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	3	5	0.08	1.509	4.742×10^{-11}	16	60	0.05	9	49	0.02	1.509	8.844×10^{-8}
DIPIGRI	7	4	87	94	78.71	6.806×10^{2}	1.265×10^{-13}	7019	23031	78.4	14	212	0.06	6.806×10^{2}	0.000

Table C.9. Noiseless Problems with Infeasible x_0 $\tau = 10^{-5}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

			FIBO								knitro				
Problem	n	m	#iter	#feval	time	f	feas err	#iter(sub)	#ceval	time(sub)	#iter	#feval	time	f	feas err
HS13	2	1	3	5	0.29	1.000	1.309×10^{-14}	136	191	0.28	19	102	0.09	9.983×10^{-1}	6.579×10^{-10}
HS22	2	2	1	3	0.04	1.000	1.789×10^{-12}	9	25	0.02	3	16	0.01	1.000	1.231×10^{-6}
HS23	2	5	3	5	0.05	2.000	0.000	11	16	0.03	6	28	0.01	2.000	0.000
HS26	3	1	43	46	12.41	2.878×10^{-8}	1.110×10^{-16}	5194	38851	12.32	28	203	0.08	2.719×10^{-11}	8.518×10^{-7}
HS32	3	2	4	7	0.05	1.000	0.000	10	17	0.03	2	15	0.01	1.000	0.000
HS34	3	2	6	9	0.09	-8.340×10^{-1}	1.776×10^{-15}	32	71	0.07	6	39	0.01	-8.340×10^{-1}	5.446×10^{-9}
HS40	4	3	4	8	0.07	-2.500×10^{-1}	1.058×10^{-12}	13	36	0.04	4	33	0.01	-2.500×10^{-1}	5.162×10^{-8}
HS44	4	6	3	8	0.05	-1.000(*)	0.000	12	24	0.03	4	30	0.01	-1.500×10^{1}	0.000
HS47	5	3	23	28	3.44	1.989×10^{-7}	3.781×10^{-11}	1391	7878	3.36	72	696	0.33	1.007×10^{1}	6.661×10^{-16}
HS50	5	3	19	24	1.12	1.056	2.398×10^{-14}	126	423	1.06	13	100	0.03	2.358×10^{-8}	4.441×10^{-16}
HS64	3	1	17	20	0.59	6.300×10^{3}	0.000	156	447	0.52	11	64	0.03	6.300×10^{3}	6.916×10^{-6}
HS66	3	2	2	5	0.03	5.182×10^{-1}	3.563×10^{-12}	7	13	0.02	5	33	0.01	5.182×10^{-1}	5.154×10^{-11}
HS67	3	14	40	43	1.46	-1.162×10^{3}	2.274×10^{-12}	458	1390	1.36	11	60	0.03	-1.162×10^{3}	1.364×10^{-12}
HS72	4	2	7	11	0.25	7.277×10^{2}	2.819×10^{-18}	44	111	0.2	11	72	0.04	7.277×10^{2}	8.766×10^{-7}
HS75	4	5	16	20	0.35	5.174×10^{3}	5.684×10^{-14}	97	185	0.28	5	36	0.03	5.174×10^{3}	1.880×10^{-4}
HS85	5	21	31	36	1.42	-2.216	0.000	357	904	1.3	10	78	0.03	-2.216	2.389×10^{-8}
HS87	6	4	9	15	0.7	8.997×10^{3}	2.274×10^{-13}	80	287	0.67	6	75	0.02	8.997×10^{3}	7.329×10^{-11}
HS88	2	1	4	6	0.34	1.363	1.301×10^{-14}	69	229	0.3	16	94	0.05	1.363	1.074×10^{-8}
HS89	3	1	6	9	0.78	1.363	5.535×10^{-16}	142	676	0.66	23	147	0.07	1.362	7.400×10^{-7}
HS90	4	1	11	15	3.63	1.363	8.118×10^{-18}	718	3460	3.57	36	334	0.18	1.363	1.142×10^{-7}
HS93	6	2	36	42	2.19	1.351×10^{2}	0.000	483	2292	2.06	9	96	0.03	1.351×10^{2}	0.000
HS98	6	4	2	8	0.05	3.136	0.000	4	11	0.02	6	56	0.01	3.136	0.000
HS100	7	4	106	113	28.46	6.806×10^{2}	0.000	3280	22448	28.08	28	362	0.11	6.806×10^{2}	5.093×10^{-11}
HS101	7	5	90	97	15.51	1.810×10^{3}	2.742×10^{-12}	2377	14126	15.15	28	289	0.11	1.810×10^{3}	1.302×10^{-5}
HS102	7	5	89	96	12.12	9.119×10^{2}	2.012×10^{-16}	1847	9687	11.74	29	314	0.1	9.119×10^{2}	7.385×10^{-7}
HS103	7	5	42	49	4.11	5.437×10^{2}	4.302×10^{-16}	713	3625	3.92	20	212	0.07	5.434×10^{2}	3.104×10^{-4}
HS104	8	5	88	96	13.73	3.951	4.441×10^{-16}	2400	13131	13.38	11	130	0.03	3.951	5.769×10^{-7}
LOADBAL	31	31	1	15499	0.9	1.547(***)	1.798×10^{308}	0	2	0.0	18	627	0.09	4.529×10^{-1}	8.549×10^{-15}
OPTPRLOC	30	30	8	38	0.53	-1.642×10^{1}	4.504×10^{-10}	53	128	0.43	12	456	0.06	-1.642×10^{1}	1.279×10^{-13}
CB3	3	3	1	4	0.02	2.000	0.000	2	9	0.0	5	30	0.01	2.000	3.017×10^{-6}
CRESC50	6	100	1000	1006	214.2	5.939×10^{-1}(**)	5.482×10^{-13}	26160	96697	210.8	456	6023	2.34	5.948×10^{-1}	4.547×10^{-13}
DEMBO7	16	20	19	35	2.11	1.748×10^{2}	4.073×10^{-15}	302	795	2.04	16	358	0.11	1.748×10^{2}	1.074×10^{-5}
DNIEPER	57	24	23	80	3.47	1.874×10^{4}	5.684×10^{-14}	272	1091	3.27	5	359	0.06	1.874×10^{4}	3.740×10^{-10}
EXPFITA	5	22	1	10999	0.6	2.999×10^{1}(***)	1.798×10^{308}	0	2	0.0	14	105	0.04	1.137×10^{-3}	1.776×10^{-14}
HIMMELBI	100	12	114	215	69.13	-1.467×10^{3}(*)	1.789×10^{-9}	2618	12588	68.01	46	4805	0.48	-1.736×10^{3}	2.132×10^{-14}
SYNTHES1	6	6	1	2999	0.24	1.000×10^{1}(***)	1.798×10^{308}	0	2	0.0	7	65	0.02	7.593×10^{-1}	3.261×10^{-7}
TWOBARS	2	2	5	7	0.11	1.509	0.000	21	89	0.08	9	49	0.02	1.509	8.844×10^{-8}
DIPIGRI	7	4	96	103	80.89	6.806×10^{2}	4.547×10^{-13}	7235	24606	80.56	28	365	0.11	6.806×10^{2}	7.237×10^{-11}

Table C.10. Noiseless Problems with Infeasible x_0 $\tau = 10^{-7}$; n: number of variables, m: number of constraints, #iter: number of (outer) iterations, #feval: number of function evaluations, time: total CPU time passed, f: final objective value, feas err: final feasibility error, #iter(sub): total number of iterations for solving TR subproblem, #ceval: total number of constraint evaluations(note that the number of constraint evaluations and function evaluations are same for `KNITRO`), time(sub): total CPU time elapsed for solving subproblem. * indicates that `FIBO` terminates with singular interpolation system error, ** indicates that `FIBO` terminates with maximum number of iterations, *** indicates that `FIBO` terminates with maximum number of function evaluations.

APPENDIX D

Analyzing the Performance of DFO Methods on a Wider Class of Problems

D.1. Proof for Theorem 6.3.1

Proof. By definition of the quadratic model (4.2.1), we have that

$$\begin{aligned} f(y^i) - m(x) &= m(y^i) - m(x) \\ &= (y^i - x)^T(g + Hx) + \frac{1}{2}(y^i - x)^T H(y^i - x) \\ &= (y^i - x)^T \nabla m(x) + \frac{1}{2}(y^i - x)^T H(y^i - x) \end{aligned}$$

Therefore, using Taylor expansion, it holds that

(D.1.1)

$$(y^i - x)^T(\nabla m(x) - \nabla \phi(x))$$

(D.1.2)

$$= \phi(y^i) - (y^i - x)^T \nabla \phi(x) - \phi(x) - \frac{1}{2}(y^i - x)^T H(y^i - x) + \phi(x) - m(x) + \epsilon(y^i)$$

(D.1.3)

$$= \frac{1}{2}(y^i - x)^T \nabla^2 f(x + t_i(y^i - x))(y^i - x) - \frac{1}{2}(y^i - x)^T H(y^i - x) + \phi(x) - m(x) + \epsilon(y^i)$$

for some $t_i \in (0, 1)$

Subtracting the (D.1.3) associated with y^0 from $i = 1, ..., p$,

$$
\begin{aligned}
&(y^i - y^0)(\nabla m(x) - \nabla \phi(x)) \\
&= \frac{1}{2}(y^i - x)^T \nabla^2 f(x + t_i(y^i - x))(y^i - x) - \frac{1}{2}(y^i - x)^T H(y^i - x) \\
&- \frac{1}{2}(y^0 - x)^T \nabla^2 f(x + t_i(y^0 - x))(y^0 - x) + \frac{1}{2}(y^0 - x)^T H(y^0 - x) + \epsilon(y^i) - \epsilon(y^0) \\
&\leq \frac{5}{2}(\|H\| + L_2)\Delta^2 + 2\epsilon_f
\end{aligned}
$$

where the last inequality is derived using $\|y^i - x\| \leq 2\Delta$ and $\|y^0 - x\| \leq \Delta$ and the bounded noise assumption.

Combining p equations, we have that

$$\|M(\nabla m(x) - \nabla \phi(x))\| \leq \frac{5\sqrt{p}}{2}(\|H\| + L_2)\Delta^2 + 2\sqrt{p}\epsilon_f$$

where

$$M = \begin{bmatrix} y^1 - y^0 & y^2 - y^0 & ... & y^p - y^0 \end{bmatrix}$$

Defining the scaled matrix $\hat{M} = M/\Delta$, we have that

$$\|\nabla m(x) - \nabla \phi(x)\| \leq \frac{5\sqrt{p}\|\hat{M}^\dagger\|}{2}(\|H\| + L_2)\Delta + \frac{2\sqrt{p}\|\hat{M}^\dagger\|\epsilon_f}{\Delta}$$

In addition, substituting $x = y^0$ in (D.1.3), we have that

$$\begin{aligned}(y^i - y^0)^T(\nabla m(x) - \nabla\phi(x)) &\leq \frac{1}{2}(L_2 + \|H\|)\Delta^2 + \phi(y^0) - m(y^0) + \epsilon(y^i) \\ \implies m(y^0) - \phi(y^0) &\leq \frac{1}{2}(L_2 + \|H\|)\Delta^2 + \|\nabla m(x) - \nabla\phi(x)\|\Delta + \epsilon_f \\ &\leq \frac{5\sqrt{p}\|\hat{M}^\dagger\| + 1}{2}(\|H\| + L_2)\Delta^2 + (2\sqrt{p}\|\hat{M}^\dagger\| + 1)\epsilon_f\end{aligned}$$

□

D.2. Numerical Results for Comparing DFO-TR against CMA-ES

An alternative IBO solver is DFO-TR, which has higher linear algebra cost than NEWUOA at every iteration. Therefore, we consider problems with less than 50 variables and validate our findings with respect to NEWUOA. We set final trust region radius to 10^{-8} and the stop_predict convergence test to 0 to avoid early termination. As demonstrated in Figure D.1 - D.2, similar conclusions can be made regarding the relative performance of DFOTR and CMA-ES. However, we note in Figure D.3 that relaxing the ratio test improves the final accuracy of DFO-TR, especially when the noise level is large.

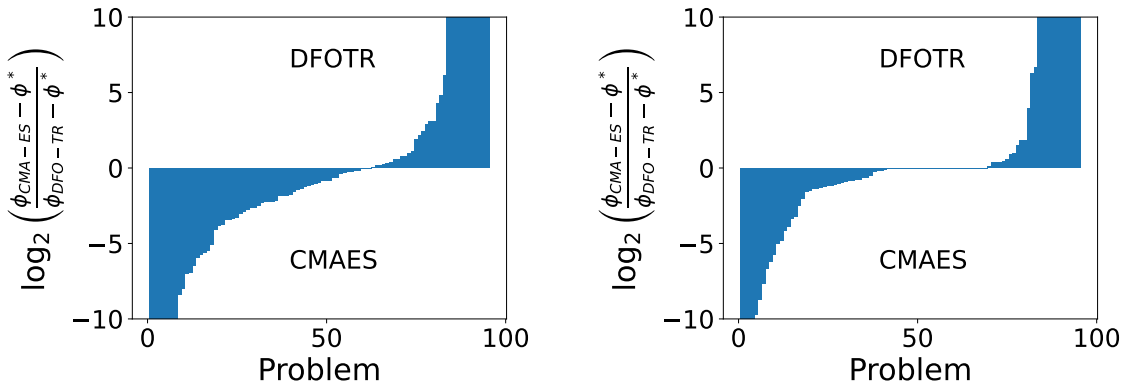

Figure D.1. *(Selected) Accuracy Log-Ratio Profiles for Noisy Problems with noise level* 0.001 *(Left) and* 10^{-7} *(Right).* Plots of (6.2.1) comparing `CMA-ES` and `DFO-TR` with restarts within a budget of $500n$.

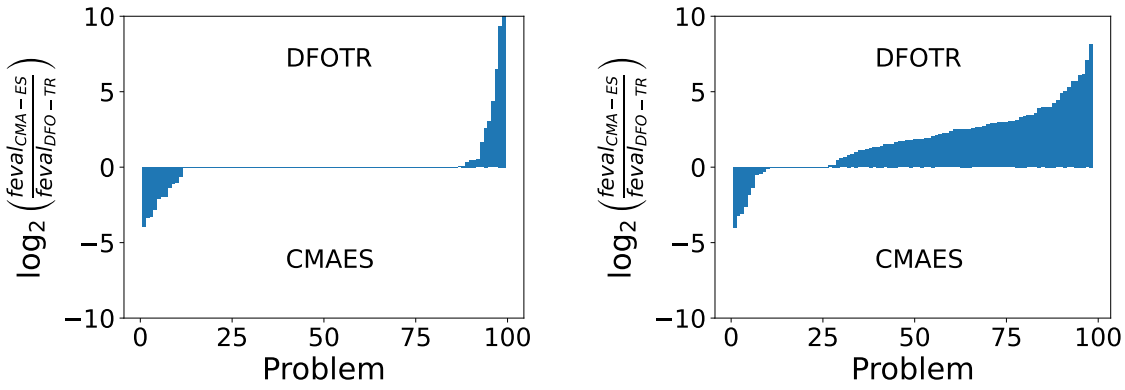

Figure D.2. *(Selected) Efficiency Log-Ratio Profiles for Noisy Problems with noise level* 0.001 *(Left) and* 10^{-7} *(Right).* Plots of (6.2.3) comparing `CMA-ES` and `DFO-TR` with restarts for $\tau = 10^{-8}$.

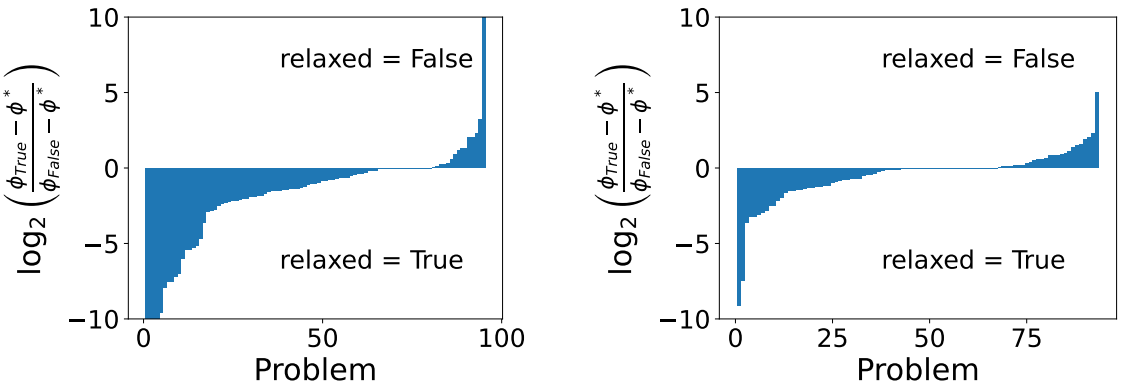

Figure D.3. *(Selected) Accuracy Log-Ratio Profiles for Noisy Problems with noise level* 0.001 *(Left) and* 10^{-7} *(Right).* Plots of (6.2.1) comparing `DFO-TR` with and without ratio relaxation using a budget of $500n$.