Derivative-Free Algorithms for Nonlinear Optimisation Problems

Lindon Roberts
Mansfield College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2019

Acknowledgements

This research was supported by the EPSRC Centre For Doctoral Training in Industrially Focused Mathematical Modelling (EP/L015803/1) in collaboration with the Numerical Algorithms Group. This support would not have been possible without the work of Chris Breward and Colin Please. I also acknowledge the honorary support of the James Fairfax – Oxford Australia Scholarship Fund, and the use of the University of Oxford Advanced Research Computing facility in carrying out this work.[1]

My largest thanks must go to my supervisor, Coralia Cartis, for her enthusiasm, tireless guidance and support at all stages of this project, and for ensuring that I don't leave interesting questions unexplored. I would also like to thank my industrial supervisors Jan Fiala and Benjamin Marteau for their keen interest and advice, and their efforts in commercialising my research.

I would also like to thank the following people for their invaluable advice and discussions on this research: Michael Ferris, Jaroslav Fowkes, Nicholas Gould, Raphael Hauser, Katya Scheinberg, Oliver Sheridan-Methven, Jared Tanner, Simon Tett, Andrew Wathen, and Amy Willis.

Finally, my thanks go to my family and friends, and in particular Erin Stewart, for their ongoing encouragement and assistance.

[1] `http://dx.doi.org/10.5281/zenodo.22558`

Abstract

Minimising a nonlinear function is a ubiquitous problem in applications, and standard algorithms need accurate evaluations of the function and its derivatives. However, if the function is black-box, expensive to evaluate, or noisy, it may be impractical or even impossible to obtain accurate derivatives, and we require derivative-free optimisation (DFO). Model-based DFO methods perform well in practice, taking many features from derivative-based methods, but can have lower performance for noisy problems and a high linear algebra cost. In this thesis, we aim to improve the flexibility, robustness and scalability of state-of-the-art methods for model-based DFO by designing, analysing, implementing and comprehensively testing three new algorithms for nonlinear optimisation, with a special focus on nonlinear least-squares problems (such as parameter fitting).

The first, DFO-LS, is designed for nonlinear least-squares problems, and is simpler than existing methods, constructing linear residual models with a flexible approach. DFO-LS can benefit from a reduced initialisation cost, and has improved scalability and runtime over existing methods without a performance penalty. DFO-LS also has features for noisy problems, including a novel multiple restarts mechanism, which are low cost in evaluations and linear algebra, giving DFO-LS better performance compared to existing techniques for handling noise.

We then introduce Py-BOBYQA, an extension of BOBYQA (Powell, 2009) with a similar multiple restarts mechanism to DFO-LS, amongst other new enhancements. It matches or outperforms BOBYQA and other state-of-the-art solvers on both smooth and noisy problems. We also propose a simple extension of Py-BOBYQA that may enable it to escape local minima and progress towards the global optima.

Lastly, we introduce, for large-scale nonlinear least-squares problems, DFBGN, the first model-based DFO method to optimise in low-dimensional subspaces at each iteration. DFBGN has lower linear algebra costs, improved runtime, and hence improved performance on large-scale problems than DFO-LS, as it can perform many more iterations within a reasonable runtime limit. It can also make progress under very small evaluation budgets, with a low runtime cost.

Contents

Chapter 1

Introduction and Literature Survey

This thesis concerns the optimisation of nonlinear, nonconvex functions. That is, it concerns algorithms that aim to find inputs minimising (or maximising) a function of interest called the 'objective' function. Problems of this type are ubiquitous in application areas across quantitative disciplines, and so the development of efficient and robust methods for such problems is an important topic of research. Many state-of-the-art numerical techniques for nonconvex optimisation rely on being able to calculate the value of the objective and its derivatives accurately; we call such methods *derivative-based*. In these methods, derivative information is used to accomplish critical tasks, such as calculating a potential search direction (i.e. a direction where we expect the objective value to increase/decrease), and determining when the algorithm should terminate (e.g. when the gradient is close to zero).

The need for derivative information in an optimisation method imposes the requirement that the user provide computer code that accurately evaluates the derivatives of the objective, in addition to code that evaluates the objective itself. However, there are realistic settings where this requirement cannot be met. For example, parameter estimation of climate models requires comparing results of multi-year climate simulations to observational data. The high computational cost of these simulations, coupled with the inaccuracies inherent in the results (due to the chaotic nature of the model), means that we have inaccurate objective values [211]. Hence for this problem it is impractical to calculate or estimate derivatives of the objective.

In settings where accurate derivatives of the objective are not available, we must turn to optimisation algorithms that only use function information. These are called *derivative-free* optimisation (DFO) methods, which have a long history, but have grown in popularity and sophistication over the last 20 years. They can be useful, for instance, for problems with noise in the objective, where, in general, derivative-based methods cannot be applied. In addition, DFO methods may also perform only few evaluations of the objective at each

iteration. Hence, if objective evaluations are the dominant cost of the optimisation, DFO methods may achieve a given accuracy faster than derivative-based methods.

To understand the remit of DFO methods, we must consider how we can meet the requirement to provide derivatives of the objective [164, Chapter 8]. The most straightforward approach is to calculate the derivatives of the objective by hand (or via a symbolic computation package), and to write code to evaluate these expressions directly. This requires that the analytic form of the objective is known to the user, and is simple enough for this process to be practical. Alternatively, derivatives may be approximated by finite differencing [38, Chapter 4], which requires the objective to be evaluated at several nearby points. Another popular alternative is to use algorithmic differentiation software [96], which inspects the code for evaluating the objective, and then constructs code for evaluating (analytic) derivatives.

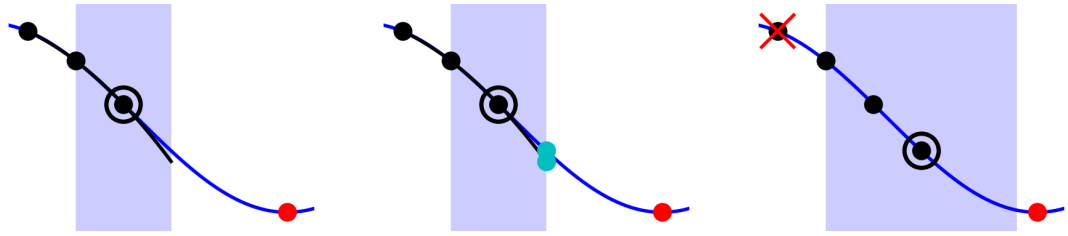
In situations where these approaches are inappropriate or impractical, DFO methods may be the only available approaches. If evaluating the objective relies on some black-box code (e.g. proprietary or legacy software), then it is likely that the analytic form of the objective is not known, and algorithmic differentiation packages may not be applicable. In this setting, finite differencing may be a good way to provide derivative information. However, this too is not universally applicable. For instance, if the objective is very expensive to compute, then the evaluations required to estimate a single gradient, usually on every iteration of the optimisation routine, may quickly exceed the user's computational resources.[1] Alternatively, if the evaluation of the objective is noisy, then finite differencing may give very inaccurate results [164, Lemma 9.1]. Both of these features are present in the parameter estimation problem for climate models mentioned above, for instance.

Given their suitability for expensive, black-box, and noisy objectives, DFO methods can be used to solve many diverse real-world problems. For example, DFO methods have been applied: to algorithm and parameter tuning [14, 60, 198], imaging [166, 204], tsunami detection [12], simulating gas explosions [32], and inverse optimal control for robotic motion [153]. More examples can be found in the textbooks [60, 12] and survey [7], for instance.

This research has been supported by the Numerical Algorithms Group (NAG), a British technology company that sells the NAG Library, a collection of mathematical software. The NAG Library includes many optimisation routines, which are an important part of its interest to customers. The project arose because NAG wished to improve their offering of DFO methods, in particular to include specialised methods for nonlinear least-squares problems.[2]

[1] The simplest finite differencing approach requires evaluating the objective at the point of interest, plus one extra evaluation per variable being optimised.

[2] Before this project, the NAG Library already had Powell's BOBYQA [186], described in Section 1.1.4. The software developed in this thesis takes its underlying algorithmic framework from BOBYQA for this reason.



(a) Build interpolation model *(b) Calculate candidate step* *(c) Update interpolation set*

Figure 1.1. An illustration of one iteration of model-based DFO for a one-dimensional problem. The goal is to find the minimum point (in red) of the objective (green); the current interpolation set are the black points and the large black point is the current iterate. The black curve is the local model, here a quadratic function. The light blue point is the candidate step, and it is added to the interpolation set, replacing the point furthest from the new iterate (red cross).

This is an important sub-class of optimisation problem, and DFO methods have been applied to real-world nonlinear least-squares problems such as: parameter estimation of climate models [211] and of nuclear fusion reactions [6], and calibrating energy density functionals for nuclear physics [225]. This problem structure is also found in the gas explosion [32] and robotic motion [153] examples noted above.

In this thesis, we will focus on both general objective and nonlinear least-squares problems. Our primary emphasis will be on finding local optima (i.e. points that are optimal in a neighbourhood, which may not be optimal over all points). Also, our work aims to develop DFO methods and software with a view to their effectiveness in many situations, and so we will not be concerned with applications of these methods to specific real-world problems.

There are many classes of DFO methods, including *direct search*, *model-based*, *Nelder-Mead*, *implicit filtering*, and more. In this thesis, we consider model-based methods, for several reasons: they try to incorporate the features of successful derivative-based methods as much as possible, such as using curvature of the objective [66]; they have been shown to perform well in practice compared to other classes of DFO methods (e.g. [75, 157, 200]); and they are suited to exploiting structure in the objective (e.g. [52, 83, 90]). We note that methods in one class can use techniques very similar to other classes, but we delineate them in a standard way (e.g. as in [129]).

Figure 1.1 illustrates the basic framework of model-based DFO methods on a one-dimensional minimisation problem. At each iteration we construct a local model, which we hope will approximate the objective well in a region near our current iterate. We do this via interpolation: we maintain a set of points (close to the current iterate) where we have evaluated the objective, and we require our model to match the objective value at these points. Given this model, we calculate a new candidate point by minimising the model in a region around the current iterate. We evaluate the objective at this new point, and if it

achieves sufficient reduction in the objective, it becomes the next iterate and is incorporated into the interpolation set (by removing an existing point); we also use this information to update the size of the region where we expect our model to be accurate. As the algorithm progresses, the interpolation set moves towards the optimal point.

In Section 1.1 below, we give an overview of the different classes of DFO methods and a detailed literature survey of model-based DFO. This chapter ends with an overview of the key research questions and contributions covered in this thesis, as well as a description of the structure of the main chapters, in Section 1.2.

1.1 Literature Survey

There are several classes of DFO methods; this thesis will focus on model-based methods, as described above. In this section we provide a brief review of the DFO literature: in Section 1.1.1 we give an overview of the main classes of DFO approaches, except for model-based methods. We then provide a detailed survey of general model-based DFO methods (Section 1.1.2), and classical and derivative-free methods for nonlinear least-squares problems (Section 1.1.3). We conclude with a review of existing model-based DFO software and approaches for comparing such software in Section 1.1.4. This review describes the techniques at a high level; we reserve a discussion of the key technical details for Chapter 2.

In this section, and throughout, we use $\|\cdot\|$ for the 2-norm of vectors and matrices (i.e. Euclidean norm and largest singular value, respectively) unless otherwise specified, and $\|\cdot\|_F$ for the Frobenius norm of matrices (i.e. $\|A\|_F^2 := \sum_{i,j} a_{ij}^2$). We also define, for $\mathbf{x} \in \mathbb{R}^n$ and $\Delta > 0$, the closed ball $B(\mathbf{x}, \Delta) := \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{y} - \mathbf{x}\| \leq \Delta\}$. Here and throughout, all vectors and matrices are only over $\mathbb{R}$.

1.1.1 Survey of DFO Classes

In this section we contextualise model-based DFO by briefly describing other DFO classes for local optimisation. We exclude model-based DFO here, as it is the focus of Section 1.1.2. Our discussion is based on methods for solving the general problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \tag{1.1}$$

and from a theoretical perspective we are interested in *global convergence* (i.e. convergence, usually to a stationary point, from any starting point). More details can be found in the references listed, as well as the textbooks [164, 60, 12] and survey articles [230, 56, 178, 121, 66, 129].

Direct Search Direct search methods are one of the most actively-studied classes of DFO algorithm, covering algorithms such as pattern search and linesearch methods. In essence, at each iteration of these methods, we evaluate the objective in a set of directions, and move to a new point that provides sufficient objective decrease, if one has been found.

The basic framework for pattern search methods is [121], at each iteration k:

1. *(Search step)* Select a finite set of test points (possibly empty), and if any of these points reduces the objective sufficiently, select it to be the next iterate. In this case, the iteration is complete, otherwise we continue.

2. *(Poll step)* Generate a finite set of 'poll directions' $\mathcal{D}_k$, and evaluate the objective at $\mathbf{x}_k + \alpha_k \mathbf{d}_k$ for $\mathbf{d}_k \in \mathcal{D}_k$, where $\mathbf{x}_k$ is the current iterate and $\alpha_k > 0$ is a step size. If one of these points reduced the objective by at least $\rho(\alpha_k) > 0$, then set this to be the new iterate; otherwise, reduce α_k by a constant factor.

There are three key ingredients that need to be specified, which we now describe.

First, we must specify how to build the poll directions $\mathcal{D}_k$. A typical requirement here is that at least one direction must be a descent direction of the objective f (whenever $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$), which guarantees that the poll step will succeed for sufficiently small α_k. In practice, this is achieved by ensuring that $\mathcal{D}_k$ is a *positive spanning set*; that is,

$$\mathrm{cm}(\mathcal{D}_k) := \min_{\mathbf{s} \neq \mathbf{0}} \max_{\mathbf{d} \in \mathcal{D}_k} \frac{\mathbf{s}^\top \mathbf{d}}{\|\mathbf{s}\| \|\mathbf{d}\|} > 0, \tag{1.2}$$

where $\mathrm{cm}(\mathcal{D}_k)$ is the *cosine measure* of $\mathcal{D}_k$. The value of $\mathrm{cm}(\mathcal{D}_k)$ is the cosine of the largest angle formed by any vector $\mathbf{s}$ and its 'nearest' vector in $\mathcal{D}_k$ (in the sense of forming the smallest angle with $\mathbf{s}$); condition (1.2) corresponds to there being at least one element of $\mathcal{D}_k$ that forms an acute angle with $\mathbf{s}$. For convergence, we generally require that $\beta_{\min} \leq \|\mathbf{d}\| \leq \beta_{\max}$ for all $\mathbf{d} \in \mathcal{D}_k$, and $\mathrm{cm}(\mathcal{D}_k) \geq \beta$, where the constants $\beta_{\min}$, $\beta_{\max}$ and β are strictly positive and independent of k. The construction of positive spanning sets is not difficult, for instance

$$\mathcal{D}_k = \{\pm \mathbf{e}_1, \ldots, \pm \mathbf{e}_n\} \qquad \text{or} \qquad \{\mathbf{e}_1, \ldots, \mathbf{e}_n, -\mathbf{e}/\sqrt{n}\}, \tag{1.3}$$

where $\mathbf{e}_i$ is the i-th coordinate vector and $\mathbf{e}$ is the vector of ones.

The polling step is typically done in one of two ways: *complete polling* checks all $\mathbf{d} \in \mathcal{D}_k$ and selects $\mathbf{d}_k$ as the one yielding the greatest decrease. This allows for stronger convergence results (specifically convergence of the full sequence $\|\nabla f(\mathbf{x}_k)\|$ to zero rather than a subsequence) [121], but requires the objective to be evaluated at all points. By contrast, *opportunistic polling* involves ordering $\mathcal{D}_k$ in some way, and choosing $\mathbf{d}_k$ to be the first point satisfying the sufficient decrease condition. In this case, the evaluation cost of the algorithm can be reduced by choosing a sensible ordering of $\mathcal{D}_k$ [10, 67].

Next, we need a sufficient decrease function $\rho : [0, \infty) \to [0, \infty)$. For theoretical guarantees of convergence, we usually require that $\rho(\alpha)$ is strictly increasing in α, and $\rho(\alpha) = o(\alpha)$ as $\alpha \to 0$; a common choice is $\rho(\alpha) = \alpha^2$ [60, Chapter 7.7]. A worst-case complexity analysis[3] for this algorithm was given in [221], providing bounds that match derivative-based algorithms (and model-based DFO methods; see Section 1.1.2). Globally-convergent methods can also be constructed where we accept a simple decrease in the poll step (i.e. $\rho(\alpha) \equiv 0$), however this imposes additional constraints on $\mathcal{D}_k$ (and the search step), for example, to ensure that all iterates lie on a rational lattice [9].

Lastly, the search step is a flexible, optional algorithm component, and allows the algorithm to be augmented with any strategy for finding new points that improve the objective value. The main requirement is that it tests a finite number of points at each iteration. Several approaches for the search step have been proposed, such as using interpolation models from previously-evaluated points [67, 65, 133, 54], the Nelder-Mead method (see below) [15], and techniques from global optimisation [220, 8].

Many extensions of pattern search algorithms are available. For instance, the Mesh-Adaptive Direct Search algorithm generates poll directions that are asymptotically dense in $\mathbb{R}^n$, and can be used for nonsmooth functions or constrained problems (by modifying the objective so that $f \equiv +\infty$ for infeasible points) [10]. Other modifications of pattern search exist for second-order convergence [1, 91] and multiobjective problems [64]. Alternatively, the evaluation cost of pattern search can be reduced by randomly selecting poll sets that may not be positive spanning sets (e.g. $\mathcal{D}_k = \{\pm \mathbf{d}\}$ for some random unit vector $\mathbf{d}$) [92, 94]. Software packages based on pattern search methods include NOMAD [132, 11], SID-PSM [68] and BFO [171].

Another major class of direct search methods are *linesearch DFO* methods. These are similar to pattern search (with a sufficient decrease condition), but where the size of α_k can vary within each iteration. They also have similarities to derivative-based linesearch methods [164, Chapter 3]. Here, at each iteration, we choose a poll set and evaluate the objective at $\mathbf{x}_k + \alpha \mathbf{d}_k$ for different poll directions $\mathbf{d}_k$ and step sizes $\alpha \geq \alpha_k^{\min}$, until we find a point that gives a sufficiently large improvement in the objective. If this process is unsuccessful, we reduce the lower bound $\alpha_k^{\min}$. A wide variety of methods in this class have been developed, for unconstrained [142], constrained nonsmooth [77] and mixed-integer nonlinear [138] problems, for instance. The software package DFL [137] implements linesearch DFO methods for a variety of problem types.

[3] This type of analysis provides an upper bound on the number of iterations/evaluations before an optimality measure, such as $\|\nabla f(\mathbf{x}_k)\|$, falls below ϵ for the first time as a function of ϵ.

A related class of algorithms is the classical *coordinate search* [231], where at each step we perform a linesearch in one coordinate direction, and change which coordinate is selected at each iteration. The simplest approach, in which the directions are chosen as a cycle (e.g. $\mathbf{e}_1, -\mathbf{e}_1 \ldots, \mathbf{e}_n, -\mathbf{e}_n, \mathbf{e}_1, \ldots$) does not converge, even with exact linesearch, as shown via a counterexample in [176]. However, some convergence results exist under stronger assumptions [231]. To speed up coordinate search methods, we can, after searching through all directions once, search along the line connecting the first iterate with the last (i.e. before and after the n coordinate searches). This process forms the basis of the Hooke and Jeeves method [209]. A more sophisticated version of this approach updates the set of search directions using these increments ($\mathbf{x}_n - \mathbf{x}_0$, etc.), and so builds a set of conjugate directions; the resulting method [172] is similar to the nonlinear conjugate gradient method for derivative-based optimisation [81, Chapter 4.2].

Nelder-Mead Method The Nelder-Mead or simplex method[4] [162] is possibly the most well-known derivative-free optimisation method. In this method, we maintain a set of $n+1$ points, and in each iteration we move the point with the worst objective value to a new location with a better value via one of several transformations. If this fails to find an improved point, then all points are moved closer to the current best point. The process of updating a set of points across iterations is similar to model-based DFO, but the mechanism for selecting new points is closer to pattern search.

Specifically, suppose we have points $\{\mathbf{x}_0, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^n$, ordered so that $f(\mathbf{x}_0) \leq f(\mathbf{x}_1) \leq \cdots \leq f(\mathbf{x}_n)$. Then we first try to just replace the worst point $\mathbf{x}_n$ with a point on the line between $\mathbf{x}_n$ and the centroid of the best n points $\overline{\mathbf{x}} := \sum_{i=0}^{n-1} \mathbf{x}_i/n$; if we define $\overline{\mathbf{x}}(t) := \overline{\mathbf{x}} + t(\mathbf{x}_n - \overline{\mathbf{x}})$, then we evaluate the objective at some combination of points $\overline{\mathbf{x}}(-1)$, $\overline{\mathbf{x}}(-2)$, $\overline{\mathbf{x}}(-1/2)$ and $\overline{\mathbf{x}}(1/2)$—called 'reflection', 'expansion', 'outside contraction' and 'inside contraction' respectively. If none of these points have an objective value less than $f(\mathbf{x}_n)$, we move all points towards $\mathbf{x}_0$ (i.e. replace $\mathbf{x}_i$ with $(\mathbf{x}_i + \mathbf{x}_0)/2$ for $i = 1, \ldots, n$)—we call this a 'shrink' step.

The first counterexample to the question of the convergence of the original Nelder-Mead method was the illustration of a two-dimensional nonconvex function where the simplex converges to its (nonstationary) starting point [229]. Later, a family of strictly convex two-dimensional functions where the method converges to a nonstationary point was constructed in [149]. However, it is known that the Nelder-Mead method converges for strictly convex functions with $n = 1$ [126], and for strictly convex, twice-differentiable functions with $n = 2$, provided no expansion steps are allowed [125]. If expansion steps are allowed, then

[4] This is not the same as the simplex method for linear programming [164, Chapter 13].

a weaker result holds for strictly convex functions with $n = 2$: the objective values at all vertices converge to the same value and the diameter of the simplex approaches zero [126]. Modifications of the original algorithm that have convergence guarantees have been proposed, e.g. [217, 116]. In particular, we note [116], which includes a sufficient decrease condition and a restart mechanism when an iteration gives insufficient (but still some) objective decrease.

Implicit Filtering Implicit filtering methods are essentially derivative-based linesearch methods, but with gradients estimated using finite differencing with an adaptively-chosen step size [117]. This approach can estimate the steepest descent direction, or more sophisticated approaches such as a quasi-Newton direction. A key issue is that this estimated gradient may not produce a descent direction, so the linesearch must terminate if the step size becomes sufficiently small (and the algorithm should instead reduce the finite differencing step size, similar to linesearch DFO methods). If no points evaluated during the finite differencing step reduce the objective, we can simply reduce the step size, avoiding the need to estimate the gradient and perform a linesearch [119]. In this case, the method resembles pattern search with $\mathcal{D}_k = \{\pm\mathbf{e}_1, \ldots, \pm\mathbf{e}_n\}$. The underlying theory for implicit filtering is related to the notion of *simplex gradients*, which is an approximation to a function gradient based on linear interpolation. This theory was analysed in detail in [193, 102], both of which consider simplex gradients from (possibly underdetermined) interpolation and linear regression, and the efficient calculation of simplex gradients was recently studied in [63]. The analysis of simplex gradients is relevant to the construction of linear models for model-based DFO (e.g. Section 2.2.1).

Implicit filtering is related to 'dynamic accuracy' (derivative-based) methods. In this setting, we assume that we can evaluate the objective and its gradient to within some error that we can control (e.g. given any $\epsilon > 0$, we can compute $\widetilde{f}(\mathbf{x}, \epsilon)$ with $|\widetilde{f}(\mathbf{x}, \epsilon) - f(\mathbf{x})| \leq \epsilon$). We then force these errors to approach zero over the course of the algorithm; the idea is that we only use high-accuracy evaluations towards the end of the algorithm, since they are likely more expensive to obtain. This could instead be achieved by finite differencing with a step size approaching zero.

Dynamic accuracy methods are studied in the context of trust-region methods in [53, Chapters 8 & 10], and have also been considered in the framework of adaptive regularisation with cubics (ARC) [46]. In ARC [42, 43], we calculate a candidate point $\mathbf{x}_k + \mathbf{s}_k$ by solving

$$\min_{\mathbf{s}\in\mathbb{R}^n} m_k(\mathbf{s}) := f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_k \mathbf{s} + \frac{\sigma_k}{3}\|\mathbf{s}\|^3, \tag{1.4}$$

where $H_k \approx \nabla^2 f(\mathbf{x}_k)$ is the Hessian of the objective, or an approximation to it (e.g. via quasi-Newton updating), and $\sigma_k > 0$ is a regularisation parameter. An advantage of ARC is

that it has better worst-case complexity than other classes of methods: that is, it requires at most $k = \mathcal{O}(\epsilon^{-3/2})$ iterations to achieve $\|\nabla f(\mathbf{x}_k)\| \leq \epsilon$ for the first time, compared to at most $\mathcal{O}(\epsilon^{-2})$ for trust-region methods [41].

One-Dimensional Methods There are several DFO methods for one-dimensional optimisation and root-finding problems. For instance, the golden section (optimisation) and bisection (root-finding) methods maintain an interval of decreasing size where a solution is guaranteed to exist, and the secant and regula falsi methods (both root-finding) are based on linear interpolation from already-evaluated points (see [189, Chapters 9–10] and [38, Chapter 2], for instance). Brent's methods[5] [34] combine these techniques to balance speed of convergence and robustness.

1.1.2 Model-Based DFO

Further to the brief description above (see Figure 1.1), this section contains a more detailed survey of model-based DFO methods. We focus largely on solving (1.1), i.e. unconstrained local minimisation without specific structure in the objective. In the next section, we will discuss how these methods have been specialised to nonlinear least-squares problems, a problem class that will form a large part of this thesis.

Derivative-Based Trust-Region Methods Most model-based DFO methods are variations of derivative-based trust-region methods [53]. In trust-region methods, at each iteration k we construct a model that we hope approximates the objective in a neighbourhood of the current iterate $\mathbf{x}_k \in \mathbb{R}^n$. We also maintain a parameter $\Delta_k > 0$ that is a measure of the size of this neighbourhood. In essence, we expect our model to be an accurate approximation for f in $B(\mathbf{x}_k, \Delta_k)$.

The basic framework for derivative-based trust-region methods for solving (1.1) is:

1. Construct a local model $m_k(\mathbf{s}) \approx f(\mathbf{x}_k + \mathbf{s})$; this model is usually quadratic, such as the second-order Taylor series

$$m_k(\mathbf{s}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_k \mathbf{s}, \tag{1.5}$$

   where $H_k \approx \nabla^2 f(\mathbf{x}_k)$ is the Hessian of the objective, or an approximation to it (e.g. via quasi-Newton updating).

[5] Methods by this name exist for both optimisation and root-finding.

2. Calculate a candidate step by minimising the model inside the 'trust region'; i.e. find $\mathbf{s}_k$ solving

$$\min_{\mathbf{s}\in\mathbb{R}^n} m_k(\mathbf{s}), \qquad \text{subject to} \qquad \|\mathbf{s}\| \leq \Delta_k, \tag{1.6}$$

where $\|\cdot\|$ is the Euclidean 2-norm. If m_k is quadratic, there are efficient methods for solving (1.6) to global optimality, or to find approximate solutions [53, Chapter 7].

3. Evaluate $f(\mathbf{x}_k + \mathbf{s}_k)$ and calculate a measure of progress from this step.

4. If sufficient progress was made, then accept the step ($\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$) and choose $\Delta_{k+1} \geq \Delta_k$, otherwise reject the step ($\mathbf{x}_{k+1} = \mathbf{x}_k$) and choose $\Delta_{k+1} < \Delta_k$.

Derivative-Free Trust-Region Methods In our model-based DFO methods, we will use a similar approach to derivative-based trust-region methods. The main difference is that we must replace (1.5) with a method for model construction that avoids the use of derivatives of f. To this end, we use *interpolation*: we maintain a set of interpolation points $Y_k \subset \mathbb{R}^n$, which we update on each iteration by adding the new point $\mathbf{x}_k + \mathbf{s}_k$ (and removing an old point, to keep the number of interpolation points constant). However, to achieve an effective, convergent method, we must ensure Y_k does not become degenerate—for instance, all points lying in a subspace—by checking and ensuring Y_k has 'good' geometry. This measure of 'goodness' is usually based on the Lagrange polynomials[6] for Y_k [60]; more details are given in Section 2.2.

The basic structure of trust-region model-based DFO methods for solving (1.1) is:

1. Construct a local (usually quadratic) model $m_k(\mathbf{s}) \approx f(\mathbf{x}_k + \mathbf{s})$ by enforcing the interpolation conditions $m_k(\mathbf{y} - \mathbf{x}_k) = f(\mathbf{y})$ for all $\mathbf{y} \in Y_k$. For instance, if $|Y_k| = (n+1)(n+2)/2$, then, provided the geometry of Y_k is not degenerate, there is a unique quadratic[7]

$$m_k(\mathbf{s}) = c_k + \mathbf{g}_k^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_k \mathbf{s}, \tag{1.7}$$

with $H_k = H_k^\top$, satisfying $m_k(\mathbf{y} - \mathbf{x}_k) = f(\mathbf{y})$ for all $\mathbf{y} \in Y_k$;

2. Calculate a candidate step by minimising the model inside the trust region (1.6);

3. Evaluate $f(\mathbf{x}_k + \mathbf{s}_k)$ and calculate a measure of progress from this step.

4. If sufficient progress was made, then accept the step ($\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$) and choose $\Delta_{k+1} \geq \Delta_k$, otherwise reject the step ($\mathbf{x}_{k+1} = \mathbf{x}_k$) and choose $\Delta_{k+1} < \Delta_k$.

[6] i.e. the set of polynomials that take the value 1 at one point in Y_k and 0 at the remainder; see Section 2.2.

[7] The degrees of freedom for a quadratic function defined on $\mathbb{R}^n$ is $1 + n + n(n+1)/2 = (n+1)(n+2)/2$.

5. Update the interpolation set: add $\mathbf{x}_{k+1}$ to Y_k and remove a point (usually one that is far from $\mathbf{x}_{k+1}$), and if sufficient progress was not made, ensure the geometry of Y_k is 'good' by (possibly) changing some points in Y_k.

This procedure is illustrated in Figure 1.1 earlier (with the last image corresponding to the case where sufficient progress was made and the step accepted).

Both the derivative-based and derivative-free versions of trust-region methods are globally convergent [53, 60]. In essence, this relies on the model construction method (Taylor series, interpolation, etc.) becoming more accurate as $\Delta_k \to 0$. This ensures that after a certain number of iterations where the objective was not reduced sufficiently, Δ_k becomes so small that the model is guaranteed to be accurate, so minimising the model necessarily reduces the objective, and progress towards a solution continues.

Development of Polynomial Interpolation Models The first derivative-free model construction was proposed by Winfield [228], who suggested a trust-region method with a quadratic interpolation model built from the $(n+1)(n+2)/2$ points where the objective has already been evaluated that are closest to the current iterate, including the iterate itself. The next key development in model-based DFO was the COBYLA algorithm by Powell [177], which solves general nonconvex problems with nonlinear inequality constraints. COBYLA is based on linear interpolating models, both for the objective and the constraints. One issue here is that the linearised constraints may produce an infeasible trust-region subproblem; in this case, the remedy is to ignore the objective, and compute a point to improve feasibility only.

More recently, Powell [182] introduced the idea of underdetermined quadratic interpolation models. This is applicable to the situation where we have $n+1 < p+1 < (n+1)(n+2)/2$ interpolation points including the current iterate (i.e. between linear and quadratic). Motivated by quasi-Newton methods, the idea is to build an interpolating quadratic model (1.7) where the model parameters c_k, $\mathbf{g}_k$ and H_k are the minimisers of

$$\min_{c,\mathbf{g},H} \|H - H_{\text{prev}}\|_F^2 \quad \text{subject to} \quad H = H^\top \text{ and } \quad m(\mathbf{y}_t - \mathbf{x}) = f(\mathbf{y}_t), \quad \forall t = 0, \ldots, p, \tag{1.8}$$

where H_{prev} is the Hessian of the previous model (initially set to zero). The interpolation problem (1.8) yields a linear system of size $p+n+2$. There is an added benefit that changing a single interpolation point yields a rank-2 update of this system (so its inverse can be maintained via the Sherman–Morrison–Woodbury formula). Importantly, the rank-2 update of the inverse matrix involves terms that are related to the Lagrange polynomials of the interpolation points. More details on this approach are given in Section 2.2.3.

Problems of the form (1.8) were studied by Zhang [241] in terms of the Sobolev seminorm of the quadratic model, who used this analysis to propose replacing $\|H - H_{\text{prev}}\|_F^2$ in the objective of (1.8) with

$$\|H - H_{\text{prev}}\|_F^2 + \frac{n+2}{M^2\Delta^2}\|\mathbf{g} - \mathbf{g}_{\text{prev}} - H_{\text{prev}}(\mathbf{x}_k - \mathbf{x}_{k-1})\|^2, \tag{1.9}$$

where $\mathbf{g}_{\text{prev}}$ is model gradient term in (1.7) from the previous iteration, and $M > 0$ is some constant (e.g. $M = 10$). A similar choice of objective to (1.9) had previously been proposed by Powell [187].

Typically, the interpolation set is kept at a constant size across iterations, so when a new point is added, one must be removed. An alternative to this was studied by Wild [223], who considered building underdetermined quadratic models with differing numbers of interpolation points at each iteration.

Impact of Geometry Conn and Toint [61] noted that the accuracy of the model is dependent on the geometry of the interpolating set, and showed that maximising Lagrange polynomials is a useful way to find a geometry-improving point.[8] The benefits of Lagrange polynomials were described by Powell [179], including cheap updating if a single interpolation point is changed, and being able to bound the error between a fully quadratic interpolating model m (see Section 2.2.3 for details) and an objective f by

$$|m(\mathbf{y}) - f(\mathbf{y})| \leq \text{const} \sum_{t=0}^{p} |\ell_t(\mathbf{y})| \, \|\mathbf{y} - \mathbf{y}_t\|^3, \tag{1.10}$$

for an interpolation set $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ with $p+1 = (n+1)(n+2)/2$, and Lagrange polynomials $\{\ell_0, \ldots, \ell_p\}$.

It was noted by Fasano, Morales and Nocedal [78] that good numerical performance could be observed in practice by a model-based algorithm that did not consider the geometry of the interpolation set; although the interpolation system can become ill-conditioned and the method can have more unsuccessful steps, it ultimately solves problems with a similar number of evaluations of f (saving on the cost of evaluating f at geometry-improving points). However, Scheinberg and Toint [203] provided an example where geometry must be considered to guarantee convergence.

An alternative approach for ensuring good geometry of the interpolation set was proposed by Marazzi and Nocedal [147]. They introduce the idea of a 'wedge method', where the trust-region subproblem (1.6) has an extra constraint $\mathbf{s} \notin \mathcal{W}_k$. The 'wedge' set $\mathcal{W}_k$ is defined

[8] We note that the standard measure of 'good geometry', Λ-poisedness, requires the Lagrange polynomials to take small values (see Section 2.2 for details). However, given an existing interpolation set, selecting new geometry-improving points involves finding maximisers of Lagrange polynomials, as described in Section 2.3.

by a priori selecting the interpolation point that will be replaced by the trust region step, then taking $\mathcal{W}_k$ to be a relaxation of the set of points that would make the new interpolation set degenerate. They provide methods for solving this new trust-region subproblem for both fully linear and fully quadratic models.

Alternative Model Constructions Aside from polynomial interpolation, other approaches for model construction have been proposed. One important category of methods uses radial basis function interpolation; that is, interpolants of the form

$$m(\mathbf{y}) = \sum_{t=0}^{p} \alpha_t \phi(\|\mathbf{y} - \mathbf{y}_t\|) + q(\mathbf{y}), \tag{1.11}$$

where q is some low-degree polynomial (e.g. linear) and $\phi : \mathbb{R} \to \mathbb{R}$ is a radial function (e.g. Gaussian). As with the polynomial case, the geometry of the interpolation set must be nondegenerate in order to construct the model. The first DFO method based on radial basis functions, called BOOSTERS, was developed by Oeuvray and Bierlaire [165, 167]. One issue that must be addressed in this setting is that the interpolant is no longer quadratic, and so the standard techniques for solving the trust region subproblem do not apply. This is addressed by using a general SQP solver for the trust region subproblem (in low dimensions), or by backtracking Armijo linesearch along the steepest descent direction (in higher dimensions).

The use of radial basis function models was extended by Wild, Regis and Shoemaker [226], whose ORBIT method allowed the number of interpolation points to vary between iterations (in practice, between $n + 1$ and $3n$). Global convergence of ORBIT was proven in [224, 227].

The SNOWPAC algorithm by Augustin and Marzouk [17] combines polynomial and radial basis function interpolation models, in order to improve the accuracy of models for noisy problems. Specifically, they take all previously-evaluated points in $B(\mathbf{x}_k, 2\Delta_k)$ and construct a Gaussian Process model for the objective (this is very similar to radial basis function interpolation; see [192]). Such models provide a distribution for the possible objective values at each point. Then, they build a polynomial interpolation model, but with the modified interpolation conditions

$$m_k(\mathbf{y}_t - \mathbf{x}_k) = \gamma_{k,t} f(\mathbf{y}_t) + (1 - \gamma_{k,t}) m_{\mathrm{GP}}(\mathbf{y}_t), \qquad \forall t = 0, \ldots, p, \tag{1.12}$$

where m_{GP} is the expected value of the Gaussian Process model, and $\gamma_{k,t} \in [0, 1]$ is a weighting factor. They set $\gamma_{k,t} = \exp(-\sigma_{\mathrm{GP}}(\mathbf{y}_t))$, where σ_{GP} is the standard deviation of the Gaussian Process model, to balance the uncertainty in the (noisy) f values and the approximation m_{GP}.

Lastly, Kannan and Wild [115] considered an alternative quadratic interpolation scheme, designed for noisy objectives. Instead of imposing exact interpolation, they relax (1.8) and try to solve

$$\min_{c,\mathbf{g},H} \|H\|_F^2 \quad \text{subject to} \quad H = H^\top \text{ and } \quad |m(\mathbf{y}_t - \mathbf{x}) - f(\mathbf{y}_t)| \leq \epsilon, \quad \forall t = 0, \ldots, p, \quad (1.13)$$

for some estimate $\epsilon > 0$ of the maximum noise in the objective. An important issue that needs to be considered in this formulation is that, if the problem is overdetermined (i.e. $p + 1 > (n + 1)(n + 2)/2$), then (1.13) may have no solution for ϵ sufficiently small; however, it is shown that there always exists a nonempty interval $[\epsilon_1, \epsilon_2]$, where (1.13) is feasible for $\epsilon \geq \epsilon_1$, and $H \neq 0$ for $\epsilon < \epsilon_2$.

We also note that there are alternative approaches for step calculation, other than the trust-region subproblem (1.6). Huang and Zhu [106] develop an interpolation-based ARC method (i.e. a model-based variant of (1.4)), which includes variable scaling and box constraints, and prove a superlinear local convergence rate, and Hare and Lucet [105] replace a trust-region framework with a proximal point method (quadratic regularisation).

Convergence Theory The first global convergence result for model-based DFO was developed by Conn, Scheinberg and Toint [55], which was based on an algorithm using Newton fundamental polynomials, instead of Lagrange polynomials as in (1.10). The next key breakthroughs in the convergence theory for model-based DFO were developed in a series of papers by Conn, Scheinberg and Vicente. In [59], global first- and second-order convergence results were established in a trust-region framework; this framework is not specific to one model-construction approach, and instead allows any approach that can generate 'fully linear/quadratic' models (defined in Section 2.2). The concept of Λ-poisedness (based on Lagrange polynomials, also defined in Section 2.2) was introduced in [57] for interpolation models, and it was shown that this yields the required fully linear/quadratic models. These results were extended to underdetermined (quadratic) interpolation and regression models in [58].

This theoretical framework was extended to include worst-case complexity bounds by Garmanjani, Júdice and Vicente [83] for first-order and Júdice [114] for second-order. These bounds match those of direct search [221] and derivative-based methods [41, 45], namely at most $k = \mathcal{O}(\epsilon^{-2})$ iterations to first achieve $\|\nabla f(\mathbf{x}_k)\| \leq \epsilon$, and $k = \mathcal{O}(\epsilon^{-3})$ iterations to first achieve $\max(\|\nabla f(\mathbf{x}_k)\|, -\lambda_{\min}(\nabla^2 f(\mathbf{x}_k))) \leq \epsilon$.

Probabilistic Model Construction More recent work, motivated in part by the use of DFO for noisy problems, has been on understanding the convergence of trust-region methods with probabilistically-accurate models. In the first work on this topic, Bandeira, Scheinberg and Vicente [22] extended the convergence theory from [59] to situations where models are constructed at each iteration, and are fully linear/quadratic with some probability $p_m > 1/2$ (rather than using geometry-improving steps to force full linearity on specific iterations). They prove convergence to first/second-order stationary points with probability 1. Worst-case complexity bounds were proven for this framework by Cartis and Scheinberg [48] for linesearch and ARC methods, and by Gratton, Royer, Vicente and Zhang [93] for trust-region methods. In both cases, the number of iterations is greater than for the deterministic case (i.e. $p_m = 1$) by a factor of $(2p_m - 1)^{-1}$.

This framework was extended by Chen, Menickelly and Scheinberg [50] to include probabilistic accuracy in both model construction and in objective evaluations when checking for sufficient decrease; the requirement is that the error in objective evaluations, specifically for checking decrease, is $\mathcal{O}(\Delta_k^2)$ with some probability. They prove convergence of a general algorithm, called STORM, which does not prescribe how probabilistically-accurate models and evaluations are achieved (although examples are provided). One approach is to use sample averaging with at least Δ_k^{-4} samples for each point; another, which we will use for our testing in Section 6.3, uses this sample averaging for checking objective decrease, but uses overdetermined regression models with Δ_k^{-4} points. The analysis of STORM was extended to worst-case complexity bounds by Blanchet, Cartis, Menickelly and Scheinberg [31].

In a related work, Jamieson, Nowak and Recht [111] show that, for general strongly convex functions, optimising using only noisy objective evaluations requires $\Omega(\epsilon^{-2})$ evaluations to achieve optimality gap less than ϵ, regardless of the algorithm used.

Model-Based DFO for Noisy Problems Since optimising noisy objectives is one of the key regimes where DFO can be useful, there has been a large amount of work on noisy problems specifically. The assumption is that we wish to optimise the expected value of the noisy objective.

One approach, already mentioned, is the development of *regression quadratic models* by Conn, Scheinberg and Vicente in [58]. The idea is that if we try to fit our model to more points than we have degrees of freedom in the model, then we should be able to reduce the impact of noise in the evaluations; a contribution of [58] is to prove that regression models are fully linear. This was extended by Billups, Larson and Graf [30], who show that this theory holds for weighted regression models (provided the regression Lagrange polynomials are constructed using the same weights).

A convergent algorithm based on the use of regression models to produce probabilistically-fully-linear models was proposed by Larson and Billups [127]. Here, linear regression models are suggested, and the number of sample points grows like $k\Delta_k^{-4}$, so the model accuracy grows as the algorithm progresses (c.f. STORM with sample averaging [50], described above).

Another standard approach for limiting the impact of noise in model construction is through *sample averaging*; that is, requiring that the interpolating model satisfies $m(\mathbf{y}_t - \mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{y}_t; \xi_i)$, where ξ_i are random variables representing the noise in the evaluation of f. The use of sample averaging was first proposed by Deng and Ferris [70]. Here, they extend the UOBYQA algorithm [180] to use sample averaging, where the number of samples is chosen dynamically—after a given number of evaluations of each interpolation point, they derive a posterior estimate of the model coefficients, which ultimately leads to a distribution over trust-region subproblem solutions (via a Monte Carlo simulation). The number of samples used in model construction is increased until the distribution of candidate trust-region steps has sufficiently small variance.

The same authors used a different Bayesian approach to sample averaging in [71]. Here, the available samples are used to estimate the distribution of the true interpolating model, and more samples are added until the probability that the computed trust-region step satisfies a Cauchy decrease condition in the true model is sufficiently large.

Another sample averaging method called ASTRO-DF was proposed more recently by Shashaani, Hashemi and Pasupathy [206]. Here, the model construction algorithm considered geometry and sampling errors simultaneously; interpolation points are added and the trust-region radius is decreased (until the model gradient is sufficiently large compared to the trust-region radius), where each interpolation point is sampled until the sample variance is sufficiently small. Global first-order convergence with probability 1 is proved for both [71] and [206].

Along different lines, Moré and Wild proposed approaches for both efficiently *estimating the level of noise* in objective evaluations [158] and using this to determine adaptive step sizes for accurate finite differencing [159]. This was extended by Berahas, Byrd and Nocedal [25] to implement a quasi-Newton linesearch with noise-aware adaptive finite differencing. A derivative-free variant of steepest descent with linesearch for noisy functions was analysed by Paquette and Scheinberg [168] in the context of probabilistic models (see above).

Model-Based DFO for Composite Functions The flexibility of model-based DFO means there has been work to adapt these approaches to specific classes of problems. Given the importance of nonlinear least-squares problems to this thesis, we address these separately in Section 1.1.3.

The first attempt to construct models exploiting problem structure was by Colson and Toint [52], who considered 'partially-separable' functions, which take the form

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}|_{\mathcal{I}_i}), \tag{1.14}$$

for index sets $\mathcal{I}_i \subset \{1, \ldots, n\}$, usually with $|\mathcal{I}_i| \ll n$. Here, we assume that we can evaluate all values $f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})$ for a given $\mathbf{x}$ (i.e. we receive a vector of values in $\mathbb{R}^m$, rather than just the scalar $f(\mathbf{x})$). The approach is then to build quadratic interpolation models for each f_i, which, since $|\mathcal{I}_i| \ll n$, requires fewer objective evaluations than building a quadratic model for the (high-dimensional) full objective f directly.

An alternative approach was presented by Garmanjani, Júdice and Vicente [83], where they extended their complexity results (see above) to the composite problem

$$\min_{\mathbf{x}} h(\mathbf{r}(\mathbf{x})), \tag{1.15}$$

where h is convex (and possibly nonsmooth); this is achieved by building a linear model for $\mathbf{r}$, yielding the approximate model $h(\mathbf{x}_k + \mathbf{s}) \approx h(\mathbf{r}(\mathbf{x}_k) + J_k\mathbf{s})$, where J_k approximates the Jacobian of $\mathbf{r}$ at $\mathbf{x}_k$. Similarly, Grapiglia, Yuan and Yuan [90] consider the more general problem

$$\min_{\mathbf{x}} f(\mathbf{x}) := g(\mathbf{x}) + h(\mathbf{r}(\mathbf{x})), \tag{1.16}$$

where g and $\mathbf{r}$ are smooth and (possibly) nonconvex, and h is convex (but possibly nonsmooth). They also provide a worst-case complexity analysis—building on the derivative-based analysis by Cartis, Gould and Toint [44]—based on building models

$$f(\mathbf{x}_k + \mathbf{s}) \approx g(\mathbf{x}_k) + \mathbf{g}_k^\top \mathbf{s} + h(\mathbf{r}(\mathbf{x}_k) + J_k\mathbf{s}) + \frac{1}{2}\mathbf{s}^\top H_k \mathbf{s}, \tag{1.17}$$

where $\mathbf{g}_k$ and J_k approximate the gradient and Jacobian of g and $\mathbf{r}$ at $\mathbf{x}_k$ respectively (e.g. from interpolation) and H_k is designed to capture curvature in both g and $h \circ \mathbf{r}$. This method is then specialised to the case of a penalty method for equality-constrained minimisation. We note that the complexity analysis here yields a first-order bound on evaluations with an extra factor of $|\log \epsilon|$ compared to [83], due to how it includes iterations of the criticality phase; this is discussed more in Section 4.3.

The treatment of (1.15) was specialised to $h(\mathbf{r}(\mathbf{x})) := \|\mathbf{r}(\mathbf{x})\|_1$ by Larson, Menickelly and Wild [128], to $h(\mathbf{r}(\mathbf{x})) := \max_i r_i(\mathbf{x})$ and f convex by Hare, Planiden and Sagastizábal [104], and to piecewise-linear h by Khan, Larson and Wild [120].

Lastly, using tools from compressive sensing, Bandeira, Scheinberg and Vicente [21] considered the problem of quadratic interpolation when the objective has a sparse Hessian (or is well-approximated by quadratic functions with sparse Hessians). Specifically, by framing

the quadratic interpolation problem as minimising the ℓ_1 norm of the quadratic terms (subject to exact interpolation), they show that a fully quadratic model can be constructed from $\mathcal{O}((n_H + n)\log(n_H + n)\log n)$ randomly-chosen interpolation points, where n_H is the number of nonzero entries in the (true) objective Hessian, an improvement on the usual $\mathcal{O}(n^2)$ if $n_H \ll n^2$.

Other Problem Classes We briefly note that there is a body of work on model-based DFO for constrained problems, including strictly feasible methods based on convexifying the feasible set [16, 196], filter methods [33, 79], active set methods [219, 95], SQP methods [216], and augmented Lagrangian methods [236, 13]. We also mention that model-based DFO methods have been developed for general nonsmooth (and possibly nonconvex) problems by [83, 139]; for bilevel optimisation problems [62]; and for min-max problems [150]. A more complete survey of this material can be found in [129].

1.1.3 Nonlinear Least-Squares Problems

As mentioned above, an important feature of model-based DFO is that it is well-placed to exploit problem structure, if this information is known. A major focus of this thesis is nonlinear least-squares problems, where f in (1.1) has the form

$$f(\mathbf{x}) := \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{x})^2, \tag{1.18}$$

for $\mathbf{r} : \mathbb{R}^n \to \mathbb{R}^m$ smooth and possibly nonlinear. We assume that we have access to evaluations of $\mathbf{r}(\mathbf{x})$, not just $f(\mathbf{x})$. This problem type appears frequently in parameter fitting, and also for solving nonlinear systems of equations, where we typically have $m = n$, and wish to find an $\mathbf{x}$ satisfying $\mathbf{r}(\mathbf{x}) = \mathbf{0}$ [164, Chapter 10]. In this section, we first outline derivative-based methods, then the key model-based DFO contributions, for solving (1.18).

Derivative-Based Gauss-Newton Methods Classically, nonlinear least-squares problems are solved with the (derivative-based) Gauss-Newton and/or Levenberg-Marquardt methods [164, Chapter 10]. Defining the Jacobian matrix $J(\mathbf{x}) \in \mathbb{R}^{m\times n}$ of $\mathbf{r}$, with rows $\nabla r_i(\mathbf{x})^\top$ for $i = 1, \ldots, m$, we have

$$\nabla f(\mathbf{x}) = J(\mathbf{x})^\top \mathbf{r}(\mathbf{x}), \qquad \text{and} \qquad \nabla^2 f(\mathbf{x}) = J(\mathbf{x})^\top J(\mathbf{x}) + \sum_{i=1}^{m} r_i(\mathbf{x})\nabla^2 r_i(\mathbf{x}). \tag{1.19}$$

For both methods, we use only first-order terms in (1.19) and make the approximation

$$\nabla^2 f(\mathbf{x}) \approx J(\mathbf{x})^\top J(\mathbf{x}). \tag{1.20}$$

This corresponds to linearising $\mathbf{r}(\mathbf{x}+\mathbf{s}) \approx \mathbf{r}(\mathbf{x}) + J(\mathbf{x})\mathbf{s}$ and constructing the quadratic approximation for f,

$$f(\mathbf{x}+\mathbf{s}) \approx m(\mathbf{s}) := \frac{1}{2}\|\mathbf{r}(\mathbf{x}) + J(\mathbf{x})\mathbf{s}\|^2 = f(\mathbf{x}) + \mathbf{g}^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H\mathbf{s}, \tag{1.21}$$

where $\mathbf{g} := J(\mathbf{x})^\top \mathbf{r}(\mathbf{x}) = \nabla f(\mathbf{x})$ and $H := J(\mathbf{x})^\top J(\mathbf{x})$. The approximation (1.21) is quadratic, but based on only first-order information; i.e. it partially captures the curvature in f, but without requiring higher-order derivative information. Given (1.19), we may expect the approximation (1.20) to be better when $\mathbf{r}(\mathbf{x})$ is small (i.e. $f(\mathbf{x})$ is small), and we see this in the convergence theory.

Minimising m (1.21) exactly at each iteration gives the Gauss-Newton method

$$\mathbf{x}_{k+1} := \mathbf{x}_k - [J(\mathbf{x}_k)^\top J(\mathbf{x}_k)]^{-1} J(\mathbf{x}_k)^\top \mathbf{r}(\mathbf{x}_k), \tag{1.22}$$

but this has the drawback of not being well-defined if $J(\mathbf{x}_k)$ is rank-deficient. If we implement the model (1.21) inside a trust-region method, then the resulting algorithm is known as the Levenberg-Marquardt method. This is equivalent—in the case of solving the trust-region subproblem to full optimality—to a regularised Gauss-Newton step

$$\mathbf{x}_{k+1} := \mathbf{x}_k - [J(\mathbf{x}_k)^\top J(\mathbf{x}_k) + \lambda_k I]^{-1} J(\mathbf{x}_k)^\top \mathbf{r}(\mathbf{x}_k), \tag{1.23}$$

for some $\lambda_k \geq 0$ (determined by the trust-region subproblem) [154]. An alternative approach for the Levenberg-Marquardt algorithm is to explicitly choose λ_k at each iteration, and calculate our step using (1.23). This corresponds to a quadratic regularisation method, namely updating $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$ where $\mathbf{s}_k$ solves

$$\min_{\mathbf{s}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{r}(\mathbf{x}_k) + J(\mathbf{x}_k)\mathbf{s}\|^2 + \frac{\lambda_k}{2}\|\mathbf{s}\|^2. \tag{1.24}$$

The Levenberg-Marquardt method is globally convergent [164, Theorem 10.3] with a linear asymptotic rate in general, but a quadratic asymptotic rate for zero-residual problems (i.e. $f(\mathbf{x}^*) = 0$) [72, Theorem 10.2.6]. That is, the Gauss-Newton approximation (1.21) gives faster convergence for zero-residual problems than for nonzero residual problems, matching our intuition about the approximation (1.20).

Brown and Dennis [36] analysed the Levenberg-Marquardt method with finite differencing (using adaptively-chosen step size) to calculate $J(\mathbf{x}_k)$, and Kelley [118] proposed an implicit filtering version of the classical Gauss-Newton method (with backtracking linesearch) to estimate $J(\mathbf{x}_k)$ at each iteration.

Early DFO Methods There were several early DFO methods for nonlinear systems and nonlinear least-squares problems, largely based around linear interpolation. Powell [173] and Ralston and Jennrich [191] used linear interpolation models for $\mathbf{r}(\mathbf{x})$ constructed from the most recent iterates, and a model for the objective was built in the style of (1.21). In both cases a step is calculated using an analogue of (1.22), although Powell adds a linesearch mechanism. Peckham [169] used a similar approach but with model construction using weighted linear regression, and Spendley [208] modified the Nelder-Mead method to use occasional Gauss-Newton steps (constructed by linear interpolation of $\mathbf{r}(\mathbf{x})$ using the points in the current simplex).

Another early derivative-free technique, which has some similarity to quasi-Newton methods, is Broyden's method [37]. Here, an approximation to the Jacobian $J_k \approx J(\mathbf{x}_k)$ is updated by a rank-1 perturbation, after taking a step $\mathbf{x}_{k+1} := \mathbf{x}_k + \mathbf{s}_k$, namely:

$$J_{k+1} := J_k + \frac{1}{\|\mathbf{s}_k\|^2} \left(\mathbf{r}(\mathbf{x}_{k+1}) - \mathbf{r}(\mathbf{x}_k) - J_k \mathbf{s}_k\right) \mathbf{s}_k^\top, \tag{1.25}$$

which corresponds to the solution of

$$\min_{J \in \mathbb{R}^{m \times n}} \|J - J_k\|_F^2, \qquad \text{s.t.} \quad \mathbf{r}(\mathbf{x}_{k+1}) - \mathbf{r}(\mathbf{x}_k) = J\mathbf{s}_k. \tag{1.26}$$

This approach was included within a trust-region framework by Powell [175, 174], who proved its global convergence. This convergence applies as long as the steps $\{\mathbf{s}_k\}$ satisfy a uniform linear independence condition, which can be ensured via regular geometry-improving steps. This condition was studied in more detail by Moré and Trangenstein [156], who noted its similarity to both (1.2) and a property similar to Λ-poisedness for linear interpolation (Definition 2.11). If Jacobian-vector products are available, there are a variety of other low-rank updates of J_k that can be defined [146]. We also note that secant-like conditions can be used to approximate $\nabla^2 f(\mathbf{x}_k)$ (1.19) (see [152, 145] for examples in a derivative-based context).

Model-Based DFO Methods The first model-based DFO method for nonlinear least-squares was DFLS by Zhang, Conn and Scheinberg [240]. DFLS is based on building quadratic interpolation models for each component of $\mathbf{r}(\mathbf{x})$ separately:

$$r_i(\mathbf{x}_k + \mathbf{s}) \approx m_{k,i}(\mathbf{s}) := r_i(\mathbf{x}_k) + \mathbf{g}_{k,i}^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_{k,i}\mathbf{s}, \tag{1.27}$$

using standard approaches. Intuitively, one would then build a model for the full objective f via

$$f(\mathbf{x}_k + \mathbf{s}) \approx \frac{1}{2}\sum_{i=1}^{m} m_{k,i}(\mathbf{s})^2, \tag{1.28}$$

however this would yield a quartic model rather than the usual quadratic. The authors propose a second-order approximation to this quartic, enabling the usual approaches for solving the trust-region subproblem; that is, we take

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) := f(\mathbf{x}_k) + \mathbf{g}_k^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_k \mathbf{s}, \tag{1.29}$$

where $J_k \in \mathbb{R}^{m\times n}$ has rows $\mathbf{g}_{k,i}^\top$, $\mathbf{g}_k := J_k^\top \mathbf{r}(\mathbf{x}_k)$, and H_k is defined by

$$H_k := \begin{cases} J_k^\top J_k, & \text{if } \|\mathbf{g}_k\| \geq \kappa_1, \quad \textit{(Gauss-Newton)} \\ J_k^\top J_k + \kappa_3 \|\mathbf{r}(\mathbf{x}_k)\| I_{n\times n}, & \text{if } \|\mathbf{g}_k\| < \kappa_1 \text{ and } f(\mathbf{x}_k) < \kappa_2 \|\mathbf{g}_k\|, \textit{(Levenberg-Marquardt)} \\ J_k^\top J_k + \sum_{i=1}^m r_i(\mathbf{x}_k) H_{k,i}, & \text{if } \|\mathbf{g}_k\| < \kappa_1 \text{ and } f(\mathbf{x}_k) \geq \kappa_2 \|\mathbf{g}_k\|. \quad \textit{(Newton)} \end{cases} \tag{1.30}$$

The three cases correspond to three possible situations in the algorithm: use the Hessian derived from the Gauss-Newton method initially, then when we believe we are approaching a stationary point, we either use a regularised Gauss-Newton step, analogous to the Levenberg-Marquardt method, if we expect our problem to be zero residual, or a full Newton step if we expect it to be nonzero residual. The authors prove global convergence of DFLS, and provide numerical results showing it requires fewer evaluations than both a general objective model-based DFO solver (Powell's NEWUOA [184]; see below) and a derivative-based nonlinear least-squares solver with finite differencing. Later, Zhang and Conn [239] proved a local quadratic convergence rate of DFLS for zero-residual problems, matching the result for the Levenberg-Marquardt method.

The DFLS method was modified by Wild for POUNDERS [225], which always uses the 'Newton' form for H (1.30), and selects the interpolation points from the entire history of evaluations (choosing points close to the current trust region regardless of when they were evaluated).

A related work by Bergou, Gratton and Vicente [27] considers probabilistically-accurate Gauss-Newton style models (i.e. linear models for $\mathbf{r}$ with the natural quadratic model corresponding to (1.28)). In this case, global convergence is ensured using quadratic regularisation (1.24), but assuming the availability of sufficiently accurate approximations for $J(\mathbf{x}_k)$ and $\nabla f(\mathbf{x}_k)$ with some given probability. This work was extended by Bergou, Diouane, Kungurtsev and Royer [26] to include worst-case complexity bounds.

Similarly, Bellavia, Gratton and Riccietti [24] also consider a quadratic regularisation method, but where approximations for both $\mathbf{r}$ and the true Jacobian J can be computed to arbitrary accuracy (with the intent that cheaper, low accuracy approximations are used in early stages). The authors prove global convergence, worst-case complexity and show local convergence (in the sense that $\|\mathbf{x}_k - \mathbf{x}^*\| \to 0$ monotonically).

Linesearch DFO Methods More recently, there have been several papers studying linesearch methods for nonlinear systems, where several possibilities for the search direction are used, such as $\mathbf{s}_k = \pm\mathbf{r}(\mathbf{x}_k)$ and $\mathbf{s}_k = -J_k^{-1}\mathbf{r}(\mathbf{x}_k)$ for some suitably-updated J_k, such as (1.25). These include Li and Fukushima [134], who guarantee monotonicity of the iterates by using both choices of $\mathbf{s}_k$, and nonmonotone linesearches studied by La Cruz and Raydan [124], Grippo and Sciandrone [97], La Cruz, Martínez and Raydan [123], and Li [135]. This work has been extended by Li and Li [136] to the case where $\mathbf{r}(\mathbf{x})$ is monotone but not necessarily differentiable, and by La Cruz [122], Morini, Porcelli and Toint [160] and Marini, Morini and Porcelli [148] to systems with convex constraints. Separately, Begiato, Custódio and Gomes-Ruggiero [23] combine this approach with direct search.

1.1.4 Software and Numerical Testing

The work described above often includes numerical results from implementations of the relevant methods; here, we consider work specifically related to the development of DFO software and testing frameworks. Perhaps the largest contribution to software development for model-based DFO is by Powell, who produced several solvers largely based on quadratic interpolation models and geometry management via Lagrange polynomials. These Fortran packages, freely available from [242], include:

- COBYLA [177], which solves unconstrained minimisation problems subject to nonlinear inequality constraints via linear interpolation models for both the objective and the constraints;
- UOBYQA [180], which solves unconstrained minimisation problems using quadratic interpolation models with $(n+1)(n+2)/2$ points;
- NEWUOA [184, 185], which solves unconstrained problems using underdetermined quadratic interpolation models (1.8);
- BOBYQA [186], which extends NEWUOA to allow box constraints; and
- LINCOA [188], which allows for linear inequality constraints.

The software developed in this thesis inherits its general framework from BOBYQA (see Sections 3.1 and 3.3 for more algorithmic details). A superlinear convergence rate was proven for a variant of UOBYQA by Han and Liu [98], and the BOBYQA package was modified by Arouxét, Echebest and Pilotta [5] to use $\|\cdot\|_\infty$ for the trust-region subproblem (1.6), to more closely align with box constraints, and by Newby and Ali [163] to solve mixed-integer nonlinear programs.

The DFO package by Scheinberg [202] also uses quadratic models, and allows for general nonlinear constraints—to achieve this, the trust-region subproblem is augmented with these constraints and solved using derivative-based methods; hence, it requires that derivatives of the constraint functions can be provided (or estimated). Similarly, CONDOR by Vanden Berghen and Bersini [219] extends UOBYQA to allow general nonlinear constraints using an active set method, and also allows the calculation of geometry-improving points and subsequent objective evaluations in parallel to the main solver.

We also mention (S)NOWPAC by Augustin and Marzouk [16, 17], available at [18]. The NOWPAC package [16] handles general minimisation with nonlinear inequality constraints via underdetermined quadratic interpolation for both the objective and constraints; here, some work is needed to find feasible steps by constructing a suitable convex domain in which to solve the trust-region subproblem. This was extended in SNOWPAC [17] to handle noisy objectives and constraints, using a combination of Gaussian Process and polynomial interpolation models (see Section 1.1.1).

For nonlinear least-squares problems, the DFLS algorithm outlined above by Zhang, Conn and Scheinberg [240] is implemented in the Fortran package DFBOLS, which adapts BOBYQA to construct quadratic models for each residual function (including allowing box constraints). The POUNDERS algorithm by Wild [225] is implemented in TAO, the optimisation library associated with PETSc [20].

A more general work aimed at the development of high-quality software is by Larson and Wild [130]. This work considers the construction of sensible termination criteria based on measures such as objective decrease and convergence of iterates. We adopt one of these tests in our software (see Section 3.3.3).

Finally, we note the work of Moré and Wild [157], who outline a methodology for the testing of DFO algorithms. The two main contributions of this work are a now-common set of test problems, and the development of 'data profiles', which compare the performance of solvers on a test set based on (assumed expensive) objective evaluations. This complements the more well-known performance profiles of Dolan and Moré [73]; more details on this work are given in Section 2.4.

1.2 Research Questions and Contributions

This thesis primarily focuses on model-based DFO methods for finding local minimisers to unconstrained (or possibly bound-constrained) nonconvex problems, including general minimisation and nonlinear least-squares minimisation. There are two regimes where derivative-free methods are preferable to derivative-based methods that we specifically target in this thesis:

Expensive: objectives are expensive to evaluate, and may be noiseless. Here, the goal is to make reasonable progress—not necessarily reaching high accuracy in the solution—using very few evaluations; and,

Noisy: objectives that contain noise, but may be cheap(er) to evaluate. This noise may be either stochastic or deterministic (e.g. arising from the finite termination of an iterative procedure). We aim to improve the robustness of the solver—maximising the amount of progress the solver can make, and hence, the number of problems that can be solved—despite the difficulties associated with inaccurate local models and objective evaluations.

We note that these two regimes are not mutually exclusive. It is entirely possible for expensive objectives to also have inaccuracies in their calculation, such as parameter fitting for climate models [211]. To this end, when we compare the performance of methods for the noisy regime, we consider both the overall robustness of each method, and also the objective evaluation budget required by the methods. In addition, throughout, we develop our algorithms under the consideration of the cost and scalability of the algorithm. That is, we always aim for algorithms with efficient implementations (e.g. in the relevant linear algebra) and therefore that have greater potential to provide reasonable performance for larger-scale problems.

The research questions addressed in this thesis, and our key contributions, are:

1. *How can we use the structure of nonlinear least-squares problems to produce efficient model-based DFO methods?*

   We develop DFO-LS (Derivative-Free Optimisation for Least-Squares), a model-based DFO method for nonlinear least-squares problems based on the classical (derivative-based) Gauss-Newton method. Compared to DFBOLS, the implementation from [240], the key difference is that it constructs linear interpolation (or optionally regression) models for each residual, rather than quadratic models, without loss of performance. Although the theoretical algorithm DFLS from [240] allows for linear models, its implementation DFBOLS and numerical results require quadratic models. We provide a global convergence and worst-case complexity analysis of DFO-LS, carefully counting the evaluation cost of model construction. Our results extend the analysis in [240]—which only showed global convergence—to allow inexact solutions of the trust-region subproblem, and showing that the whole sequence, not just a subsequence, of the gradients at the iterates converges to zero. We show that, similar to the derivative-based case, DFO-LS has the same worst-case complexity as general objective

methods, but with a dependency on dimension that is between first- and second-order methods, corresponding to it partially capturing curvature information, as per (1.20). We also show numerically that little to nothing is lost by this simplified approach in terms of algorithm performance on a given evaluation budget, when applied to smooth and noisy, zero- and nonzero residual problems. Furthermore, significant gains are made in terms of reduced computational cost of the interpolation problem and memory costs of storing the models.

When the high computational cost of evaluations is more of a concern than scalability, DFO-LS still offers the advantage of a reduced evaluation cost for the initialisation, again due to choosing a smaller interpolation set, without loss in overall performance. However, if objective evaluations are very expensive, even this reduced initialisation cost (of $n + 1$ evaluations for an n-dimensional problem), may still be too high. Thus, we augment DFO-LS with the ability to begin the main iteration after an initial setup cost of as few as 2 objective evaluations. After this low setup cost, subsequent evaluations are selected based on minimising a model for the objective, and represent a genuine attempt to progress towards the solution.[9]

The DFO-LS algorithm, including the construction of linear models and the mechanism for reducing the initialisation cost, is outlined in Chapter 3. We give the convergence and worst-case complexity analysis for DFO-LS in Chapter 4. Chapter 5 has extensive numerical results, demonstrating the benefits of linear residual models described above and the performance of DFO-LS with reduced initialisation cost.

2. *How can model-based DFO methods be made more robust to noise through techniques that have a low cost in both evaluations and computations?*

   The implementation of DFO-LS includes several features aimed at improving its performance in the noisy regime. In particular, DFO-LS implements different default trust-region parameters for noisy problems, regression models, a flexible regime for sample averaging, and a novel multiple restart mechanism coupled with an auto-detection strategy for identifying when restarts should occur. We perform several numerical studies of these features, and conclude that multiple restarts is an effective way at achieving a high level of robustness to noise. Additionally, compared to other mechanisms (such as sample averaging, regression models, and surrogate model

[9] In practice, solvers can accept any objective decrease from the initial evaluations [186, 12], and use the best initial point as the first iterate. DFO-LS does this, but here we refer to a mechanism to begin the main iteration after few initial evaluations, where new iterates are specifically generated to progress towards the solution from the best point so far, rather than just sample the search space.

construction), multiple restarts benefit from being cheap to implement in terms of evaluations and computation.

The multiple restarts mechanism is not specific to the nonlinear least-squares problem structure. Therefore we also introduce Py-BOBYQA, a Python implementation of BOBYQA (for general minimisation with optional bound constraints) [186], which uses quadratic interpolation models for the objective. Py-BOBYQA implements several enhancements from DFO-LS, such as multiple restarts, and we modify the theoretical analysis of DFO-LS to give convergence and worst-case complexity results. Our numerical results demonstrate that multiple restarts give a substantially improved robustness to noise compared to the original BOBYQA, and also perform well compared to other methods specifically designed for noisy problems (such as STORM and SNOWPAC; see Section 1.1.2).

Multiple restarts allow the algorithm to escape regions where the noise in the objective dominates genuine changes in the objective value, and we observe that this mechanism can also be used to escape from local minima. With only a small adjustment, we demonstrate that Py-BOBYQA with multiple restarts yields an effective heuristic for escaping local minima with few objective evaluations, by comparison to global optimisation routines, on a collection of global optimisation test problems.

The multiple restarts mechanism (as well as the sample averaging and regression model mechanisms) for DFO-LS is outlined in Chapter 3, and tested numerically in Chapter 5. The Py-BOBYQA algorithm is introduced, analysed and tested in Chapter 6.

3. *How can model-based DFO methods be modified to improve their scalability?*

   Having DFO methods suitable for large-scale nonconvex problems is important for nonlinear least-squares problems such as data assimilation, and general minimisation problems such as adversarial example generation for neural networks. However, the high linear algebra cost of model-based DFO methods means that they are unable to efficiently solve such problems.

   To address this issue, we introduce DFBGN (Derivative-Free Block Gauss-Newton), a model-based DFO method for nonlinear least-squares problems that relies on constructing a model in a subspace at each iteration. To our knowledge, this is the first *subspace* model-based DFO method. Our novel approach enables model-based DFO methods to be applied in a large-scale regime by giving the user explicit control over the linear algebra cost of the method at each iteration. The framework of DFBGN is similar to DFO-LS, but it reduces the linear algebra cost of model construction

by allowing for fewer interpolation points at every iteration. We do not prescribe a particular choice for the interpolation set (and corresponding subspace) size, but instead allow it to be user-specified based on their computational resources.

The DFBGN algorithm is simpler than DFO-LS, not including several algorithm components, and instead relying on replacing some existing interpolation points with randomly-chosen orthogonal directions at each iteration. Our approach for replacing interpolation points is geometry-aware and keeps the trust-region radius coupled with the model gradient, without using more complex strategies from DFO-LS.

We show that DFBGN with a full-sized interpolation set has similar performance to DFO-LS in terms of objective evaluations, but shows improved performance on runtime. As the size of the interpolation set changes, we demonstrate a tradeoff between reduced linear algebra costs and increased evaluations required to achieve a given objective reduction. The flexibility of DFBGN allows this tradeoff to be explicitly managed. When tested on large-scale problems, DFO-LS frequently reaches a reasonable runtime limit without making substantial progress, whereas DFBGN with small subspace size can perform many more iterations and hence make better progress than DFO-LS. In the case of small budgets, DFBGN can make progress with few objective evaluations in a similar way to DFO-LS, but with substantially lower linear algebra costs.

The DFBGN algorithm is described and tested in Chapter 7.

Software Availability The solvers DFO-LS and Py-BOBYQA are both publicly-available on Github.[10] They are released under the GNU General Public License. DFBGN will be made publicly-available in the future.

Thesis Structure In Chapter 2 we discuss a variety of relevant technical background work, including polynomial interpolation and geometry-management for DFO and numerical testing methodologies. Then, in Chapter 3, we introduce the DFO-LS algorithm and the features of its implementation, including linear residual models, reduced initialisation cost, and multiple restarts. The global convergence and complexity results for DFO-LS are given in Chapter 4. The features of DFO-LS are tested numerically in Chapter 5, which also includes a comparison of DFO-LS against other solvers. In Chapter 6, we describe Py-BOBYQA for general minimisation, provide our theoretical analysis, and show show its numerical performance. We introduce and test DFBGN for large-scale nonlinear least-squares problems

[10] At `https://github.com/numericalalgorithmsgroup/dfols` and `https://github.com/numericalalgorithmsgroup/pybobyqa` respectively.

in Chapter 7. Finally, we summarise our conclusions and provide several possible future research directions in Chapter 8.

Statement of Originality At the time of writing, three papers based on the research in this thesis are publicly-available, of which the first two have been published. The third has been submitted for publication.

The first, [47], co-authored with Coralia Cartis, proposes, analyses and tests the use of linear residual interpolation models. The second, [39], is co-authored with Coralia Cartis, Jan Fiala and Benjamin Marteau, and introduces both DFO-LS and Py-BOBYQA, and focuses on their new algorithmic features and implementation. The content from these papers appears in Section 2.4.2 and Chapters 3–6, except for Section 6.4.

The third, [49], is co-authored with Coralia Cartis and Oliver Sheridan-Methven. It proposes and tests the modification of Py-BOBYQA to escape local minima. The content from this paper appears here in Section 6.4.

All work in this thesis was carried out by the author, under the supervision of Coralia Cartis, Jan Fiala and Benjamin Marteau, except for the testing framework in Section 6.4, which was developed by the author in conjunction with Oliver Sheridan-Methven.

Chapter 2

Technical Background

In this chapter, we provide the technical foundations for the remainder of the thesis. We begin by defining the problems under consideration and their underlying assumptions, especially around their smoothness, in Section 2.1. Next, we summarise the key existing theoretical results from the literature—which can be found in [60], for example—concerning polynomial interpolation (Section 2.2) and associated methods for managing the geometry of interpolation sets (Section 2.3).

The last part of this chapter, Section 2.4, outlines the framework for evaluating the numerical performance of DFO methods. This framework is largely based on the approach in [157], but with an expanded collection of test problems. Additionally, we extend the framework from [157] to include a novel mechanism for the adaptive choice of accuracy thresholds for noisy problems.

We recall from Section 1.1 that we use $\|\cdot\|$ for the 2-norm of vectors and matrices, $\|\cdot\|_F$ for the Frobenius norm of matrices, and define the ball $B(\mathbf{x}, \Delta) := \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{y} - \mathbf{x}\| \leq \Delta\}$ for $\mathbf{x} \in \mathbb{R}^n$ and $\Delta > 0$.

2.1 Problem Classes and Assumptions

There are two broad problem classes we will consider in this thesis: general (nonconvex) minimisation and nonlinear least-squares minimisation. In both cases, we will undertake theoretical analysis of algorithms for unconstrained problems, but our software implementations will allow for optional bound constraints. We also note that although we may assume that the objective is differentiable, our methods are derivative-free in the sense that they do not employ gradient evaluations at any point. Similarly, although we assume the objective is smooth, we sometimes consider problems with noise (i.e. only inaccurate evaluations of the objective are available).

2.1.1 General Minimisation

In the general minimisation setting, we are interested in developing algorithms for solving

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is some smooth, possibly nonconvex, function. Problem (2.1) will form the basis of our theoretical analysis, but we to make assumptions about f. To this end, we define the relaxed level set

$$\mathcal{L}(\mathbf{x}_0, \Delta_{max}) := \text{conv}\left(\{\mathbf{y} \in \mathbb{R}^n : \mathbf{y} \in B(\mathbf{x}, \Delta_{\max}) \text{ for some } \mathbf{x} \text{ with } f(\mathbf{x}) \leq f(\mathbf{x}_0)\}\right), \tag{2.2}$$

where conv$(\cdot)$ is the convex hull and, in the assumptions below, $\mathbf{x}_0 \in \mathbb{R}^n$ will be the starting point for the algorithm and $\Delta_{\max} > 0$ will be an upper bound on the trust-region radius.

Assumption 2.1. *Given $\mathbf{x}_0 \in \mathbb{R}^n$ and $\Delta_{\max} > 0$, the function f in (2.1) is continuously differentiable, and its gradient is Lipschitz continuous with constant $L_{\nabla f} > 0$, in $\mathcal{L}(\mathbf{x}_0, \Delta_{\max})$; that is,*

$$\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\| \leq L_{\nabla f}\|\mathbf{y} - \mathbf{x}\|, \tag{2.3}$$

for all $\mathbf{x}, \mathbf{y} \in \mathcal{L}(\mathbf{x}_0, \Delta_{\max})$. In addition, f is bounded below by f_{low} in $\mathbb{R}^n$.

A key consequence of Assumption 2.1 is the following.

Lemma 2.2 (Appendix A, [164])**.** *Suppose f satisfies Assumption 2.1. Then we have the bound*

$$|f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x})| \leq \frac{1}{2}L_{\nabla f}\|\mathbf{y} - \mathbf{x}\|^2, \tag{2.4}$$

for all $\mathbf{x}, \mathbf{y} \in \mathcal{L}(\mathbf{x}_0, \Delta_{\max})$.

Our software implementation extends (2.1) to allow for bound constraints; that is, it can solve

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}, \tag{2.5}$$

for some vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, where the inequalities hold component-wise.

Assumption 2.3. *The bounds $\mathbf{a}$ and $\mathbf{b}$ in (2.5) satisfy $\mathbf{a} < \mathbf{b}$, where this inequality holds component-wise.*

2.1.2 Nonlinear Least-Squares Minimisation

When we consider nonlinear least-squares problems, we mean that f in (2.1) may be written as the sum of squares of m residual functions. That is, we wish to solve

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) := \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{x})^2, \tag{2.6}$$

where $\mathbf{r}(\mathbf{x}) := [r_1(\mathbf{x}) \ \cdots \ r_m(\mathbf{x})]^\top$ maps $\mathbb{R}^n \to \mathbb{R}^m$. We will allow both $m \geq n$ (least-squares) and $m \leq n$ (inverse problems). If $\mathbf{r}(\mathbf{x})$ is differentiable, then we define its Jacobian matrix of first derivatives as per Section 1.1.3, namely:

$$J(\mathbf{x}) := \begin{bmatrix} \nabla r_1(\mathbf{x})^\top \\ \vdots \\ \nabla r_m(\mathbf{x})^\top \end{bmatrix} \in \mathbb{R}^{m\times n}. \tag{2.7}$$

For our theoretical analysis, our assumptions on $\mathbf{r}$ are as follows.

Assumption 2.4. *Given $\mathbf{x}_0 \in \mathbb{R}^n$ and $\Delta_{\max} > 0$, the function $\mathbf{r}$ is continuously differentiable, and its Jacobian is Lipschitz continuous with constant $L_J > 0$, in $\mathcal{L}(\mathbf{x}_0, \Delta_{\max})$; that is,*

$$\|J(\mathbf{y}) - J(\mathbf{x})\| \leq L_J\|\mathbf{y} - \mathbf{x}\|, \tag{2.8}$$

for all $\mathbf{x}, \mathbf{y} \in \mathcal{L}(\mathbf{x}_0, \Delta_{\max})$. We also assume that $\mathbf{r}(\mathbf{x})$ and $J(\mathbf{x})$ are uniformly bounded in the same region; i.e. $\|\mathbf{r}(\mathbf{x})\| \leq r_{\max}$ and $\|J(\mathbf{x})\| \leq J_{\max}$ for all $\mathbf{x} \in \mathcal{L}(\mathbf{x}_0, \Delta_{\max})$.

The uniform boundedness assumption can be achieved if the level set $\{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$ is bounded, which is assumed in [240], for instance.

Lemma 2.5. *Suppose $\mathbf{r}$ satisfies Assumption 2.4. Then $f(\mathbf{x})$ in (2.6) satisfies Assumption 2.1 with*

$$L_{\nabla f} := r_{\max}L_J + J_{\max}^2, \tag{2.9}$$

and $f_{\text{low}} = 0$. We also have the bound

$$\|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x}) - J(\mathbf{x})(\mathbf{y} - \mathbf{x})\| \leq \frac{1}{2}L_J\|\mathbf{y} - \mathbf{x}\|^2, \tag{2.10}$$

for all $\mathbf{x}, \mathbf{y} \in \mathcal{L}(\mathbf{x}_0, \Delta_{\max})$.

Proof. We have $f(\mathbf{x}) \geq 0$ by definition, and (2.10) is proved in [164, Appendix A]. To show (2.9), we choose $\mathbf{x}, \mathbf{y} \in \mathcal{L}(\mathbf{x}_0, \Delta_{\max})$ and compute (using Assumption 2.4)

$$\|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x})\| = \left\| \int_0^1 J(\mathbf{x} + \alpha(\mathbf{y} - \mathbf{x}))(\mathbf{y} - \mathbf{x}) d\alpha \right\| \leq J_{\max}\|\mathbf{y} - \mathbf{x}\|. \tag{2.11}$$

Now we use this, the identity $\|A\| = \|A^\top\|$, Assumption 2.4, and $\nabla f(\mathbf{x}) = J(\mathbf{x})^\top \mathbf{r}(\mathbf{x})$ to get

$$\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\| \leq \|(J(\mathbf{y}) - J(\mathbf{x}))^\top \mathbf{r}(\mathbf{y})\| + \|J(\mathbf{x})^\top (\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x}))\|, \tag{2.12}$$

$$\leq \|J(\mathbf{y}) - J(\mathbf{x})\| \cdot \|\mathbf{r}(\mathbf{y})\| + \|J(\mathbf{x})\| \cdot \|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x})\|, \tag{2.13}$$

$$\leq L_J \|\mathbf{y} - \mathbf{x}\| \cdot r_{\max} + J_{\max} \cdot J_{\max} \|\mathbf{y} - \mathbf{x}\|, \tag{2.14}$$

and we are done. □

Similar to the general minimisation case, our software implementation extends (2.6) to include bound constraints,

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}) := \frac{1}{2} \|\mathbf{r}(\mathbf{x})\|^2 \quad \text{s.t.} \quad \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}. \tag{2.15}$$

Assumption 2.6. *The bounds* $\mathbf{a}$ *and* $\mathbf{b}$ *in* (2.15) *satisfy* $\mathbf{a} < \mathbf{b}$.

2.2 Polynomial Interpolation Theory

The algorithms in later chapters are fundamentally based on the construction of linear or quadratic interpolation models for the objective function (or residual functions for least-squares problems).[1] In this section, we summarise the relevant theory of polynomial interpolation, which can be found in more detail in [60].

To build a convergence theory for model-based DFO, it is necessary to consider the accuracy of interpolation models, which depends on the geometry of the interpolation points. In this setting, our goal is to build models that are comparably accurate to a first-order Taylor series approximation. The below condition is satisfied by linear Taylor models and linear interpolation models (with a full set of $n + 1$ well-spaced interpolation points—see Section 2.2.1), for instance, and so we call models which satisfy this condition 'fully linear'.

Definition 2.7 (Definition 10.3, [60])**.** *A model* $m(\mathbf{s})$ *for a function* $f(\mathbf{x} + \mathbf{s})$ *satisfying Assumption 2.1 is fully linear in* $B(\mathbf{x}, \Delta)$ *if* m *is continuously differentiable and there exist constants* $\kappa_{\text{ef}}, \kappa_{\text{eg}} > 0$*, independent of* $\mathbf{x}$ *and* $\Delta > 0$*, such that*

$$|f(\mathbf{x} + \mathbf{s}) - m(\mathbf{s})| \leq \kappa_{\text{ef}} \Delta^2, \tag{2.16}$$

$$\|\nabla f(\mathbf{x} + \mathbf{s}) - \nabla m(\mathbf{s})\| \leq \kappa_{\text{eg}} \Delta, \tag{2.17}$$

for all $\|\mathbf{s}\| \leq \Delta$.

[1] Since the number of degrees of freedom for polynomials grows exponentially in the polynomial degree, we consider only linear and quadratic models.

We note that (2.3) and (2.4) imply that the first-order Taylor series $m(\mathbf{s}) := f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \mathbf{s}$ is a fully linear model for f with $\kappa_{\text{ef}} = L_{\nabla f}/2$ and $\kappa_{\text{eg}} = L_{\nabla f}$.

There is a stronger notion, satisfied by quadratic Taylor models and quadratic interpolation models (with a full set of $(n+1)(n+2)/2$ well-spaced interpolation points), called 'fully quadratic'; we discuss this briefly in Section 2.2.3.

2.2.1 Linear Interpolation

Suppose we want to construct a linear model for f, which we have evaluated at $n+1$ points $\{\mathbf{y}_0, \ldots, \mathbf{y}_n\} \subset \mathbb{R}^n$. For our algorithms, we will want to build the model based around the current iterate, which we take to be $\mathbf{y}_0$. That is, we want a local model $m : \mathbb{R}^n \to \mathbb{R}$ given by

$$f(\mathbf{y}_0 + \mathbf{s}) \approx m(\mathbf{s}) := c + \mathbf{g}^\top \mathbf{s}, \tag{2.18}$$

where $c \in \mathbb{R}$ and $\mathbf{g} \in \mathbb{R}^n$. By imposing the interpolation conditions $m(\mathbf{y}_t - \mathbf{y}_0) = f(\mathbf{y}_t)$ for $t = 0, \ldots, n$, we arrive at the $(n+1) \times (n+1)$ linear system

$$\begin{bmatrix} 1 & (\mathbf{y}_0 - \mathbf{y}_0)^\top \\ \vdots & \vdots \\ 1 & (\mathbf{y}_n - \mathbf{y}_0)^\top \end{bmatrix} \begin{bmatrix} c \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} f(\mathbf{y}_0) \\ \vdots \\ f(\mathbf{y}_n) \end{bmatrix}. \tag{2.19}$$

We write the matrix in (2.19) as

$$W := \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{e} & L \end{bmatrix}, \tag{2.20}$$

where $\mathbf{e} \in \mathbb{R}^n$ is the vector of ones and $L \in \mathbb{R}^{n \times n}$ has rows $(\mathbf{y}_t - \mathbf{y}_0)^\top$ for $t = 1, \ldots, n$.

Definition 2.8 (Definition 2.9, [60])**.** *The set $\{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ is poised for linear interpolation if W in (2.20) is invertible.*

For m to be a fully linear model for f, we need to bound $\|L^{-1}\|$; to do this we must consider the geometry of the interpolation set. We will use the Lagrange polynomials associated with the interpolation set for this purpose.

Definition 2.9 (Definition 3.3, [60])**.** *The Lagrange polynomials associated with the interpolation set $\{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ are the linear polynomials $\{\ell_0(\mathbf{x}), \ldots, \ell_n(\mathbf{x})\}$ satisfying $\ell_s(\mathbf{y}_t) = \delta_{s,t}$ for all $s, t = 0, \ldots, n$.*

Lemma 2.10 (Lemma 3.4, [60])**.** *If $\{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ is poised for linear interpolation then the associated Lagrange polynomials exist and are unique.*

Then, the required notion of 'good geometry' is the following:

Definition 2.11 (Definition 3.6, [60]). *Fix $\Lambda > 0$ and a set $B \subset \mathbb{R}^n$. A set $\{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ poised for linear interpolation is Λ-poised for linear interpolation in B if the associated Lagrange polynomials satisfy $|\ell_t(\mathbf{x})| \leq \Lambda$ for all $\mathbf{x} \in B$ and all $t = 0, \ldots, n$.*

If any $\mathbf{y}_t \in B$, which will usually be the case, then we must have $\Lambda \geq 1$. In general, a smaller value of Λ indicates 'better' geometry. We note that the concept of Λ-poisedness is closely related to the Lebesgue constant from (univariate) polynomial interpolation theory [214, Chapter 15]. Specifically, if we write the Lagrange polynomials as a vector $\boldsymbol{\ell}(\mathbf{x}) := [\ell_0(\mathbf{x}), \ldots, \ell_n(\mathbf{x})]^\top$, then the set $\{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ is Λ-poised for linear interpolation in B if $\max_{\mathbf{x} \in B} \|\boldsymbol{\ell}(\mathbf{x})\|_\infty \leq \Lambda$. By contrast, the Lebesgue constant for $\{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ is the smallest value Λ such that $\max_{\mathbf{x} \in B} \|\boldsymbol{\ell}(\mathbf{x})\|_1 \leq \Lambda$.[2] That is, the two concepts are the same up to a change in norm, and hence a change in constant.

The concept of Λ-poisedness gives us the desired result—linear interpolation yields fully linear models.

Theorem 2.12. *Suppose f is continuously differentiable and ∇f is Lipschitz continuous with constant $L_{\nabla f} > 0$ in an open ball containing $B(\mathbf{y}_0, \Delta)$. If $Y := \{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ is Λ-poised for linear interpolation in $B(\mathbf{y}_0, \Delta)$ and $Y \subset B(\mathbf{y}_0, \Delta)$, then the model m defined by (2.19) is a fully linear model for f in $B(\mathbf{y}_0, \Delta)$ with*

$$\kappa_{\text{ef}} = \frac{L_{\nabla f}}{2}\left(3 + (n+1)\Lambda\right), \qquad \text{and} \qquad \kappa_{\text{eg}} = \frac{L_{\nabla f}}{2}\left(2 + (n+1)\Lambda\right). \tag{2.21}$$

Proof. [60, Theorems 2.11 & 2.12] give full linearity with constants

$$\kappa_{\text{ef}} = \frac{L_{\nabla f}}{2}\left(3 + \sqrt{n}\|\hat{L}^{-1}\|\right), \qquad \text{and} \qquad \kappa_{\text{eg}} = \frac{L_{\nabla f}}{2}\left(2 + \sqrt{n}\|\hat{L}^{-1}\|\right), \tag{2.22}$$

where $\hat{L} := L/\Delta$ and L is the sub-matrix from (2.20). Separately, [60, Theorem 3.14] gives

$$\|\hat{W}^{-1}\| := \left\| \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{e} & \hat{L} \end{bmatrix}^{-1} \right\| \leq \sqrt{n+1}\,\Lambda. \tag{2.23}$$

The result follows since $\|\hat{L}^{-1}\| \leq \|\hat{W}^{-1}\|$, which can be checked by observing

$$\hat{W}^{-1} = \begin{bmatrix} 1 & \mathbf{0}^\top \\ -\hat{L}^{-1}\mathbf{e} & \hat{L}^{-1} \end{bmatrix}, \tag{2.24}$$

and considering a unit vector $\mathbf{u}$ for which $\|\hat{L}^{-1}\mathbf{u}\| = \|\hat{L}^{-1}\|$. □

We discuss how to ensure Λ-poisedness of an interpolation set in Section 2.3.

[2] The Lebesgue constant is defined as the maximal value—at any point—of an interpolating polynomial to a function bounded by 1. For univariate interpolation, this definition can be interpreted in terms of Lagrange polynomials, as above.

2.2.2 Linear Regression

In our algorithm for nonlinear least-squares problems (2.6), we will construct linear models for the residual functions, but with greater flexibility than strict interpolation by allowing the use of regression models. Compared to Section 2.2.1, we now want to construct a linear model (2.18) for f, having evaluated it at $p+1$ points $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\} \subset \mathbb{R}^n$ for some $p \geq n$. The theoretical analysis and implementation of our algorithm for (2.6) allows for interpolation or regression models, however our default implementation uses interpolation models. We will compare the use of interpolation and regression models for (2.6) in Chapter 5, however we note the theoretical work [51], which considers the benefit of regression compared to interpolation for general function approximation problems (not specific to linear or polynomial approximation).

With $p+1$ points, the model (2.18) is defined by the overdetermined $(p+1) \times (n+1)$ linear system, solved in the least-squares sense:

$$W \begin{bmatrix} c \\ \mathbf{g} \end{bmatrix} := \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{e} & L \end{bmatrix} \begin{bmatrix} c \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} 1 & (\mathbf{y}_0 - \mathbf{y}_0)^\top \\ \vdots & \vdots \\ 1 & (\mathbf{y}_p - \mathbf{y}_0)^\top \end{bmatrix} \begin{bmatrix} c \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} f(\mathbf{y}_0) \\ \vdots \\ f(\mathbf{y}_p) \end{bmatrix}, \tag{2.25}$$

where $\mathbf{e} \in \mathbb{R}^p$ is the vector of ones and $L \in \mathbb{R}^{p \times n}$ has rows $(\mathbf{y}_t - \mathbf{y}_0)^\top$ for $t = 1, \ldots, p$ (c.f. (2.19)).

Definition 2.13 (Definition 2.10, [60])**.** *The set $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ with $p \geq n$ is poised for linear regression if W (2.25) has full column rank.*

We then have to adjust our definition of Lagrange polynomials and Λ-poisedness appropriately for this context.

Definition 2.14 (Definition 4.4, [60])**.** *The Lagrange polynomials associated with the regression set $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ with $p \geq n$ are the linear polynomials $\{\ell_0(\mathbf{x}), \ldots, \ell_p(\mathbf{x})\}$ satisfying $\ell_s(\mathbf{y}_t) = \delta_{s,t}$ for all $s, t = 0, \ldots, p$, where the overdetermined interpolation system for ℓ_s is solved in the least-squares sense.*

Lemma 2.15 (Lemma 4.5, [60])**.** *If $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ is poised for linear regression then the associated Lagrange polynomials exist and are unique.*

Definition 2.16 (Definition 4.7, [60])**.** *Fix $\Lambda > 0$ and a set $B \subset \mathbb{R}^n$. A set $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ poised for linear regression is Λ-poised for linear regression in B if the associated Lagrange polynomials satisfy $|\ell_t(\mathbf{x})| \leq \Lambda$ for all $\mathbf{x} \in B$ and all $t = 0, \ldots, p$.*

Although this definition matches Definition 2.11, we can no longer guarantee $\Lambda \geq 1$ if some $\mathbf{y}_t \in B$.[3] However, with this definition we arrive at the regression version of Theorem 2.12; note that the two results are identical if $p = n$.

Theorem 2.17. *Suppose f is continuously differentiable and ∇f is Lipschitz continuous with constant $L_{\nabla f} > 0$ in an open ball containing $B(\mathbf{y}_0, \Delta)$. If $Y := \{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ is Λ-poised for linear regression in $B(\mathbf{y}_0, \Delta)$ and $Y \subset B(\mathbf{y}_0, \Delta)$, then the model m given by (2.25) is a fully linear model for f in $B(\mathbf{y}_0, \Delta)$ with*

$$\kappa_{\text{ef}} = \frac{L_{\nabla f}}{2}\left(3 + (p+1)\Lambda\right), \qquad \textit{and} \qquad \kappa_{\text{eg}} = \frac{L_{\nabla f}}{2}\left(2 + (p+1)\Lambda\right). \tag{2.26}$$

Proof. [60, Theorem 2.13] gives full linearity with constants

$$\kappa_{\text{ef}} = \frac{L_{\nabla f}}{2}\left(3 + \sqrt{p}\|\hat{L}^\dagger\|\right), \qquad \text{and} \qquad \kappa_{\text{eg}} = \frac{L_{\nabla f}}{2}\left(2 + \sqrt{p}\|\hat{L}^\dagger\|\right), \tag{2.27}$$

where $\hat{L} := L/\Delta$ and L is the sub-matrix from (2.25). Separately, [58, Theorem 2.9] gives

$$\|\hat{W}^\dagger\| := \left\| \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{e} & \hat{L} \end{bmatrix}^\dagger \right\| \leq \sqrt{p+1}\,\Lambda. \tag{2.28}$$

The result follows since $\|\hat{L}^\dagger\| \leq \|\hat{W}^\dagger\|$, which can be checked as follows: let $\mathbf{u}^* \in \mathbb{R}^p$ be the n-th left singular vector of $\hat{L}$. Then since $\hat{L}$ has rank n, $\|\hat{L}^\dagger\| = \|\hat{L}^\dagger \mathbf{u}^*\|$ and $\mathbf{u}^*$ is in the range of $\hat{L}$; i.e. $\mathbf{u}^* = \hat{L}\mathbf{v}^*$ for some $\mathbf{v}^* \in \mathbb{R}^n$.[4] Now consider the linear least-squares problem

$$\min_{c \in \mathbb{R}, \mathbf{v} \in \mathbb{R}^n} \left\| \hat{W} \begin{bmatrix} c \\ \mathbf{v} \end{bmatrix} - \begin{bmatrix} 0 \\ \mathbf{u}^* \end{bmatrix} \right\|^2 = \left\| \begin{bmatrix} c \\ c\,\mathbf{e} + \hat{L}\mathbf{v} - \mathbf{u}^* \end{bmatrix} \right\|^2. \tag{2.29}$$

The solution to this problem is $c = 0$ and $\mathbf{v} = \mathbf{v}^*$, since this yields a zero objective value (and is unique since $\hat{W}$ has full column rank). Alternatively, we may write the solution as $\hat{W}^\dagger \begin{bmatrix} 0 \\ \mathbf{u}^* \end{bmatrix}$, by the properties of the pseudoinverse. Hence we have

$$\|\hat{L}^\dagger\| = \|\hat{L}^\dagger \mathbf{u}^*\| \leq \|\hat{L}^\dagger \hat{L}\|\|\mathbf{v}^*\| \leq \|\mathbf{v}^*\| = \left\| \begin{bmatrix} 0 \\ \mathbf{v}^* \end{bmatrix} \right\| = \left\| \hat{W}^\dagger \begin{bmatrix} 0 \\ \mathbf{u}^* \end{bmatrix} \right\| \leq \|\hat{W}^\dagger\|, \tag{2.30}$$

since $\hat{L}^\dagger \hat{L}$ is an orthogonal projector and $\|\mathbf{u}^*\| = 1$. □

We conclude with the following result, which we will use later.

[3] For example (with $n = 1$), the set $\{0, 1, -1, 2\}$ is Λ-poised for linear regression in $B(0, 1) = [-1, 1]$ with $\Lambda = 0.7$.

[4] Specifically, $\mathbf{v}^* = \sigma_n \mathbf{v}_n$ where σ_n and $\mathbf{v}_n$ are the n-th singular value and right singular vector of $\hat{L}$ respectively.

Lemma 2.18 (Section 4.2, [60]). *If $Y := \{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ is poised for linear regression, then the associated Lagrange polynomials $\{\ell_0, \ldots, \ell_p\}$ yield the minimal-norm solution to the underdetermined $(n+1) \times (p+1)$ linear system*

$$W^\top \begin{bmatrix} \ell_0(\mathbf{x}) \\ \vdots \\ \ell_p(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} - \mathbf{y}_0 \end{bmatrix}, \tag{2.31}$$

for any $\mathbf{x} \in \mathbb{R}^n$, where W is the matrix defined in (2.25).

Proof. The Lagrange polynomials are linear, and so can be written as $\ell_t(\mathbf{x}) := c_t + \mathbf{g}_t^\top(\mathbf{x} - \mathbf{y}_0)$. By definition of Lagrange polynomials, we can write these coefficients as

$$\begin{bmatrix} c_t \\ \mathbf{g}_t \end{bmatrix} = W^\dagger \mathbf{e}_t, \tag{2.32}$$

for $t = 0, \ldots, p$, where $\mathbf{e}_t$ is the t-th coordinate vector, and so

$$\ell_t(\mathbf{x}) = \begin{bmatrix} c_t \\ \mathbf{g}_t \end{bmatrix}^\top \begin{bmatrix} 1 \\ \mathbf{x} - \mathbf{y}_0 \end{bmatrix} = \left[(W^\dagger)^\top \begin{bmatrix} 1 \\ \mathbf{x} - \mathbf{y}_0 \end{bmatrix} \right]_t. \tag{2.33}$$

That is,

$$\begin{bmatrix} \ell_0(\mathbf{x}) \\ \vdots \\ \ell_p(\mathbf{x}) \end{bmatrix} = (W^\dagger)^\top \begin{bmatrix} 1 \\ \mathbf{x} - \mathbf{y}_0 \end{bmatrix}, \tag{2.34}$$

and we are done, since $(W^\dagger)^\top = (W^\top)^\dagger$. □

2.2.3 Quadratic Interpolation

When we move from solving nonlinear least-squares to general objective problems, we will need to switch from linear to quadratic interpolation models to capture curvature in the objective. To do this in the simplest way, we would need $(n+1)(n+2)/2$ interpolation points to fully determine a quadratic function in $\mathbb{R}^n$. However, in DFO it is common to use $\mathcal{O}(n)$ interpolation points, which requires fewer initial evaluations of the objective and reduces the linear algebra cost of constructing the model at each iteration, without substantially impacting the performance of the method.

There are a number of ways to formulate the corresponding underdetermined interpolation problem, but here we follow the approach of Powell [182], and suppose we have evaluated f at $p+1$ points $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ with $n+2 \leq p+1 \leq (n+1)(n+2)/2$. Then we wish to find the model (based around $\mathbf{y}_0$)

$$f(\mathbf{y}_0 + \mathbf{s}) \approx m(\mathbf{s}) := c + \mathbf{g}^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H \mathbf{s}, \tag{2.35}$$

for $c \in \mathbb{R}$, $\mathbf{g} \in \mathbb{R}^n$ and $H \in \mathbb{R}^{n \times n}$, by solving

$$\begin{aligned} \min_{c,\mathbf{g},H} \quad & \frac{1}{4}\|H - H_{\text{prev}}\|_F^2, && \text{(2.36a)} \\ \text{subject to} \quad & m(\mathbf{y}_t - \mathbf{y}_0) = f(\mathbf{y}_t), \quad \forall t = 0, \dots, p, && \text{(2.36b)} \\ & H = H^\top, && \text{(2.36c)} \end{aligned}$$

where $H_{\text{prev}} = H_{\text{prev}}^\top$ is the Hessian of the previous interpolation model (or zero, in the first iteration). The solution to (2.36) is given by the $(p+n+2) \times (p+n+2)$ linear system

$$\begin{bmatrix} \frac{1}{2}(\mathbf{s}_0^\top \mathbf{s}_0)^2 & \cdots & \frac{1}{2}(\mathbf{s}_0^\top \mathbf{s}_p)^2 & 1 & \mathbf{s}_0^\top \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \frac{1}{2}(\mathbf{s}_p^\top \mathbf{s}_0)^2 & \cdots & \frac{1}{2}(\mathbf{s}_p^\top \mathbf{s}_p)^2 & 1 & \mathbf{s}_p^\top \\ 1 & \cdots & 1 & 0 & \mathbf{0} \\ \mathbf{s}_0 & \cdots & \mathbf{s}_p & \mathbf{0} & 0_{n \times n} \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \vdots \\ \lambda_p \\ c \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} f(\mathbf{y}_0) - \frac{1}{2}\mathbf{s}_0^\top H_{\text{prev}} \mathbf{s}_0 \\ \vdots \\ f(\mathbf{y}_p) - \frac{1}{2}\mathbf{s}_p^\top H_{\text{prev}} \mathbf{s}_p \\ 0 \\ \mathbf{0} \end{bmatrix}, \tag{2.37}$$

where we define $\mathbf{s}_t := \mathbf{y}_t - \mathbf{y}_0$ for convenience, and the full model Hessian is $H = H_{\text{prev}} + \sum_{t=0}^{p} \lambda_t (\mathbf{y}_t - \mathbf{y}_0)(\mathbf{y}_t - \mathbf{y}_0)^\top$. The system (2.37) corresponds to the KKT conditions for (2.36) excluding the $H = H^\top$ constraint, with the symmetry of H implicitly induced by this solution; the values λ_t are the Lagrange multipliers for (2.36b). Since (2.36) is convex in c and $\mathbf{g}$ and strictly convex in H, we always have a unique H, and c and $\mathbf{g}$ are unique provided $\{\mathbf{y}_0, \dots, \mathbf{y}_p\}$ is poised for linear regression [183]. The matrix in (2.37) symmetric, and the top-left quadratic block (of size $p+1$) is positive semidefinite.

Definition 2.19 (Section 5.3, [60])**.** *The set $\{\mathbf{y}_0, \dots, \mathbf{y}_p\}$ is poised for minimum Frobenius norm quadratic interpolation if the linear system in* (2.37) *is invertible.*

Definition 2.20 (Definition 5.5, [60])**.** *The minimum Frobenius norm Lagrange polynomials associated with the interpolation set $\{\mathbf{y}_0, \dots, \mathbf{y}_p\}$ with $p \geq n$ are the quadratic polynomials $\{\ell_0(\mathbf{x}), \dots, \ell_p(\mathbf{x})\}$ satisfying* (2.36) *with the interpolation conditions $\ell_s(\mathbf{y}_t) = \delta_{s,t}$ for all $s, t = 0, \dots, p$, and $H_{\text{prev}} = 0$.*

Definition 2.21 (Definition 5.6, [60])**.** *Fix $\Lambda > 0$ and a set $B \subset \mathbb{R}^n$. A set $\{\mathbf{y}_0, \dots, \mathbf{y}_p\}$ poised for minimum Frobenius norm quadratic interpolation is Λ-poised for minimum Frobenius norm quadratic interpolation in B if the associated Lagrange polynomials satisfy $|\ell_t(\mathbf{x})| \leq \Lambda$ for all $\mathbf{x} \in B$ and all $t = 0, \dots, p$.*

As in the linear case, Λ-poisedness implies that the interpolation model (2.35) is a fully linear model for f in the sense of Definition 2.7, provided we can control the size of the model Hessian. To show this, we need to restrict ourselves to the case $H_{\text{prev}} = 0$ [60].

Lemma 2.22. *Suppose $Y := \{\mathbf{y}_0, \dots, \mathbf{y}_p\}$ is Λ-poised for minimum Frobenius norm quadratic interpolation in $B(\mathbf{y}_0, \Delta)$ with $H_{\text{prev}} = 0$. Then Y is $(\sqrt{p+1}\,\Lambda)$-poised for linear regression in $B(\mathbf{y}_0, \Delta)$.*

Proof. From the definition of Lagrange polynomials we have, for all $\mathbf{x}$,

$$m(\mathbf{x} - \mathbf{y}_0) = \sum_{t=0}^{p} f(\mathbf{y}_t)\ell_t(\mathbf{x}). \tag{2.38}$$

However, minimum Frobenius norm quadratic interpolation using Y exactly interpolates linear functions, since $H = 0$ yields a global minimum for (2.36), and so for f linear, we also have $f(\mathbf{x}) = m(\mathbf{x} - \mathbf{y}_0)$. By taking $f \equiv 1$ and $f(\mathbf{x}) = x_i$, we get

$$\sum_{t=0}^{p} \ell_t(\mathbf{x}) = 1, \qquad \text{and} \qquad \sum_{t=0}^{p} y_{t,i}\ell_t(\mathbf{x}) = x_i, \tag{2.39}$$

which we can combine to get $\sum_{t=0}^{p}(y_{t,i} - y_{0,i})\ell_t(\mathbf{x}) = x_i - y_{0,i}$, and so

$$W^\top \begin{bmatrix} \ell_0(\mathbf{x}) \\ \vdots \\ \ell_p(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} - \mathbf{y}_0 \end{bmatrix}, \tag{2.40}$$

where W is the matrix from (2.25). Thus by defining $\boldsymbol{\ell}(\mathbf{x}) := [\ell_0(\mathbf{x}), \ldots, \ell_p(\mathbf{x})]^\top$ and denoting $\boldsymbol{\ell}^{\text{quad}}$ for minimum Frobenius norm quadratic interpolation and $\boldsymbol{\ell}^{\text{reg}}$ for linear regression, we have $\|\boldsymbol{\ell}^{\text{reg}}(\mathbf{x})\| \leq \|\boldsymbol{\ell}^{\text{quad}}(\mathbf{x})\|$ from Lemma 2.18, and so

$$\|\boldsymbol{\ell}^{\text{reg}}(\mathbf{x})\|_\infty \leq \|\boldsymbol{\ell}^{\text{reg}}(\mathbf{x})\| \leq \|\boldsymbol{\ell}^{\text{quad}}(\mathbf{x})\| \leq \sqrt{p+1}\|\boldsymbol{\ell}^{\text{quad}}(\mathbf{x})\|_\infty \leq \sqrt{p+1}\,\Lambda, \tag{2.41}$$

as required. □

Theorem 2.23. *Suppose f is continuously differentiable and ∇f is Lipschitz continuous with constant $L_{\nabla f} > 0$ in an open ball containing $B(\mathbf{y}_0, \Delta)$. If $Y := \{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ is Λ-poised for minimum Frobenius norm quadratic interpolation in $B(\mathbf{y}_0, \Delta)$ with $H_{\text{prev}} = 0$, and $Y \subset B(\mathbf{y}_0, \Delta)$, then the model m given by (2.36) is a fully linear model for f in $B(\mathbf{y}_0, \Delta)$ with*

$$\kappa_{\text{ef}} = \left(\frac{5}{2}(p+1)^{3/2}\Lambda + \frac{1}{2}\right)(L_{\nabla f} + \kappa_H), \quad \textit{and} \quad \kappa_{\text{eg}} = \frac{5}{2}(p+1)^{3/2}\Lambda(L_{\nabla f} + \kappa_H), \tag{2.42}$$

where κ_H is an upper bound for $\|H\|$.

Proof. From [60, Theorem 5.4], we have

$$\kappa_{\text{ef}} = \left(\frac{5}{2}\sqrt{p}\|\hat{W}^\dagger\| + \frac{1}{2}\right)(L_{\nabla f} + \kappa_H), \quad \text{and} \quad \kappa_{\text{eg}} = \frac{5}{2}\sqrt{p}\|\hat{W}^\dagger\|(L_{\nabla f} + \kappa_H), \tag{2.43}$$

where $\hat{W}$ is the same matrix as in the proof of Theorem 2.17. The result then follows from Lemma 2.22 and (2.28), which together imply that $\|\hat{W}^\dagger\| \leq (p+1)\Lambda$. □

If we continue to assume $H_{\text{prev}} = 0$, then we also get the bound

$$\|H\| \leq \frac{4(p+1)\sqrt{q}\,L_{\nabla f}\Lambda}{\min\left\{1, \Delta_{\max}^{-1}, \Delta_{\max}^{-2}\right\}}, \tag{2.44}$$

where $\Delta_{\max}$ is an upper bound on Δ and $q := (n+1)(n+2)/2$ [60, Theorem 5.7].

Quadratic Models in BOBYQA In BOBYQA [186], quadratic models are constructed by solving (2.36) at each iteration (with $H_{\text{prev}} = 0$ in the first iteration). The linear algebra cost associated with solving (2.37) is reduced by noting that changing a single interpolation point yields a rank-2 update of the matrix in (2.37).[5] Hence, BOBYQA only stores the inverse of this matrix, and updates it at each iteration via the Sherman-Morrison-Woodbury formula. Full details of this update are given in [182, Section 4].

Fully Quadratic Models In the case where $p+1 = (n+1)(n+2)/2$, then the interpolation conditions (2.36b) define a unique quadratic model, and we do not need to select the minimal-norm solution (2.36a). Hence the solution to (2.36)—regardless of the choice of H_{prev}—can be found by solving the $(p+1) \times (p+1)$ square system

$$\begin{bmatrix} 1 & \mathbf{s}_0^\top & s_{0,1}^2/2 & \cdots & s_{0,n}^2/2 & s_{0,1}s_{0,2} & \cdots & s_{0,n-1}s_{0,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathbf{s}_p^\top & s_{p,1}^2/2 & \cdots & s_{p,n}^2/2 & s_{p,1}s_{p,2} & \cdots & s_{p,n-1}s_{p,n} \end{bmatrix} \begin{bmatrix} c \\ \mathbf{g} \\ H_{1,1} \\ \vdots \\ H_{n,n} \\ H_{1,2} \\ \vdots \\ H_{n-1,n} \end{bmatrix} = \begin{bmatrix} f(\mathbf{y}_0) \\ \vdots \\ f(\mathbf{y}_p) \end{bmatrix}, \tag{2.45}$$

where again $\mathbf{s}_t := \mathbf{y}_t - \mathbf{y}_0$ for convenience, instead of the larger $(p+n+2) \times (p+n+2)$ system (2.37).

In this case, we get a fully linear model by Theorem 2.23, but we actually get a stronger condition, which allows us to prove second-order convergence results (e.g. [60, Chapter 10.5]).

Definition 2.24 (Definition 10.4, [60])**.** *A model $m(\mathbf{s})$ for a twice continuously-differentiable function $f(\mathbf{x}+\mathbf{s})$ is fully quadratic in $B(\mathbf{x}, \Delta)$ if m is twice continuously differentiable and there exist constants $\kappa_{\text{ef}}, \kappa_{\text{eg}}, \kappa_{\text{eh}} > 0$, independent of $\mathbf{x}$ and $\Delta > 0$, so that*

$$|f(\mathbf{x}+\mathbf{s}) - m(\mathbf{s})| \leq \kappa_{\text{ef}}\Delta^3, \tag{2.46}$$

$$\|\nabla f(\mathbf{x}+\mathbf{s}) - \nabla m(\mathbf{s})\| \leq \kappa_{\text{eg}}\Delta^2, \tag{2.47}$$

$$\|\nabla^2 f(\mathbf{x}+\mathbf{s}) - \nabla^2 m(\mathbf{s})\| \leq \kappa_{\text{eh}}\Delta, \tag{2.48}$$

for all $\|\mathbf{s}\| \leq \Delta$.

[5] As formulated here, this applies when changing any point except $\mathbf{y}_0$. The convention in [182] is to base the model at a point not necessarily in the interpolation set; i.e. $f(\mathbf{y}_b + \mathbf{s}) \approx m(\mathbf{s})$ and $\mathbf{s}_t := \mathbf{y}_t - \mathbf{y}_b$. The more complicated update to the matrix inverse from changing $\mathbf{y}_b$ is then done less frequently. Full details on this process are given in [182].

Theorem 2.25. *Suppose f is twice continuously differentiable and $\nabla^2 f$ is Lipschitz continuous with constant $L_H > 0$ in an open ball containing $B(\mathbf{y}_0, \Delta)$. If $Y := \{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ with $p = (n+1)(n+2)/2 - 1$ is Λ-poised for quadratic interpolation in $B(\mathbf{y}_0, \Delta)$ and $Y \subset B(\mathbf{y}_0, \Delta)$, then the model m given by (2.36) is a fully quadratic model for f in $B(\mathbf{y}_0, \Delta)$ with*

$$\kappa_{\text{ef}} = 2(6 + 9\sqrt{2})L_H(p+1)^{3/2}\Lambda + \frac{1}{6}L_H, \tag{2.49}$$

$$\kappa_{\text{eg}} = 6(1 + \sqrt{2})L_H(p+1)^{3/2}\Lambda, \tag{2.50}$$

$$\kappa_{\text{eh}} = 6\sqrt{2}\, L_H(p+1)^{3/2}\Lambda. \tag{2.51}$$

Proof. Similar to the proof of Theorem 2.12, [60, Theorem 3.16] gives constants in terms of the norm of the inverse of the matrix from (2.45) (scaled by Δ), and [60, Theorem 3.14] bounds this norm in terms of Λ. □

2.3 Poisedness-Improving Methods

In Section 2.2 we introduced the notion of Λ-poisedness for interpolation and regression sets, and proved that this property yields fully linear models; this will be enough to ensure convergence of our algorithm. We now outline the algorithms needed to ensure the Λ-poisedness of an interpolation/regression set, which will be called several times in our optimisation procedure. More details of this material can be found in [60].

Certifying Λ-poisedness The first procedure we will need is a method to determine if an interpolation or regression set is Λ-poised. This is straightforward; the previous sections show how each Lagrange polynomial can be constructed by solving a linear system. We then need to solve

$$\max_{\mathbf{x} \in B} |\ell_t(\mathbf{x})|, \tag{2.52}$$

where in our case B will be a Euclidean ball. For linear interpolation/regression, where ℓ_t is linear, this problem can be solved analytically; for quadratic interpolation, this reduces to solving two trust-region subproblems. We note that we only need to maximise $|\ell_t|$ to sufficient accuracy to determine if the maximum is above or below Λ, not necessarily to full global optimality.

Ensuring Λ-poisedness Next, we will need a method to take a set that is not Λ-poised, and improve its geometry so that it becomes Λ-poised. We only need to consider methods to ensure Λ-poisedness of interpolation sets, due to the following result:

Lemma 2.26 (p. 63, [60])**.** *If $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ with $p > n$ contains a subset of size $n+1$ that is Λ-poised for linear interpolation, then the whole set is $(\sqrt{n+1}\,\Lambda)$-poised for linear regression.*

Algorithm 2.1 Algorithm for ensuring Λ-poisedness [60, Algorithm 6.3].

Input: Interpolation set $Y_0 \subset \mathbb{R}^n$, poisedness constant $\Lambda > 1$, set of interest $B \subset \mathbb{R}^n$.

for $k = 0, 1, 2, \ldots$ **do**
  if Y_k is Λ-poised for interpolation in B **then**
    return Y_k and **terminate**
  else
    Find i_k such that $\max_{\mathbf{y} \in B} |\ell_{i_k}(\mathbf{y})| > \Lambda$, and let $\mathbf{y}_k^*$ be the associated maximiser.
    Set $Y_{k+1} := Y_k \cup \{\mathbf{y}_k^*\} \setminus \{\mathbf{y}_{i_k}\}$.
  end if
end for

Proof. Suppose we perform linear interpolation with just the set Λ-poised for interpolation. Since these points allow us to exactly interpolate linear functions, by the same argument as in the proof of Lemma 2.22, we have

$$W^\top \begin{bmatrix} \ell_0(\mathbf{x}) \\ \vdots \\ \ell_n(\mathbf{x}) \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} - \mathbf{y}_0 \end{bmatrix}, \tag{2.53}$$

where the system is of size $(n+1) \times (p+1)$. We conclude with the same reasoning as used to reach (2.41), where we instead use $\|[\ell_0(\mathbf{x}) \ \cdots \ \ell_n(\mathbf{x})]^\top\|_\infty \leq \Lambda$. □

An algorithm for making an interpolation set Λ-poised, which applies to both linear interpolation (2.19) and fully quadratic interpolation (2.45), is given in Algorithm 2.1. As seen from the proof of Theorem 2.23, ensuring Λ-poisedness for minimum Frobenius norm quadratic interpolation is achieved by ensuring Λ-poisedness for linear regression.[6]

Lemma 2.27 (Theorem 6.3, [60])**.** *For any $\Lambda > 1$ and closed ball B, Algorithm 2.1 is guaranteed to produce a set that is Λ-poised for interpolation in B in at most a (finite) number of iterations that is independent of Y_0 and B, but proportional to $1/\log \Lambda$.*

It will be relevant for our analysis that Algorithm 2.1 does not require any evaluations of the objective function. However, after this process has finished, we will usually need to evaluate the objective at all the new interpolation points, but the number of required evaluations is bounded by the size of the interpolation set (regardless of how many iterations of Algorithm 2.1 were needed).

Criticality Phase In our algorithm, we will need a second geometry-improving procedure, the 'criticality phase'. This is a procedure, called when we believe we are close to a stationary point, which modifies the interpolation points and trust region radius to ensure our interpolation set is Λ-poised *and* that the trust region radius is not too large compared

[6] This also guarantees (2.36) has a unique solution, as per the discussion in Section 2.2.3.

Algorithm 2.2 Geometry-Improvement for Criticality Phase [60, Algorithm 10.2].

Input: Iterate $\mathbf{x}$, initial set Y^{init}, trust region radius $\Delta^{\text{init}} > 0$ and poisedness constant $\Lambda > 0$.
Parameters: $\mu > 0$ and $\omega_C \in (0, 1)$.

1: Set $Y_0 = Y^{\text{init}}$.
2: **for** $i = 1, 2, \ldots$ **do**
3: Form Y_i by modifying Y_{i-1} until it is Λ-poised in $B(\mathbf{x}, \omega_C^{i-1}\Delta^{\text{init}})$.
4: Solve the interpolation system for Y_i and form the corresponding fully linear model m_i.
5: **if** $\omega_C^{i-1}\Delta^{\text{init}} \leq \mu\|\nabla m_i(\mathbf{0})\|$ **then**
6: **return** $Y^* := Y_i$ and $\Delta^* := \omega_C^{i-1}\Delta^{\text{init}}$.
7: **end if**
8: **end for**

to the resulting model gradient. This is more complex than simply ensuring Λ-poisedness, since we have a coupling between the trust region radius and the interpolation set: ensuring poisedness inside the trust region depends on its radius, but the model gradient depends on the interpolation set.

Specifically, we need a procedure that, given an interpolation/regression set Y^{init} and trust region $B(\mathbf{x}, \Delta^{\text{init}})$, produces a new set Y^* and trust region radius $\Delta^* \leq \Delta^{\text{init}}$ such that Y^* is Λ-poised in the new trust region $B(\mathbf{x}, \Delta^*)$ and $\Delta^* \leq \mu\|\nabla m(\mathbf{0})\|$, where m is the (fully linear) model[7] corresponding to Y^* and $\mu > 0$ is a parameter. A procedure to achieve this is Algorithm 2.2.

Lemma 2.28 (Lemma 10.5, [60])**.** *If $\nabla f(\mathbf{x}) \neq 0$, then Algorithm 2.2 terminates in a finite number of iterations.*

In Chapter 3, we will need a stronger result than Lemma 2.28 to provide a worst-case complexity analysis of our algorithm. Specifically, in Lemma 4.7, we show that

$$\min\left(\Delta^{\text{init}}, \text{const} \cdot \|\nabla f(\mathbf{x})\|\right) \leq \Delta^* \leq \Delta^{\text{init}}. \tag{2.54}$$

If Algorithm 2.2 terminates in one iteration, then $\Delta^* = \Delta^{\text{init}}$, and we have simply made Y^{init} Λ-poised, just as in Algorithm 2.1. Otherwise, we do one model-improving iteration, then several iterations where both Δ is reduced and Y is made Λ-poised. The bound (2.54) tells us that these Δ-shrinking iterations do not occur when Δ^{init} is sufficiently small compared to $\|\nabla f(\mathbf{x})\|$.[8]

[7] We have shown in Section 2.2 that Λ-poised interpolation sets can produce fully linear models. For the nonlinear least-squares problem, we show that interpolation sets that are Λ-poised for linear interpolation produce fully linear *quadratic* models for the objective (see Section 4.1).

[8] The more common approach in the criticality phase (e.g. [60, 83, 240]) is to use an extra parameter $0 < \beta < \mu$ and floor Δ^* at $\beta\|\nabla m(\mathbf{0})\|$, maintaining full linearity with extra assumptions on κ_{ef} and κ_{eg} [59, Lemma 3.2], and requiring all fully linear models have Lipschitz continuous gradient with uniformly bounded Lipschitz constant.

2.4 Methodology for Numerical Tests

As described in Chapter 1, the goal of this thesis is to develop DFO algorithms that are useful for many problems, rather than tailoring our approach to solving one specific problem effectively. Our testing methodology should align with this goal. To that end, the approach we will take is to define collections of test problems (where the true solutions are known), and compare the performance of algorithms by measuring the proportion of problems solved by each, within a given budget.

Specifically, our testing methodology is based on that of Moré and Wild [157]. There are two key features of DFO that underlie this approach:

1. DFO is often applied to problems with expensive objectives, and so the number of objective evaluations required by a solver is the key driver of solver runtime; and

2. Since derivatives are not available, we cannot measure termination using the standard stationarity measure $\|\nabla f(\mathbf{x}_k)\|$, and instead look for a solver to achieve a sufficient reduction in the objective.

To compare solvers, we use data and performance profiles. First, for each solver $\mathcal{S}$, each problem P and for an accuracy level $\tau \in (0, 1)$, we determine the number of function evaluations $N_P(\mathcal{S}; \tau)$ required for a problem to be 'solved':

$$N_P(\mathcal{S}; \tau) := \#\text{ objective evals required to get } f(\mathbf{x}_k) \leq f^* + \tau(f(\mathbf{x}_0) - f^*), \tag{2.55}$$

where f^* is an estimate of the minimum value $f(\mathbf{x}^*)$.[9] We define $N_P(\mathcal{S}; \tau) = \infty$ if this was not achieved in the maximum computational budget allowed. We will typically use $\tau = 10^{-1}$ to measure low accuracy performance, and $\tau = 10^{-5}$ for high accuracy performance.

We can then compare solvers by looking at the proportion of problems solved for a given computational budget. For *data profiles*, we normalise the computational effort by problem dimension, and plot (for solver $\mathcal{S}$, accuracy level $\tau \in (0, 1)$ and problem collection $\mathcal{P}$)

$$d_{\mathcal{S},\tau}(\alpha) := \frac{|\{P \in \mathcal{P} : N_P(\mathcal{S}; \tau) \leq \alpha(n_P + 1)\}|}{|\mathcal{P}|}, \qquad \text{for } \alpha \in [0, N_g], \tag{2.56}$$

where n_P is the dimension of problem P, and N_g is the maximum computational budget in simplex gradients (i.e. we allow $N_g(n_P + 1)$ objective evaluations for problem P).

[9] Note that in [157], and subsequent other papers such as [240], the value f^* is taken to be the smallest objective value achieved by any of the solvers under consideration within a fixed budget. However, to look at the objective performance—did we achieve a sufficient reduction in the objective?—we choose to measure progress to the minimum.

For *performance profiles* [73], we normalise the computational effort by the minimum effort needed by any solver (i.e. by problem difficulty). That is, we plot

$$\pi_{\mathcal{S},\tau}(\alpha) := \frac{|\{P \in \mathcal{P} : N_p(\mathcal{S};\tau) \leq \alpha N_P^*(\tau)\}|}{|\mathcal{P}|}, \qquad \text{for } \alpha \geq 1, \tag{2.57}$$

where $N_P^*(\tau) := \min_{\mathcal{S}} N_P(\mathcal{S};\tau)$ is the minimum budget required by any solver to solve problem P.

2.4.1 Collections of Test Problems

In this thesis, we will test using five collections of problems, which are listed in full in Appendix E. Three collections have a nonlinear least-squares structure (2.15), while two have only the more general form (2.5).[10] The collections are:

(MW) The set of 53 nonlinear least-squares problems from Moré and Wild [157]. The problems are low-dimensional, with $2 \leq n \leq 12$ and $n \leq m \leq 65$, so this collection is used as the main test set for the 'noisy' regime (see noise models below);

(CR) The set of 60 nonlinear least-squares problems listed in Appendix E.2, available via the CUTEst package [86]. The problems are medium-sized, with $25 \leq n \leq 120$ and $n \leq m \leq 400$, so this collection is used as the main test set for the 'expensive' regime;

(CFMR) The set of 60 nonlinear least-squares problems (CR) and the 30 general-objective problems from CUTEst listed in Appendix E.3. These extra problems are also medium-sized, with $50 \leq n \leq 110$;

(AKZ) The collection of 50 general-objective global optimisation test problems from Ali, Khompatraporn, and Zabinsky [2] listed in Appendix E.4. To make the collection more difficult, we increased the dimension of 7 problems to $n \in [25, 50]$, where they had $n = 10$ in [2]; and

(CR-large) A subset of 28 nonlinear least-squares problems from (CR), with their dimension increased to $1000 \leq n \leq 5000$ and $n \leq m \leq 9998$, listed in Appendix E.5. This collection is used to test the scalability of our least-squares solvers in Chapter 7.

We will use (MW) and (CR) are used to test nonlinear least-squares solvers for (2.15), and (MW), (CFMR) and (AKZ) are used to test general minimisation solvers for (2.5). Our large-scale nonlinear least-squares solvers in Chapter 7 are designed for (2.6), and will be tested with (CR) and (CR-large). We also optionally enhance these collections with stochastic noise.

[10] Since (2.5) is a more general problem than (2.15), the least-squares test collections can also be used to test general objective problems.

Noise Models Since the presence of noise in the evaluation of the objective is one of the key uses cases for DFO methods, in our testing we will modify the test problems listed above by adding stochastic noise. In all cases, we add stochastic i.i.d. noise for each solver and each evaluation of the objective (including for multiple evaluations at the same $\mathbf{x}$). For nonlinear least-squares problems, we add noise to each residual randomly and independently for each solver and evaluation, and for each entry of $\mathbf{r}(\mathbf{x})$.

For general-objective problems without a least-squares structure, we allow the following noise models, so for a given objective $f(\mathbf{x})$, each solver only sees the noisy evaluations $\widetilde{f}(\mathbf{x})$, where

- Smooth (noiseless) evaluations: $\widetilde{f}(\mathbf{x}) = f(\mathbf{x})$;
- Multiplicative Gaussian noise: $\widetilde{f}(\mathbf{x}) = f(\mathbf{x})(1+\epsilon)$, where $\epsilon \sim N(0, \sigma^2)$; and
- Additive Gaussian noise: $\widetilde{f}(\mathbf{x}) = f(\mathbf{x}) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$.

For problems with a least-squares structure, we allow each solver to only see evaluations of $\widetilde{\mathbf{r}}(\mathbf{x})$, defined component-wise by the noise models:

- Smooth (noiseless) evaluations: $\widetilde{r}_i(\mathbf{x}) = r_i(\mathbf{x})$;
- Multiplicative Gaussian noise: $\widetilde{r}_i(\mathbf{x}) = (1+\epsilon)r_i(\mathbf{x})$, where $\epsilon \sim N(0, \sigma^2)$;
- Additive Gaussian noise: $\widetilde{r}_i(\mathbf{x}) = r_i(\mathbf{x}) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$; and
- Additive χ^2 noise: $\widetilde{r}_i(\mathbf{x}) = \sqrt{r_i(\mathbf{x})^2 + \epsilon^2}$, where $\epsilon \sim N(0, \sigma^2)$.

Similar to the general-objective case, ϵ is drawn i.i.d. for each evaluation of each solver, and each $i = 1, \ldots, m$. In all cases, ϵ is drawn i.i.d. for each evaluation of each solver (including multiple evaluations of the same solver at the same $\mathbf{x}$), and we will usually take $\sigma = 10^{-2}$.

For all of these noise models, we note that $\mathbb{E}[\widetilde{f}(\mathbf{x})]$ is an affine transformation of $f(\mathbf{x})$, and so minimising the expectation of the noisy objective yields the same minimiser(s) as the noiseless objective. Under these noise models (except for least-squares problems with zero residual or additive χ^2 noise), it is possible for a solver to receive a noisy value with $\widetilde{f}(\mathbf{x}) < f^*$ or $\widetilde{f}(\mathbf{x}) < \mathbb{E}[\widetilde{f}(\mathbf{x}^*)]$, but this does not impact the construction of data or performance profiles for noisy problems, as defined via (2.58) below.

2.4.2 Profiles for Noisy Problems

When we test problems modified with noise (as described above), we run each solver multiple times on each problem, where each time the solver sees a different realisation of the noise at each $\mathbf{x}$, and so will likely produce a different sequence of iterates. We will refer to this by saying that we run each solver on multiple instances of each problem. For the local optimisation test problems, each instance has the same starting point $\mathbf{x}_0$ (but different realisations of the noisy objective $\widetilde{f}(\mathbf{x}_0)$).

When comparing solvers with data profiles (2.56), we treat each problem instance as a separate problem to be 'solved'; e.g. if we run 10 instances on a set of problems $\mathcal{P}$, we plot the proportion of the $10|\mathcal{P}|$ problem instances solved within a given budget. For performance profiles (2.57), we do the same (i.e. show a proportion of problem instances), and take $N_P^*(\tau)$ in (2.57) to be the minimum budget used by any solver for any instance of problem P.

In this regime, we now have a choice in how we measure objective decrease (2.55): do we look for reductions in the underlying smooth objective, which is likely what we actually wish to optimise, or the objective with noise added, which is observable by the user? That is, if we minimise a noisy function $\widetilde{f}(\mathbf{x})$ approximating an underlying smooth function $f(\mathbf{x})$, we could construct data and performance profiles using either of:

$$\left\{ \begin{array}{l} N_P(\mathcal{S};\tau) \\ \widetilde{N}_P(\mathcal{S};\tau) \end{array} \right\} := \#\text{ objective evaluations taken to find a point } \mathbf{x} \text{ satisfying} \\ \left\{ \begin{array}{rcl} f(\mathbf{x}) & \leq & f(\mathbf{x}^*) + \tau(f(\mathbf{x}_0) - f(\mathbf{x}^*))] \\ \widetilde{f}(\mathbf{x}) & \leq & \mathbb{E}[\widetilde{f}(\mathbf{x}^*) + \tau(\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*))] \end{array} \right\}, \tag{2.58}$$

Using $N_P(\mathcal{S};\tau)$ measures genuine progress towards the minimiser, excluding any objective reductions from sampling errors, but $\widetilde{N}_P(\mathcal{S};\tau)$ measures progress using information actually available to the solver. Both approaches have each been used previously (e.g. $N_P(\mathcal{S};\tau)$ in [50] and $\widetilde{N}_P(\mathcal{S};\tau)$ in [30]). Ideally, we would have both: base our decrease measure on information observable to the solver, but which captures genuine decreases from optimisation rather than decreases from sampling errors.

Throughout this thesis we use a novel approach, where we choose a problem-specific accuracy level τ_P rather than a constant value for all problems. This level is chosen based on the accuracy that is reasonable for a solver to attain given the noise level in the problem. By adapting τ_P to each problem, we can measure progress using $\widetilde{N}_P$ but gain the benefit of N_P, thus capturing genuine progress in the objective. Having selected τ_P, we construct data and performance profiles as per (2.56) and (2.57), but replacing τ with τ_P, and choosing $\widetilde{N}_P(\mathcal{S},\tau)$.

Choice of adaptive accuracy level τ_P Suppose we have an underlying smooth objective f, but only see evaluations of the noisy objective $\widetilde{f} \approx f$, where $\mathbb{E}[\widetilde{f}(\mathbf{x})] = \alpha f(\mathbf{x}) + \beta$; i.e.

$$\widetilde{f}(\mathbf{x}) := \alpha f(\mathbf{x}) + \beta + \sigma(\mathbf{x})\epsilon, \tag{2.59}$$

for constants $\alpha > 0$ and $\beta \in \mathbb{R}$, and where $\sigma(\mathbf{x})$ is the standard deviation of the noise. The stochastic noise ϵ has zero mean and unit variance, and so ϵ has a magnitude of approximately 1. Under this assumption—which holds for all the noise models we consider in Section 2.4.1—minimising $\mathbb{E}[\widetilde{f}(\mathbf{x})]$ yields a minimiser of the true objective f. Given (2.58), we would expect the two measures N_P and $\widetilde{N}_P$ to be similar, provided that our desired objective reduction was much larger than the noise level. If N_P and $\widetilde{N}_P$ were similar, we could conclude that reductions in the observable $\widetilde{f}$ correspond to genuine objective reductions, and not sampling error.

Specifically, suppose a solver has reached a point $\mathbf{x}_k$, which corresponds to an accuracy of (exactly) τ based on N_P, and of $\widetilde{\tau}$ based on $\widetilde{N}_P$; that is

$$f(\mathbf{x}_k) = f(\mathbf{x}^*) + \tau(f(\mathbf{x}_0) - f(\mathbf{x}^*)), \quad \text{and} \quad \widetilde{f}(\mathbf{x}_k) = \mathbb{E}[\widetilde{f}(\mathbf{x}^*) + \widetilde{\tau}(\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*))]. \tag{2.60}$$

Letting ϵ_k be the realisation of ϵ for the given $\widetilde{f}(\mathbf{x}_k)$, we combine the expressions in (2.60) and (2.59), to get

$$\widetilde{f}(\mathbf{x}_k) = \mathbb{E}[\widetilde{f}(\mathbf{x}^*) + \widetilde{\tau}(\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*))], \tag{2.61}$$

$$\alpha f(\mathbf{x}_k) + \beta + \sigma(\mathbf{x}_k)\epsilon_k = \alpha f(\mathbf{x}^*) + \beta + \widetilde{\tau}\alpha(f(\mathbf{x}_0) - f(\mathbf{x}^*)), \tag{2.62}$$

$$\alpha\left[f(\mathbf{x}^*) + \tau(f(\mathbf{x}_0) - f(\mathbf{x}^*))\right] + \beta + \sigma(\mathbf{x}_k)\epsilon_k = \alpha f(\mathbf{x}^*) + \beta + \widetilde{\tau}\alpha(f(\mathbf{x}_0) - f(\mathbf{x}^*)), \tag{2.63}$$

$$\tau\alpha(f(\mathbf{x}_0) - f(\mathbf{x}^*)) + \sigma(\mathbf{x}_k)\epsilon_k = \widetilde{\tau}\alpha(f(\mathbf{x}_0) - f(\mathbf{x}^*)), \tag{2.64}$$

$$\tau\,\mathbb{E}\left[\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*)\right] + \sigma(\mathbf{x}_k)\epsilon_k = \widetilde{\tau}\,\mathbb{E}\left[\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*)\right], \tag{2.65}$$

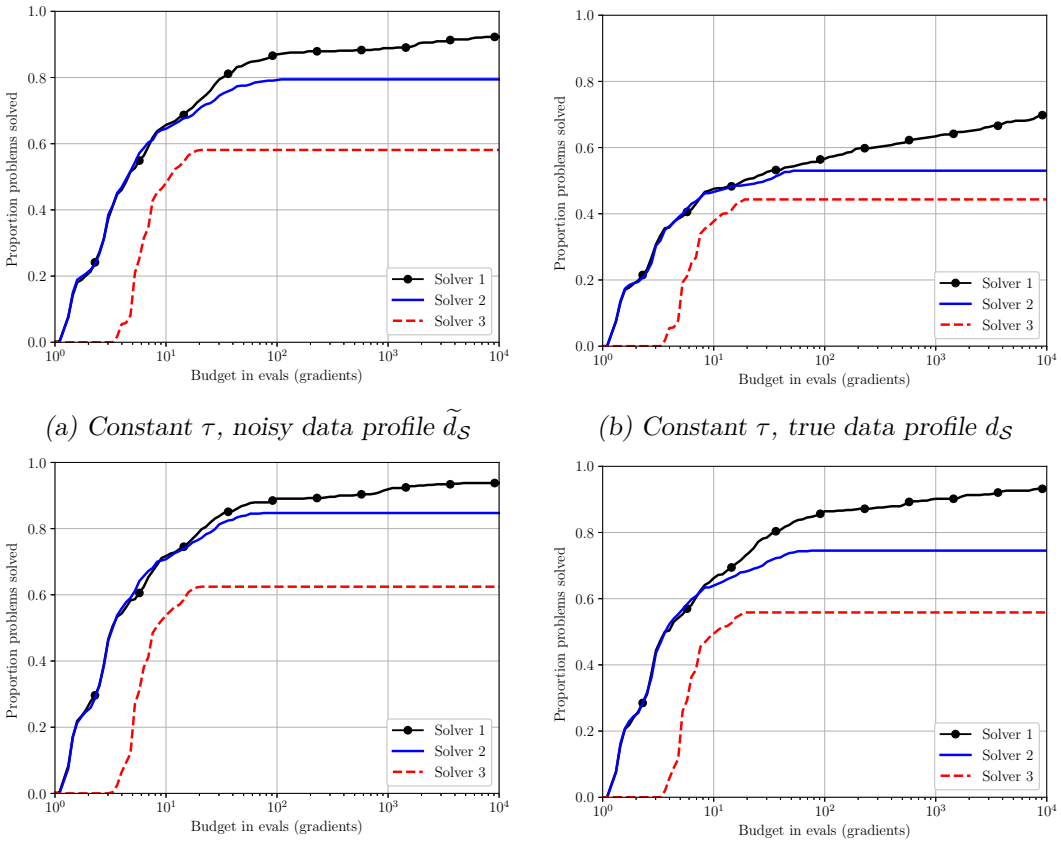
and so

$$\underbrace{\widetilde{\tau}}_{\text{noisy progress}} = \underbrace{\tau}_{\text{true progress}} + \underbrace{\frac{\sigma(\mathbf{x}_k)}{\mathbb{E}\left[\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*)\right]}\epsilon_k}_{\text{sampling error}}, \tag{2.66}$$

where the 'sampling error' term corresponds to the perceived objective reduction from the specific realisation of noise ϵ_k, and does contribute to any actual progress in the optimisation. Since $|\epsilon_k|$ is of order 1, we can expect $\widetilde{\tau}$ and τ to be similar in size whenever

$$\tau \text{ and/or } \widetilde{\tau} \;\gg\; \frac{\sigma(\mathbf{x}_k)}{\mathbb{E}\left[\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*)\right]}. \tag{2.67}$$

Effectively, (2.67) provides a limit on the accuracy we can reasonably expect a solver to achieve, given the noise level in a particular problem. In our numerical results, we



(a) Constant τ, noisy data profile $\widetilde{d}_{\mathcal{S}}$

(b) Constant τ, true data profile $d_{\mathcal{S}}$

(c) Per-problem τ_P, noisy data profile $\widetilde{d}_{\mathcal{S}}$

(d) Per-problem τ_P, true data profile $d_{\mathcal{S}}$

Figure 2.1. Example data profiles for three solvers, where we use the data profiles $d_{\mathcal{S}}$ and $\widetilde{d}_{\mathcal{S}}$ (2.56), and either choosing $\tau = 10^{-5}$ for all problems, or applying the per-problem threshold τ_P (2.69). The problem collection is (MW) with additive Gaussian noise ($\sigma = 10^{-2}$)—see Section 2.4.1 for details. These results include 10 instances of each problem for each solver.

approximate (2.67) in a way that is independent of $\mathbf{x}_k$, and say that the best accuracy we can expect a solver to achieve is $\tau_{\text{crit}}(P)$, where

$$\tau_{\text{crit}}(P) := 10^{\lceil \log_{10} \widehat{\tau}(P) \rceil}, \qquad \text{where} \qquad \widehat{\tau}(P) := \frac{\sigma(\mathbf{x}^*)}{\mathbb{E}\left[\widetilde{f}(\mathbf{x}_0) - \widetilde{f}(\mathbf{x}^*)\right]}, \tag{2.68}$$

that is, $\tau_{\text{crit}}(P)$ is $\widehat{\tau}(P)$ rounded to the next largest integer power of 10. To achieve a threshold independent of k, we have approximated $\sigma(\mathbf{x}_k) \approx \sigma(\mathbf{x}^*)$, based on the expectation that our solver has been effective, and reached a point $\mathbf{x}_k \approx \mathbf{x}^*$.

Finally, to construct data and performance profiles, we choose a desired level of accuracy τ and set our problem-specific tolerance to be either τ if we can expect the solver to reach this accuracy, or $\tau_{\text{crit}}(P)$ otherwise. Thus, in our profiles (2.56) and (2.57), we use the problem-specific accuracy level

$$\tau_P := \min(\tau_{\max}, \max(\tau_{\text{crit}}(P), \tau)), \tag{2.69}$$

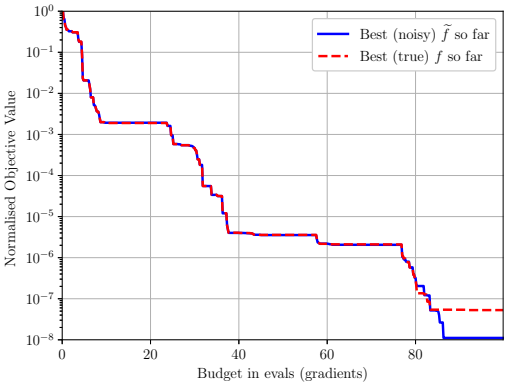


Figure 2.2. Normalised objective value for one solver run on the Osborne 1 test problem (#36 in (MW) collection) with unbiased multiplicative Gaussian noise and $\sigma = 10^{-2}$. We show the best objective value found so far, as per the true underlying smooth function or the observed noisy function.

where $\tau_{\max} := 10^{-1}$ is an upper bound on τ_P. We note that for noiseless problems we have $\tau_{\text{crit}}(P) = 0$, so $\tau_P = \tau$ is problem-independent.

Impact of $\boldsymbol{\tau_P}$ We now illustrate that using the per-problem accuracy level τ_P for measuring noisy progress $\widetilde{N}_P$ allows us to compare performance based on genuine objective decreases, not sampling errors. We do this by verifying that the performance measures N_P and $\widetilde{N}_P$, and corresponding data profiles $d_{\mathcal{S}}$ and $\widetilde{d}_{\mathcal{S}}$, give similar results.

In Figure 2.1, we show a set of example data profiles using both the measures of objective reduction (2.58), and consider either a fixed $\tau = 10^{-5}$ for all problems in (a) and (b), or using the per-problem value τ_P (2.69) in (c) and (d). We see that when a constant value of τ is used, the two data profiles look very different, with the noisy profile $\widetilde{d}_{\mathcal{S}}$ showing more problems solved than the true profile $d_{\mathcal{S}}$. When we switch to the per-problem threshold (2.69), the two profiles look much more consistent both in shape and magnitude.

As another demonstration, in Figure 2.2 we show the true and noisy objective reduction achieved by a solver on the Osborne 1 test problem from (MW) with multiplicative Gaussian noise. These are normalised, and so we are effectively plotting the best τ and $\widetilde{\tau}$ (2.60) achieved by the solver for a given budget. For this problem and noise model, we have $\tau_{\text{crit}}(P) = 10^{-7}$. We observe that the decrease achieved by the solver is almost identical under both measures until the accuracy $\tau_{\text{crit}}(P)$ is achieved, at which point the two measures start to diverge. This matches the expected behaviour, as given by (2.66).

Thus, for the remainder of this thesis, we present our numerical results using the problem-adjusted τ_P with the noisy data profile $\widetilde{d}_{\mathcal{S}}$.

Chapter 3

Derivative-Free Optimisation for Nonlinear Least-Squares

In this chapter, we define our derivative-free algorithm for nonlinear least-squares minimisation, DFO-LS (Derivative-Free Optimisation for Least-Squares).[1] DFO-LS is a model-based trust-region DFO method, which constructs linear models for each residual (along similar lines to the classical Gauss-Newton method). DFO-LS solves the general nonlinear least-squares problem (2.6) where $\mathbf{r}(\mathbf{x})$ is a continuously differentiable function from $\mathbb{R}^n$ to $\mathbb{R}^m$, but its Jacobian matrix of first derivatives is unavailable. Both the case when $m \geq n$ (overdetermined, least-squares problems) and $m \leq n$ (underdetermined, inverse problems) are allowed.

DFO-LS has several novel aspects compared to existing methods. Firstly, it uses linear residual models rather than quadratic—we note that DFLS, the algorithm from [240] in principle allows linear residual models, but its implementation only allows for quadratic residual models. DFO-LS also has a more flexible model construction approach, allowing the use of underdetermined linear models. This gives DFO-LS the ability to use a reduced initialisation cost before it begins the main iteration. DFO-LS also has several mechanisms for improving its robustness to noise, most notably the ability to use different algorithm parameters for noisy problems, and a multiple restarts mechanism. It also implements sample averaging and linear regression models, however these are not enabled by default.

In Section 3.1, we outline the algorithmic framework and state the full DFO-LS algorithm. We then outline the approach we take to the construction of linear residual models in Section 3.2, including underdetermined models leading to a reduced initialisation cost. Finally, we explain several aspects of the implementation of DFO-LS in Section 3.3, and its features for noisy problems in Section 3.4.

[1] Available from `https://github.com/numericalalgorithmsgroup/dfols`

In Chapter 4 we provide theoretical guarantees for a simplified DFO-LS algorithm, and in Chapter 5 we show numerical results for DFO-LS.

3.1 Algorithmic Framework

The DFO-LS algorithm for (2.6) is based on a trust-region framework [53]. As described in Chapter 1, in such a framework we construct a quadratic model for the f at each iteration k. In DFO-LS, we do this by first constructing a linear vector model for $\mathbf{r}(\mathbf{x})$, which we aim to be accurate close to our current iterate $\mathbf{x}_k$. That is, we construct

$$\mathbf{r}(\mathbf{x}_k + \mathbf{s}) \approx \mathbf{m}_k(\mathbf{s}) := \mathbf{r}_k + J_k\mathbf{s}, \tag{3.1}$$

using interpolation—we describe this process in Section 3.2. Using $\mathbf{m}_k$, we can construct a local quadratic model for f in a natural way, namely

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) := \frac{1}{2}\|\mathbf{m}_k(\mathbf{s})\|^2 = \frac{1}{2}\|\mathbf{r}_k\|^2 + \mathbf{g}_k^\top\mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_k\mathbf{s}, \tag{3.2}$$

where $\mathbf{g}_k := J_k^\top\mathbf{r}_k$ and $H_k := J_k^\top J_k$.

Given our model for the objective (3.2) and a trust-region radius $\Delta_k > 0$, we calculate a step by (approximately) solving the trust-region subproblem (1.6). As described in Section 1.1.2, we then calculate a measure of progress, which is the ratio

$$R_k = \frac{\text{actual reduction}}{\text{predicted reduction}} := \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}. \tag{3.3}$$

If R_k is sufficiently large, then we accept the step, otherwise we reject it. We also use R_k to update the trust-region radius Δ_k.

In our implementation, we solve the trust-region subproblem (1.6) using the method from BOBYQA [186], which is a based on the Steihaug-Toint algorithm [53, Algorithm 7.5.1]. This is an iterative algorithm, based on the Conjugate Gradient method, which does not solve (1.6) to full optimality. Instead, it finds an approximate solution—which is sufficient for global convergence (see Assumption 4.4)—in a matrix-free way; that is, it only requires H_k through matrix-vector products.

Geometry Considerations In Chapter 2, we defined the notion of fully linear models, which can be constructed using linear interpolation/regression over Λ-poised sets. As well as updating our interpolation set to incorporate $\mathbf{x}_k + \mathbf{s}_k$ at every iteration, at several points in Algorithm 3.1, we ask that the interpolation set be made Λ-poised, which can be achieved using the mechanisms described in Section 2.3.

Similar to Section 2.2, in Section 4.1 we show that if Y_k is Λ-poised, then our model m_k for f (3.2) is fully linear, with constants that depend on Λ.

3.1.1 Full Algorithm Specification

A full description of the DFO-LS algorithm is provided in Algorithm 3.1.

In each iteration, if $\mathbf{g}_k$ is small, we apply a 'criticality phase'. This ensures that Δ_k is comparable in size to $\|\mathbf{g}_k\|$, which makes both Δ_k and $\|\mathbf{g}_k\|$ good measures of progress towards optimality (see Section 2.3 for further details). After computing the trust-region step $\mathbf{s}_k$, we then apply a 'safety phase', also originally from Powell [181]. In this phase, we check if $\|\mathbf{s}_k\|$ is too small compared to a lower bound ρ_k on the trust-region radius (see below), and if so we reduce Δ_k and improve the geometry of Y_k, without evaluating $\mathbf{r}(\mathbf{x}_k + \mathbf{s}_k)$. The intention of this step is to detect situations where our trust-region step will likely not provide sufficient function decrease without evaluating the objective, which would otherwise be wasteful. If the safety phase is not called, we evaluate $\mathbf{r}(\mathbf{x}_k + \mathbf{s}_k)$ and determine how good the trust-region step was, accepting any point that achieved sufficient objective decrease. There are two possible causes for the situation $R_k < \eta_1$ (i.e. the trust region step was 'bad'): the interpolation set is not good enough, or Δ_k is too large. We first check the quality of the interpolation set, and only reduce Δ_k if necessary.

An important feature of DFO-LS, due to Powell [181], is that it maintains not only the (usual) trust region radius Δ_k—used in (1.6) and in checking Λ-poisedness—but also a lower bound on it, ρ_k. This mechanism is useful when we reject the trust region step, but the geometry of Y_k is not good (the 'Model Improvement Phase'). In this situation, we do not want to shrink Δ_k too much, because it is likely that the step was rejected because of the poor geometry of Y_k, not because the trust region was too large. The algorithm floors Δ_k at ρ_k, and only shrinks Δ_k when we reject the trust region step *and* the geometry of Y_k is good (so the model m_k is accurate)—in this situation, we know that reducing Δ_k will actually be useful.

We also note that Algorithm 3.1 includes several termination conditions and a multiple restarts mechanism. These are described in Section 3.3 and Section 3.4 respectively.

Remark 3.1. In Lemma 4.2, we show that if Y_k is Λ-poised, then m_k is fully linear with constants that depend on Λ. For the highest level of generality, one may replace 'make Y_k Λ-poised' with 'make m_k fully linear' throughout Algorithm 3.1. Any strategy that achieves fully linear models would be sufficient for the convergence results in Section 4.2; this will be relevant in Section 6.2.

Remark 3.2. The default trust-region parameter values in the implementation of DFO-LS are: $\Delta_{\max} = 10^{10}$, $\gamma_{\text{dec}} = 0.5$, $\gamma_{\text{inc}} = 2$, $\overline{\gamma}_{\text{inc}} = 4$, $\eta_1 = 0.1$, $\eta_2 = 0.7$, $\alpha_1 = 0.1$, $\alpha_2 = 0.5$, $\omega_S = 0.1$ and $\gamma_S = 0.5$. In addition, the default choice of initial trust-region radius is

Algorithm 3.1 DFO-LS: Derivative-Free Optimisation for Least-Squares.

Input: Starting point $\mathbf{x}_0 \in \mathbb{R}^n$, initial trust region radius $\Delta_0^{\text{init}} > 0$ and integers p_{init} and p, the sizes of the initial and final interpolation sets, respectively, where $1 \leq p_{\text{init}} \leq p$ and $p \geq n$.

Parameters: maximum trust-region radius $\Delta_{\max} \geq \Delta_0^{\text{init}}$, criticality threshold $\epsilon_C > 0$, criticality scaling $\mu > 0$, minimum trust-region radius $0 < \rho_{\text{end}} < \Delta_0^{\text{init}}$, trust-region radius scalings $0 < \gamma_{\text{dec}} < 1 < \gamma_{\text{inc}} \leq \overline{\gamma}_{\text{inc}}$ and $0 < \alpha_1 < \alpha_2 < 1$, acceptance thresholds $0 < \eta_1 \leq \eta_2 < 1$, safety reduction factor $0 < \omega_S < 1$, safety step threshold $0 < \gamma_S < 2c_1/(1+\sqrt{1+2c_1})$, poisedness constant $\Lambda \geq 1$, and Boolean flag `NOISY` indicating the presence of noise in the objective.

1: Build an initial interpolation set $Y_0 \subset B(\mathbf{x}_0, \Delta_0^{\text{init}})$ of size $p_{\text{init}}+1$, with $\mathbf{x}_0 \in Y_0$. Set $\rho_0^{\text{init}} = \Delta_0^{\text{init}}$.
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: **if** `NOISY` **and** *all values* $\{f(\mathbf{y}) : \mathbf{y} \in Y_k\}$ *are within noise level of* $f(\mathbf{x}_k)$ **then**
4: Call restart: set $\Delta_{k+1}^{\text{init}} = \rho_{k+1}^{\text{init}} = \Delta_0^{\text{init}}$ and build Y_{k+1} as per Section 3.4.2, then **goto** line 2.
5: **end if**
6: Given $\mathbf{x}_k$ and Y_k, construct the models $\mathbf{m}_k^{\text{init}}(\mathbf{s})$ (3.1) and m_k^{init} (3.2) by solving the interpolation problem (3.8) if $|Y_k| < p+1$, otherwise (3.6).
7: **if** $\|\mathbf{g}_k^{\text{init}}\| \leq \epsilon_C$ **then**
8: Criticality Phase: using Algorithm 2.2, modify Y_k and find $\Delta_k \leq \Delta_k^{\text{init}}$ such that Y_k is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$ and $\Delta_k \leq \mu\|\mathbf{g}_k\|$, where $\mathbf{g}_k$ is the gradient of the new m_k. Set $\rho_k = \min(\rho_k^{\text{init}}, \Delta_k)$.
9: **else**
10: Set $m_k = m_k^{\text{init}}$, $\Delta_k = \Delta_k^{\text{init}}$ and $\rho_k = \rho_k^{\text{init}}$.
11: **end if**
12: Approximately solve the trust-region subproblem (1.6) to get a step $\mathbf{s}_k$ satisfying Assumption 4.4.
13: **if** $\|\mathbf{s}_k\| < \gamma_S \rho_k$ **then**
14: **if** $|Y_k| < p+1$ **then**
15: Safety Phase (Growing): Form $Y_{k+1} = Y_k \cup \{\mathbf{x}_k + \mathbf{s}\}$ for some $\mathbf{s}$ orthogonal to $\{\mathbf{y} - \mathbf{x}_k : \mathbf{y} \in Y_k\}$ with $\|\mathbf{s}\| = \Delta_k$.
16: Set $(\rho_{k+1}^{\text{init}}, \Delta_{k+1}^{\text{init}}) = (\rho_k, \Delta_k)$.
17: **else**
18: Safety Phase: Set $\mathbf{x}_{k+1} = \mathbf{x}_k$ and $\Delta_{k+1}^{\text{init}} = \max(\rho_k, \omega_S \Delta_k)$, and form Y_{k+1} by making Y_k Λ-poised in $B(\mathbf{x}_{k+1}, \Delta_{k+1}^{\text{init}})$.
19: If $\Delta_{k+1}^{\text{init}} = \rho_k$, set $(\rho_{k+1}^{\text{init}}, \Delta_{k+1}^{\text{init}}) = (\alpha_1 \rho_k, \alpha_2 \rho_k)$, otherwise set $\rho_{k+1}^{\text{init}} = \rho_k$.
20: If $\rho_{k+1}^{\text{init}} \leq \rho_{\text{end}}$: call restart if `NOISY`, else **terminate**.
21: **end if**
22: **goto** line 2.
23: **end if**
24: Evaluate $\mathbf{r}(\mathbf{x}_k + \mathbf{s}_k)$ and calculate ratio R_k (3.3).
25: Accept/reject step and update trust region radius: set

$$\mathbf{x}_{k+1} = \begin{cases} \mathbf{x}_k + \mathbf{s}_k, & R_k \geq \eta_1, \\ \mathbf{x}_k, & R_k < \eta_1, \end{cases} \quad \text{and} \quad \Delta_{k+1}^{\text{init}} = \begin{cases} \min(\max(\gamma_{\text{inc}} \Delta_k, \overline{\gamma}_{\text{inc}} \|\mathbf{s}_k\|), \Delta_{\max}), & R_k \geq \eta_2, \\ \max(\gamma_{\text{dec}} \Delta_k, \|\mathbf{s}_k\|, \rho_k), & \eta_1 \leq R_k < \eta_2, \\ \max(\min(\gamma_{\text{dec}} \Delta_k, \|\mathbf{s}_k\|), \rho_k), & R_k < \eta_1. \end{cases} \tag{3.4}$$

26: **if** $|Y_k| < p+1$ **then**
27: Growing Phase: Form $Y_{k+1} = Y_k \cup \{\mathbf{x}_k + \mathbf{s}_k\}$ and set $\rho_{k+1}^{\text{init}} = \rho_k$.
28: **else if** $R_k \geq \eta_1$ **then**
29: Successful Phase: Form $Y_{k+1} = Y_k \cup \{\mathbf{x}_{k+1}\} \setminus \{\mathbf{y}\}$ for some $\mathbf{y} \in Y_k$ and set $\rho_{k+1}^{\text{init}} = \rho_k$.
30: If objective decrease is too slow: call restart if `NOISY`, else **terminate**.
31: **else if** `NOISY` **and** *restart auto-detected* **then**
32: Restart Auto-Detection: Call restart.
33: **else if** Y_k *is not* Λ*-poised in* $B(\mathbf{x}_k, \Delta_k)$ **then**
34: Model Improvement Phase: Form Y_{k+1} by making Y_k Λ-poised in $B(\mathbf{x}_{k+1}, \Delta_{k+1}^{\text{init}})$ and set $\rho_{k+1}^{\text{init}} = \rho_k$.
35: **else**
36: Unsuccessful Phase: Set $Y_{k+1} = Y_k$, and if $\Delta_{k+1}^{\text{init}} = \rho_k$, set $(\rho_{k+1}^{\text{init}}, \Delta_{k+1}^{\text{init}}) = (\alpha_1 \rho_k, \alpha_2 \rho_k)$, otherwise set $\rho_{k+1} = \rho_k$.
37: If $\rho_{k+1}^{\text{init}} \leq \rho_{\text{end}}$: call restart if `NOISY`, else **terminate**.
38: **end if**
39: **end for**

$\Delta_0^{\text{init}} = 0.1 \max(\|\mathbf{x}_0\|_\infty, 1)$. However, we note that we allow DFO-LS to use different default parameters for noisy problems—see Section 3.4.1.

3.2 Linear Interpolation Model Construction

At each iteration, DFO-LS constructs a linear model for $\mathbf{r}(\mathbf{x})$ (3.1) in a neighbourhood of the current iterate $\mathbf{x}_k$, mimicking the classical Gauss-Newton method (Section 1.1.3). To achieve this linearisation in a derivative-free way, we maintain a set of $p+1$ points $Y_k = \{\mathbf{x}_k, \mathbf{y}_1, \dots, \mathbf{y}_p\} \subset \mathbb{R}^n$, where we let $\mathbf{y}_0 := \mathbf{x}_k$ for notational convenience. This leads to several regimes, depending on the choice of p. The default choice for DFO-LS is to use $p = n$. This leads to interpolation problems that are square linear systems, and hence a unique linear interpolation model (see Section 2.2.1) for each residual.

DFO-LS also allows the option of $p > n$, which leads to overdetermined linear systems. This leads to linear regression models for each residual (see Section 2.2.2) rather than interpolation, and is designed for the case of noisy objectives. We discuss the implementation of regression models in DFO-LS in Section 3.4.3, but they are not used by default.

We also allow the option $p < n$ in early iterations, to reduce the initial evaluation cost of DFO-LS. This construction is detailed in Section 3.2.1 below. It is not used by default in DFO-LS, but is potentially useful in settings with very small budgets. The performance of DFO-LS with reduced initialisation cost is discussed more in Section 5.2.2.

Standard Linear Model Construction When $p \geq n$, we build a model (3.1) using linear regression in each variable (see Section 2.2.2). That is, we solve the problem

$$\min_{\mathbf{r}_k, J_k} \sum_{t=0}^{p} \|\mathbf{m}_k(\mathbf{y}_t - \mathbf{x}_k) - \mathbf{r}(\mathbf{y}_t)\|^2, \tag{3.5}$$

or equivalently find the least-squares solutions to the overdetermined $(p+1) \times (n+1)$ linear systems (c.f. (2.25))

$$W_k \begin{bmatrix} r_{k,i} \\ \mathbf{j}_{k,i} \end{bmatrix} := \begin{bmatrix} 1 & (\mathbf{y}_0 - \mathbf{x}_k)^\top \\ \vdots & \vdots \\ 1 & (\mathbf{y}_p - \mathbf{x}_k)^\top \end{bmatrix} \begin{bmatrix} r_{k,i} \\ \mathbf{j}_{k,i} \end{bmatrix} = \begin{bmatrix} r_i(\mathbf{y}_0) \\ \vdots \\ r_i(\mathbf{y}_p) \end{bmatrix}, \tag{3.6}$$

for all $i = 1, \dots, m$, where $r_{k,i}$ and $\mathbf{j}_{k,i}^\top$ are the i-th entry of $\mathbf{r}_k$ and row of J_k respectively.[2] In the usual case of $p = n$, we get $\mathbf{r}_k = \mathbf{r}(\mathbf{x}_k)$, and (3.6) is a square linear system, however we use the formulation (3.5) to allow both $p = n$ and regression models $p > n$.

The matrix W_k has full column rank whenever Y_k is poised for linear regression (Definition 2.13); we ensure this in DFO-LS by calling the poisedness-improving procedures to

[2] The same set of interpolation points are used in (3.6) for each $i = 1, \dots, m$.

ensure Y_k has good geometry. However, as the algorithm progresses, the points $\{\mathbf{y}_1, \ldots, \mathbf{y}_p\}$ get progressively closer to $\mathbf{x}_k$, so W_k becomes ill-conditioned. To avoid this issue, we precondition (3.6) by scaling the second through last columns of W_k by α_k^{-1}, where $\alpha_k := \max_{t=1,\ldots,p} \|\mathbf{y}_t - \mathbf{x}_k\|$.

As described above, once we have built the model $\mathbf{m}_k$ (3.1) for $\mathbf{r}$, we construct the quadratic model m_k for the full objective $f(\mathbf{x})$ as per (3.2).

Remark 3.3. An alternative interpolation framework that we considered, inspired by a comment in [60, Chapter 4], was to use interpolation models (i.e. $p = n$) and replace (3.5) with

$$\min_{\mathbf{r}_k, J_k} \|J_k - J_{k-1}\|_F^2 + \lambda_k \sum_{t=0}^{n} \|\mathbf{m}_k(\mathbf{y}_t - \mathbf{x}_k) - \mathbf{r}(\mathbf{y}_t)\|^2, \tag{3.7}$$

where $\lambda_k > 0$ is an algorithmic parameter. This formulation is similar to the interpolation formulation when $p < n$—see (3.8) below—but can also be viewed as a regularised version of the regression problem (3.5). This idea allows inexact interpolation, and is designed to balance accuracy of interpolation against large changes in the model between iterations, was motivated by the case of noisy objective evaluation. However, our extensive testing showed that the best results for this framework, even for noisy objectives, required setting λ_k very large (at least 10^{10}), which means that we are essentially solving (3.5).

3.2.1 Reduced Initialisation Cost for Expensive Objectives

The problem (3.5) requires $p \geq n$, so that the system (3.6) is square or overdetermined. This means that before the first model can be constructed, we must evaluate the objective at $p + 1$ points—this is common in model-based DFO algorithms. Although these evaluations may be parallelised, a user may not have the ability to do this, and the cost of these initial evaluations may be prohibitive. In such settings, DFO-LS can proceed with a reduced initialisation cost, constructing the model (3.1) using as few as 2 objective evaluations.[3]

Suppose we have evaluated the objective at $p+1$ affinely-independent points $\{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ with $\mathbf{y}_0 := \mathbf{x}_k$, where we now assume $1 \leq p < n$. We construct $\mathbf{m}_k$ by solving the same interpolation system (3.6), which is now underdetermined, and for which we select the minimal (Euclidean) norm solution. The resulting $\mathbf{r}_k$ and J_k are solutions to

$$\min_{\mathbf{r}_k, J_k} \|\mathbf{r}_k\|^2 + \alpha_k^2 \|J_k\|_F^2 \qquad \text{s.t.} \qquad \mathbf{m}_k(\mathbf{y}_t - \mathbf{x}_k) = \mathbf{r}(\mathbf{y}_t), \quad \forall t = 0, \ldots, p, \tag{3.8}$$

[3] In practice, solvers can accept any objective decrease from the initial evaluations [186, 12], and use the best initial point as the first iterate. DFO-LS does this, but here we refer to a mechanism to begin the main iteration after few initial evaluations, where new iterates are specifically generated to progress towards the solution from the best point so far, rather than just sampling the search space.

where α_k, defined above, is the column scaling used to precondition (3.6).[4]

However, the construction (3.8) is not ideal, because, as expected—and proven in Lemma 3.4 below—the resulting J_k is not full rank, so the models $\mathbf{m}_k$ and m_k are not full-dimensional; that is, there are directions along which these are constant, regardless of the objective.

Lemma 3.4. *Suppose* $\mathbf{m}_k$ (3.1) *is constructed using* (3.8) *with* $p < n$. *Then* J_k *has column rank at most* p.

Proof. We first write the interpolation matrix $W_k \in \mathbb{R}^{(p+1)\times(n+1)}$ in (3.6) as

$$W_k = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{e} & L_k/\alpha_k \end{bmatrix}, \qquad \text{where} \qquad L_k := \begin{bmatrix} (\mathbf{y}_1 - \mathbf{x}_k)^\top \\ \vdots \\ (\mathbf{y}_p - \mathbf{x}_k)^\top \end{bmatrix} \in \mathbb{R}^{p\times n}, \tag{3.9}$$

where $\mathbf{e} \in \mathbb{R}^p$ is the matrix of ones and $\alpha_k > 0$ is the scaling factor for preconditioning (see above). Suppose $L_k^\top$ has column rank $r \leq p$, and so there exists an orthonormal basis $\{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ for $\mathbb{R}^n$ so that $\{\mathbf{s}_1, \ldots, \mathbf{s}_r\}$ is a basis for $\operatorname{col}(L_k^\top)$ and $\{\mathbf{s}_{r+1}, \ldots, \mathbf{s}_n\}$ is a basis for $\operatorname{col}(L_k^\top)^\perp = \operatorname{null}(L_k)$.

Now, suppose $\mathbf{v} \in \operatorname{null}(W_k) \subset \mathbb{R}^{n+1}$. Then from (3.9) we have the first entry $v_1 = 0$ and for the remaining entries,

$$\begin{bmatrix} v_2 \\ \vdots \\ v_{n+1} \end{bmatrix} \in \operatorname{null}(L_k). \tag{3.10}$$

Thus, since $\{\mathbf{s}_{r+1}, \ldots, \mathbf{s}_n\}$ is a basis for $\operatorname{null}(L_k)$, we have

$$\operatorname{null}(W_k) = \left\{ \begin{bmatrix} 0 \\ \widetilde{\mathbf{v}} \end{bmatrix} : \widetilde{\mathbf{v}} \in \operatorname{null}(L_k) \right\} = \operatorname{span}\left\{ \begin{bmatrix} 0 \\ \mathbf{s}_{r+1} \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \mathbf{s}_n \end{bmatrix} \right\}. \tag{3.11}$$

Finally, since

$$\left\{ \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} 0 \\ \mathbf{s}_1 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \mathbf{s}_n \end{bmatrix} \right\}, \tag{3.12}$$

is an orthonormal basis for $\mathbb{R}^{n+1}$, we conclude

$$\operatorname{col}(W_k^\dagger) = \operatorname{col}(W_k^\top) = \operatorname{null}(W_k)^\perp = \operatorname{span}\left\{ \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} 0 \\ \mathbf{s}_1 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \mathbf{s}_r \end{bmatrix} \right\}. \tag{3.13}$$

Therefore from solving (3.6) we have $\mathbf{j}_{k,i} \in \operatorname{span}\{\mathbf{s}_1, \ldots, \mathbf{s}_r\}$, and so J_k has column rank at most r. □

Thus by using this model, we will not in general be able to solve (2.6), as we will only ever search in the subspace spanned by our starting directions. We now discuss how to avoid this issue in practice.

[4] Note that because in this phase of the algorithm we never remove points from Y_k, provided $\{\mathbf{y}_t - \mathbf{x}_k : t = 1, \ldots, p\}$ is linearly independent, (3.8) is equivalent to minimising the change in the model, $\min_{\mathbf{r}_k, J_k} \|\mathbf{r}_k - \mathbf{r}_{k-1}\|^2 + \alpha_k^2 \|J_k - J_{k-1}\|_F^2$.

3.2.2 Implementation of Reduced Initialisation Cost

We must begin DFO-LS by evaluating the objective at $p_{\text{init}} + 1$ points. To reduce this cost, we may set $p_{\text{init}} < n$; however, Lemma 3.4 shows that the resulting model is low rank, and so our algorithm will not search outside the initial subspace of search directions, or find a minimiser, in general. Thus we need some mechanism where, for $p_{\text{init}} < n$, we can bring in function information from the space not spanned by our existing interpolation points.

A simple way to address this issue would be to, at each iteration, replace one point with the new iterate $\mathbf{x}_{k+1}$ using standard methods, then add another point to the interpolation set, chosen to increase the dimension of the model (until a full-dimensional model is achieved). However, this would require two objective evaluations per iteration, which may be wasteful when evaluations are expensive; if the user has specified $p_{\text{init}} < n$, this is particularly likely to be relevant.

Making the Jacobian have full rank in the expensive regime After calculating the rank-deficient J_k, DFO-LS makes it full rank—and hence makes the model full-dimensional—by increasing its $n - p$ smallest singular values $\sigma_{p+1} = \cdots = \sigma_n = 0$ to the level of the smallest nonzero singular value $\sigma_p > 0$; this requires the calculation of the SVD of J_k.

To handle the case when J_k has rank (numerical or exact) strictly less than p, we also floor all singular values at a small positive value (default 10^{-6}), to ensure the model is always full-dimensional; this means that some interpolation conditions may not be satisfied, but only during this initialisation phase—once a full set of $n + 1$ interpolation points is available, we no longer perturb the singular values of J_k. Since J_k has $\min(m, n)$ singular values, if $m < n$ then this SVD-based approach still produces a model that is constant in some directions, which is not desirable. Thus the SVD-based mechanism is only used in DFO-LS when $m \geq n$.

Alternative mechanism for expanding the search space DFO-LS has another optional mechanism for increasing the dimension of the model, instead of perturbing the singular values of J_k. In this approach, after finding the trust region step $\mathbf{s}_k$, we replace the new candidate point $\mathbf{x}_k + \mathbf{s}_k$ with the perturbed point $\mathbf{x}_k + \mathbf{s}_k + \mathbf{d}_k$, where $\mathbf{d}_k$ is a random direction orthogonal to our current set of search directions (with length a constant multiple of Δ_k). This mechanism is the default in DFO-LS for inverse problems with $m < n$.

We compare these two approaches in Section 5.2.2, and conclude that the SVD-based variant has similar performance to the random direction extension for small budgets, but better performance for longer budgets. Thus the SVD approach is chosen as the default

in DFO-LS when an initialisation with fewer than n interpolation points is used, provided $m \geq n$ (and the alternative mechanism is used only when $m < n$).

3.3 General Implementation Features of DFO-LS

In this section, we outline some details of the implementation of DFO-LS, beyond the description in Algorithm 3.1.

3.3.1 Geometry-Improving Phases

In practice, DFO algorithms are generally not run to very high tolerance levels, and so the asymptotic behaviour of such algorithms is less important than for other optimisation methods. To this end, DFO-LS, like BOBYQA [186] and DFBOLS (the implementation of DFLS from [240]), does not implement a criticality phase.

In the geometry phases of the algorithm, we check the Λ-poisedness of Y_k by calculating all the Lagrange polynomials for Y_k (which are linear), then maximising the absolute value of each in $B(\mathbf{x}_k, \Delta_k)$, as outlined in Section 2.3. In DFO-LS, we follow BOBYQA and replace these geometry-checking and improvement algorithms (which are called in the safety and model-improvement phases of Algorithm 3.1) with simplified calculations. Firstly, instead of checking for the Λ-poisedness of Y_k, we instead check if all interpolation points are within some distance of $\mathbf{x}_k$, typically a multiple of Δ_k. If any point is sufficiently far from $\mathbf{x}_k$, the geometry of Y_k is improved by selecting the point $\mathbf{y}_t$ furthest from $\mathbf{x}_k$, and moving it to the maximiser of $|\ell_t|$ inside the trust region; that is, we run the full geometry-improving procedure Algorithm 2.1 for one iteration, which yields an improved model.

3.3.2 Model Updating

In Algorithm 3.1, we only update Y_{k+1}, and hence $\mathbf{m}_k$ and m_k, on successful steps. However, in our implementation, we always try to incorporate new information when it becomes available, and so we update $Y_{k+1} = Y_k \cup \{\mathbf{x}_k + \mathbf{s}_k\} \setminus \{\mathbf{y}_t\}$ on all iterations except when the safety phase is called (since in the safety phase we never evaluate $\mathbf{r}(\mathbf{x}_k + \mathbf{s}_k)$).

Regardless of how often we update the model, we need some criterion for selecting the point $\mathbf{y}_t \in Y_k$ to replace with $\mathbf{y}^+ := \mathbf{x}_k + \mathbf{s}_k$. There are three common ways to choose a particular point to remove from the interpolation set:

Furthest Point: The point is the furthest away from the current iterate;

Optimal Λ-poisedness: Replacing the point with $\mathbf{y}^+$ would give the maximum improvement in the Λ-poisedness of Y_k. That is, choose the t for which $|\ell_t(\mathbf{y}^+)|$ is maximised;

Stable Update: Replacing the point with $\mathbf{y}^+$ would induce the most stable update to the interpolation system (3.6). As introduced by Powell [182] for quadratic models, moving $\mathbf{y}_t$ to $\mathbf{y}^+$ induces a low-rank update of the matrix $W \to W_{\text{new}}$ in the interpolation system, here (3.6). From the Sherman-Morrison-Woodbury formula, this induces a low-rank update of $H = W^{-1}$, which has the form

$$H_{\text{new}} \leftarrow H + \frac{1}{\sigma_t}\left[A_t B_t^\top\right], \tag{3.14}$$

for some $\sigma_t \neq 0$ and low rank $A_t B_t^\top$. Under this measure, we would want to replace a point in the interpolation set when the resulting $|\sigma_t|$ is maximal; i.e. the update (3.14) is 'stable'. In [182], it is shown that, for underdetermined quadratic interpolation, $\sigma_t \geq \ell_t(\mathbf{y}^+)^2$.

Two approaches for selecting $\mathbf{y}_t$ combine these reasons into a single criterion. Firstly in BOBYQA, the point t is chosen by combining the 'furthest point' and 'stable update' measures:

$$t = \underset{j=0,\ldots,n}{\arg\max} \left\{ |\sigma_j| \max\left(\frac{\|\mathbf{y}_j - \mathbf{x}_k\|^4}{\Delta_k^4}, 1\right)\right\}. \tag{3.15}$$

Alternatively, Scheinberg and Toint [203] combine the 'furthest point' and 'optimal Λ-poisedness' measures:

$$t = \underset{j=0,\ldots,n}{\arg\max} \left\{ |\ell_j(\mathbf{y}^+)| \, \|\mathbf{y}_j - \mathbf{x}_k\|^2 \right\}. \tag{3.16}$$

In DFO-LS, we combine the 'furthest point' and 'optimal Λ-poisedness' in a variation of the BOBYQA criterion (3.15), and select the point to remove $\mathbf{y}_t$ using

$$t = \underset{j=0,\ldots,n}{\arg\max} \left\{ |\ell_j(\mathbf{y}^+)| \cdot \max\left(\frac{\|\mathbf{y}_j - \mathbf{x}_x\|^4}{\Delta_k^4}, 1\right)\right\}, \tag{3.17}$$

with ties broken with lexicographic ordering. However, as we now show, in DFO-LS with interpolation models (the usual case $p = n$), the two measures 'optimal Λ-poisedness' and 'stable update' coincide, meaning our framework allows a unification of the perspectives from [186] and [203], rather than having the indirect relationship via the bound $\sigma_t \geq \ell_t(\mathbf{y}^+)^2$.

To this end, suppose $p = n$, define W as the matrix in (3.6), and let $H := W^{-1}$. The Lagrange polynomials for Y_k can then be written as

$$\ell_t(\mathbf{y}) = 1 + \mathbf{g}_t^\top(\mathbf{y} - \mathbf{y}_t), \tag{3.18}$$

where $\mathbf{g}_t$ solves

$$W\mathbf{g}_t = \begin{bmatrix} \ell_t(\mathbf{y}_1) - \ell_t(\mathbf{x}_k) \\ \vdots \\ \ell_t(\mathbf{y}_n) - \ell_t(\mathbf{x}_k) \end{bmatrix} = \begin{cases} \mathbf{e}_t, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ -\mathbf{e}, & \text{if } \mathbf{y}_t = \mathbf{x}_k, \end{cases} \tag{3.19}$$

where $\mathbf{e}_t$ is the usual coordinate vector in $\mathbb{R}^n$ and $\mathbf{e} := [1 \cdots 1]^\top \in \mathbb{R}^n$. This gives us the relations

$$\ell_t(\mathbf{y}^+) = \begin{cases} 1 + (H\mathbf{e}_t)^\top(\mathbf{y}^+ - \mathbf{y}_t), & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ 1 - (H\mathbf{e})^\top(\mathbf{y}^+ - \mathbf{x}_k), & \text{if } \mathbf{y}_t = \mathbf{x}_k. \end{cases} \tag{3.20}$$

Now, we consider the 'stable update' measure. We will update the point $\mathbf{y}_t$ to $\mathbf{y}^+$, which will give us a new matrix W_{new} with inverse H_{new}. This change induces a rank-1 update from W to W_{new}, given by

$$W_{\text{new}} = W + \begin{cases} \mathbf{e}_t(\mathbf{y}^+ - \mathbf{y}_t)^\top, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ \mathbf{e}(\mathbf{x}_k - \mathbf{y}^+)^\top, & \text{if } \mathbf{y}_t = \mathbf{x}_k. \end{cases} \tag{3.21}$$

By the Sherman-Morrison-Woodbury formula, this induces a rank-1 update from H to H_{new}, given by

$$H_{\text{new}} = H - \frac{1}{\sigma_t} \begin{cases} H\mathbf{e}_t(\mathbf{y}^+ - \mathbf{y}_t)^\top H, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ H\mathbf{e}(\mathbf{x}_k - \mathbf{y}^+)^\top H, & \text{if } \mathbf{y}_t = \mathbf{x}_k. \end{cases} \tag{3.22}$$

For a general rank-1 update $W_{\text{new}} = W + \mathbf{u}\mathbf{v}^\top$, the denominator is $\sigma = 1 + \mathbf{v}^\top W^{-1}\mathbf{u}$, and so here we have

$$\sigma_t = \begin{cases} 1 + (\mathbf{y}^+ - \mathbf{y}_t)^\top H\mathbf{e}_t, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ 1 + (\mathbf{x}_k - \mathbf{y}^+)^\top H\mathbf{e}, & \text{if } \mathbf{y}_t = \mathbf{x}_k, \end{cases} \tag{3.23}$$

and hence $\sigma_t = \ell_t(\mathbf{y}^+)$, as expected.

We note that the same relationship $\sigma_t = \ell_t(\mathbf{y}^+)$ is not guaranteed for regression models, as the equivalent of the Sherman–Morrison formula for pseudoinverses is more complex [151].

3.3.3 Termination Criteria

The specification in Algorithm 3.1 includes some termination criteria, but more are needed for a practical implementation. In our implementation, there are five ways in which DFO-LS can terminate, the first three of which are present in DFBOLS, the implementation from [240].

1. Small objective value: since we always have the lower bound $f \geq 0$, we terminate when $f(\mathbf{x}_k) \leq \max\{\epsilon_{\text{abs}}, \epsilon_{\text{rel}} f(\mathbf{x}_0)\}$, for non-negative user-specified parameters ϵ_{abs} and ϵ_{rel} (default $\epsilon_{\text{abs}} = 10^{-12}$ and $\epsilon_{\text{rel}} = 10^{-20}$). Having this feature is especially useful for DFO solvers, when often just achieving some desired decrease in the objective is the goal, rather than solving to full optimality (e.g. when function evaluations are expensive);

2. Small trust region: we know that $\rho_k \to 0$ as $k \to \infty$ from Lemma 4.11, so we terminate when $\rho_k \leq \rho_{\text{end}}$ (default $\rho_{\text{end}} = 10^{-8}$);

3. Computational budget: we terminate after a given number of evaluations of the objective (default $\min(100(n+1), 1000)$);

4. Slow decrease in the objective: described below; and

5. All objective values are within noise level (if `NOISY=True` in Algorithm 3.1): more details below.

The second and third of these are designed to cause termination after a sufficient number of unsuccessful steps. The first criteria is triggered by successful steps, but is likely to be triggered only for zero-residual problems. The fourth is designed to ensure a termination based on successful steps for all problems (not just zero-residual), using an approach similar to the "$f_i^{*\prime}$ test" of Larson and Wild [130].

We define a successful iteration as 'slow' if the last K successful iterations have produced an average reduction in $\log(f(\mathbf{x}_k))$ below a given threshold. That is, if $\{k_i : i \in \mathbb{N}\}$ are the successful iterations, then iteration k_i is 'slow' if

$$\frac{\log(f(\mathbf{x}_{k_{(i-K)}})) - \log(f(\mathbf{x}_{k_i}))}{K} < \epsilon, \tag{3.24}$$

for some user-specified value $\epsilon > 0$. Note that since we are only considering successful iterations, $f(\mathbf{x}_{k_i})$ will be the best objective value found up to iteration k_i. Our termination condition is then: quit after successful iteration k_i if $\{k_{i-N+1}, \ldots, k_{i-1}, k_i\}$ were all 'slow', for some $N \in \mathbb{N}$. The default values are $K = 5$, $\epsilon = 10^{-4}$ and $N = 20n$.

For the final condition, DFO-LS also includes optional noise-aware termination. Specifically, we terminate if all (noisy) function values $\widetilde{f}(\mathbf{y}_t)$ are within some user-provided 'noise level' of $\widetilde{f}(\mathbf{x}_k)$. That is, for all $t = 1, \ldots, p$, either

$$|\widetilde{f}(\mathbf{y}_t) - \widetilde{f}(\mathbf{x}_k)| \leq \text{const} \cdot \frac{\epsilon}{\sqrt{N_t}} \qquad \text{or} \qquad \left|\frac{\widetilde{f}(\mathbf{y}_t)}{\widetilde{f}(\mathbf{x}_k)}\right| \leq \text{const} \cdot \frac{\epsilon}{\sqrt{N_t}}, \tag{3.25}$$

where N_t is the number of samples used to estimate the value $\widetilde{f}(\mathbf{y}_t)$; see Section 3.4.3.1 for details. Which of these criteria is used depends on whether the user has specified ϵ as an additive or multiplicative noise level in the evaluation of $\widetilde{f}$, and the value of 'const' is also user-provided (default is 1).

3.3.4 Linear Algebra Implementation Details

DFO-LS requires the solution of the interpolation system (3.6) for m (different) right-hand sides at each iteration. We implement this in DFO-LS with a preprocessing step, where we compute a QR factorisation of the matrix W, and a solve step, where we use back substitution to solve the system for each right-hand side. In the case where $p = n$, the

preprocessing step could alternatively be implemented using low-rank updates of $W^\dagger = W^{-1}$ whenever an interpolation point is changed, similar to Powell's approach [182] for quadratic residual interpolation models. In the case of linear residual models with $p = n$, changing one interpolation point causes a rank-1 update of W and hence of W^{-1} (see Section 3.3.2), so this approach would give W^{-1} with a cost of $\mathcal{O}(n^2)$ floating-point operations per iteration, as opposed to the $\mathcal{O}(n^3)$ per-iteration cost of a QR factorisation.

Using either the factorisation preprocessing step or low-rank updates, solving (3.6) with m right-hand sides gives a per-iteration cost for the solve step of $\mathcal{O}(mn^2)$. In nonlinear least-squares problems, we generally have $m \geq n$, so this cost dominates the preprocessing cost, regardless of the approach used. We use the factorisation approach in DFO-LS because of its simplicity and its easy extension to underdetermined and regression models.

By contrast, for underdetermined quadratic residual models with $p + 1 = \mathcal{O}(n)$ interpolation points (e.g. $n + 2$ or $2n + 1$), the resulting linear system has size $p + n + 2$. This means that the preprocessing cost is $\mathcal{O}((p + n)^3)$ for the factorisation approach or $\mathcal{O}((p + n)^2)$ for the low-rank update approach, and the solve step has cost $\mathcal{O}(m(p+n)^2)$. As a result, if m is not too large compared to n, a factorisation-based approach would give a worse per-iteration cost than the low-rank update approach, so there is a benefit to using low-rank update methods. The software DFBOLS [240] uses quadratic models with the low-rank update approach.

We note that if DFO-LS is used with reduced initialisation cost, there is an extra cost of $\mathcal{O}(mn^2)$ per iteration from computing the SVD of J_k (see Section 3.2.2). This cost is, up to a constant factor, the same as the solve step for (3.6), and is only present while we have fewer than $n + 1$ interpolation points.

3.3.5 Other Implementation Differences

Addition of bound constraints In practice, DFO-LS solves the bound-constrained problem (2.15) as well as the unconstrained problem (2.6), as it is important to practical applications. We note that we treat the bound constraints as unrelaxable—every objective evaluation requested by DFO-LS will be at a feasible point (including both iterates and geometry-improving points). Our handling of bound constraints requires no change to the logic as specified in Algorithm 3.1, but does require the addition of the same bound constraints in the algorithms for the trust-region subproblem (1.6) and calculating geometry-improving steps (Section 3.3.1). For the trust-region subproblem, there is an equivalent of the Cauchy decrease requirement (Assumption 4.4) for bound-constrained problems based on a projected gradient step [53, Section 12.2]. In practice, we use the routine from DFBOLS, which is itself a slight modification of the routine from BOBYQA (which was specifically

designed to produce feasible iterates in the presence of bound constraints).[5] This routine is efficient, requiring only $\mathcal{O}(n^2)$ work per iteration—and often $\mathcal{O}(1)$ iterations, although up to $\mathcal{O}(n)$ iterations is possible [186].

Calculating geometry-improving steps with bound constraints is easier, since the Lagrange polynomials are linear rather than quadratic, and so we need to maximise a linear objective subject to Euclidean ball and bound constraints. We use our own routine for this, which handles the bound constraints via an active set method; see Appendix B for full details.

Because bound constraints can often provide information about the natural scaling of a problem, DFO-LS allows the optional internal scaling of variables based on the bound constraints, to reduce the likelihood of ill-conditioning. If this is used, we internally shift and scale the inputs so that the new feasible region is $\mathbf{x} \in [0, 1]^n$.

We also note that DFO-LS is scale-invariant, in the sense that replacing $\mathbf{r}(\mathbf{x})$ by $c \cdot \mathbf{r}(\mathbf{x})$ in (2.6) for any $c > 0$ produces the same sequence of iterates. Similarly, replacing $\mathbf{r}(\mathbf{x})$ with $\mathbf{r}(c\,\mathbf{x})$ for any $c > 0$ produces the same sequence of iterates, provided the trust-region parameters are modified appropriately.[6]

Other differences The following changes, which are from BOBYQA, are present in the implementation of DFO-LS:

- We accept any step (i.e. set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$) where we see an objective reduction—that is, when $R_k > 0$. In fact, we always update $\mathbf{x}_k$ to be the best value found so far, even if that point came from a geometry-improving phase rather than a trust region step;
- The reduction of ρ_k in an unsuccessful step (line 36) only occurs when $R_k < 0$;
- Since we update the model on every iteration, we only allow ρ_k to be reduced after 3 consecutive iterations with both ρ_k at its current level, and $\|\mathbf{s}_k\| \leq \rho_k$; i.e. we only reduce ρ_k when Δ_k is small and after the model has been updated several times (reducing the likelihood of the unsuccessful steps being from a bad interpolating set);

[5] This routine satisfies the sufficient model decrease condition from [53] if $\mathbf{x}_k$ is not too close to the boundary of the feasible region in the direction $-\mathbf{g}_k$. However, the routine does not follow the projected gradient path [53, Section 12.2.1]; although it initially searches in the direction $-\mathbf{g}_k$, if a bound constraint is hit before the trust-region boundary or the (unconstrained) Cauchy step, its next search direction is based on $-\nabla m_k$ at the current step, rather than $-\mathbf{g}_k$. Thus, it is difficult to check the sufficient decrease condition in general.

[6] Specifically, by scaling Δ_0^{init}, $\Delta_{\max}$ and ρ_{end} by c^{-1}. A similar scale invariance to $\mathbf{r}(c\,\mathbf{x})$ holds if $c < 0$, with a trust-region scaling of $|c|^{-1}$, except that DFO-LS would use a different initial interpolation set.

- The method for reducing ρ_k is usually given by $\rho_{k+1} = \alpha_1 \rho_k$, but it changed when ρ_k approaches ρ_{end}:

$$\rho_{k+1} = \begin{cases} \alpha_1 \rho_k, & \text{if } \rho_k > 250\rho_{\text{end}}, \\ \sqrt{\rho_k \rho_{\text{end}}}, & \text{if } 16\rho_{\text{end}} < \rho_k \leq 250\rho_{\text{end}}, \\ \rho_{\text{end}}, & \text{if } \rho_k \leq 16\rho_{\text{end}}; \end{cases} \tag{3.26}$$

- In some calls of the safety phase, we only reduce ρ_k and Δ_k, without improving the geometry of Y_k (as per Section 3.3.1); and

- We construct the initial interpolation set Y_0 to be $\{\mathbf{x}_0, \mathbf{x}_0 + \Delta_0^{\text{init}} \mathbf{d}_1, \ldots, \mathbf{x}_0 + \Delta_0^{\text{init}} \mathbf{d}_p\}$ where $\{\mathbf{d}_1, \ldots, \mathbf{d}_{\min(n,p)}\}$ is a randomly-generated orthonormal set (and $\mathbf{d}_{n+1}, \ldots, \mathbf{d}_p$ are random unit vectors if $p > n$).

3.4 Features of DFO-LS for the Noisy Regime

We now outline the features of DFO-LS aimed specifically at situations where the objective has noise.

3.4.1 Default Parameters for Noisy Problems

One of the main problem types that DFO-LS is designed to solve is where objective evaluations are noisy. In this situation, the set of default parameters—which are designed for smooth objectives—are not necessarily good choices.

The most notable examples of this are the parameters that govern decreases of Δ_k and ρ_k, namely γ_{dec}, α_1 and α_2 (default values 0.5, 0.1 and 0.5 respectively; see Remark 3.2). When we have noisy evaluations, we may get unsuccessful iterations even when a step $\mathbf{s}_k$ is useful, because the noise in the objective evaluation leads to inaccuracies in the calculated R_k (3.3). This leads to unnecessary reductions in the trust region radius, causing the algorithm to progress more slowly, and potentially terminate too early.

In DFO-LS, we allow the user to specify if their objective evaluation is noisy, and consequently modify the default values for several algorithm parameters. Note that the user can choose to override any parameter value by specifying it directly, even if the default has been modified. The 'noisy problem' default values of γ_{dec}, α_1 and α_2 are 0.98, 0.9 and 0.95 respectively.

3.4.2 Multiple Restarts

To improve robustness to noisy objectives, DFO-LS uses a multiple restarts mechanism. Although this feature is novel in the model-based DFO setting, similar techniques have

been commonly used in numerical analysis, such as multiple restarts of nonlinear conjugate gradient methods [164, Chapter 5] and GMRES [69, Chapter 6], as well as for robustness improvement of the Nelder-Mead algorithm [116]. There are also connections to multistart methods, where several runs of an algorithm are initialised from different starting points, a technique common in global optimisation [140, 103], and also used for local derivative-free optimisation (e.g. [64] or [12, Example 3.5]). In contrast, our multiple restarts technique only ever uses a single initialisation point for a run, and employs the final iterate of a run as the starting point for the restarted run.

To motivate this mechanism, we note that, in trust-region DFO methods, Δ_k tends to zero as a measure of convergence (e.g. Lemma 4.11 for the deterministic case). However, when the function is noisy, as Δ_k gets small, the interpolation points get very close together and the corresponding objective values are all within noise level. As a result, Δ_k no longer reflects convergence and the solver can stagnate in a suboptimal region.

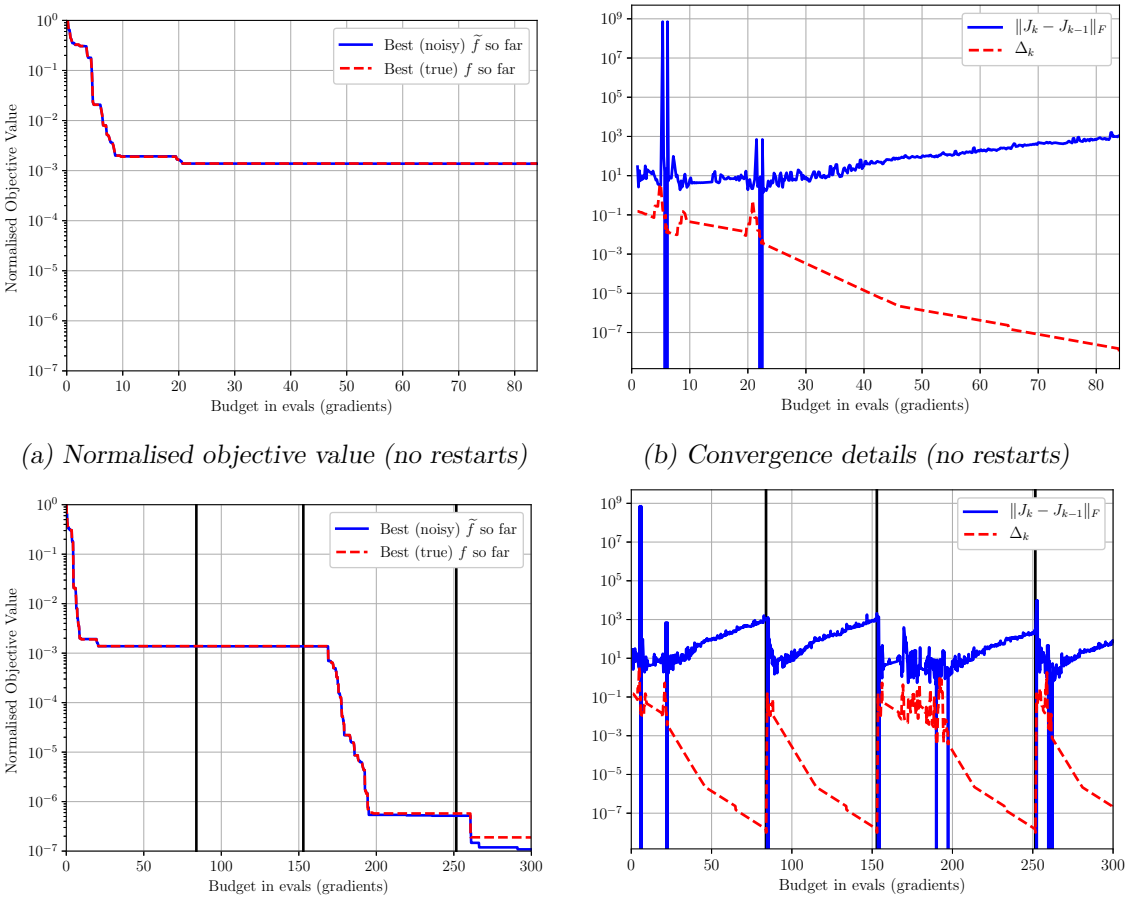
An illustrative example. We may see this effect by considering a test problem. In Figure 3.1, we compare two runs of DFO-LS—with and without multiple restarts—for the Osborne 1 test problem (#36 in (MW)), where we have added unbiased multiplicative Gaussian noise with $\sigma = 10^{-2}$. After making some initial progress, the run without restarts has many unsuccessful iterations, and Δ_k shrinks as the solver attempts to find a descent direction. When this happens, the interpolated Jacobian J_k begins to change substantially at each iteration. This indicates that the noise in the interpolation problem is dominating the true descent information.

When we introduce restarts, the stagnation can eventually be overcome. When a restart occurs (and we increase Δ_k to its original level), the changes in J_k reduce quickly, and so the interpolation is more likely to capture genuine information about changes in the objective. As a result, the solver is able to progress, and ultimately finds a much higher accuracy solution. □

In DFO-LS, a restart is triggered by all the termination criteria, except for small objective and maximum computational budget. At its simplest, a restart involves increasing Δ_k to a much larger value, and possibly moving some of the points in Y_k. There are two main types of restart that DFO-LS can perform:

Hard restart: Reset the trust region radius to $\Delta_{k+1}^{\text{init}} = \rho_{k+1}^{\text{init}} = \Delta_0^{\text{init}}$, and rebuild Y_{k+1} in the new (larger) trust region $B(\mathbf{x}_k, \Delta_{k+1}^{\text{init}})$ from scratch using the same mechanism as how Y_0 was originally constructed; and,

Soft restart (moving $\mathbf{x}_k$): Reset the trust region radius to $\Delta_{k+1}^{\text{init}} = \rho_{k+1}^{\text{init}} = \Delta_0^{\text{init}}$, and save the current best point $\mathbf{x}_k$ separately. Then, move $\mathbf{x}_k$ to a geometry-improving point,



(a) Normalised objective value (no restarts)

(b) Convergence details (no restarts)

(c) Normalised objective value (with restarts)

(d) Convergence details (with restarts)

Figure 3.1. Normalised objective value achieved by DFO-LS, measured in both the noisy and true objective, and convergence information for test problem 'Osborne 1' with $(n, m) = (5, 33)$ and unbiased multiplicative Gaussian noise of size $\sigma = 10^{-2}$. The vertical black lines indicate when restarts occurred. Restart type was 'soft (moving $\mathbf{x}_k$)', and the budget was $300(n + 1)$ evaluations; the (a) and (b) runs terminated early on small trust-region radius.

$\widetilde{\mathbf{x}}_k$, in $B(\mathbf{x}_k, \Delta^{\text{init}}_{k+1})$. Finally, move the $N - 1 < p$ points in Y_k that were closest to the old value of $\mathbf{x}_k$ to geometry-improving points in the new (larger & shifted) trust region $B(\widetilde{\mathbf{x}}_k, \Delta^{\text{init}}_{k+1})$ as per Section 3.3.1. The iteration then continues from whichever of these N new points has the least objective value, which may be worse than the value from the end of the previous iteration. The final solution returned by the solver takes the optimal value seen so far, including the saved endpoints from previous restarts.

The soft restart approach with $N = \min(3, p)$ is the default approach in DFO-LS.

We see that soft restarts require the objective to be evaluated $N < p$ times, whereas hard restarts require a full p objective evaluations (as we have changed all interpolation points except $\mathbf{x}_k$). We also note that the soft restart mechanism is intrinsically linked to the model-based DFO framework, and there is not a clear derivative-based equivalent of this

procedure.

In DFO-LS, when restarts are used, we add an extra termination criterion: we terminate if the last M consecutive restarts have not achieved any objective reduction (default $M = 10$).

Auto-detection of restarts As discussed above, we saw in Figure 3.1 that the need for a restart can be determined by a series of unsuccessful iterations, coupled with large changes in J_k, with this change increasing rapidly with k. By contrast, selecting an a priori value of ρ_{end} which provides a timely trigger for restarts is not straightforward. Thus, DFO-LS uses previous iteration information to auto-detect when a restart is needed. A restart is triggered if, in the last N iterations (default is $N = 30$):

- The trust-region radius Δ_k has never been increased, and it has been decreased on at least twice as many iterations as it has been kept constant; and,
- The slope and correlation coefficient of a linear fit through the points $\{(k, \log \|J_k - J_{k-1}\|_F)\}$ exceed given thresholds; that is, $\|J_k - J_{k-1}\|_F$ is consistently increasing at a given exponential rate.[7]

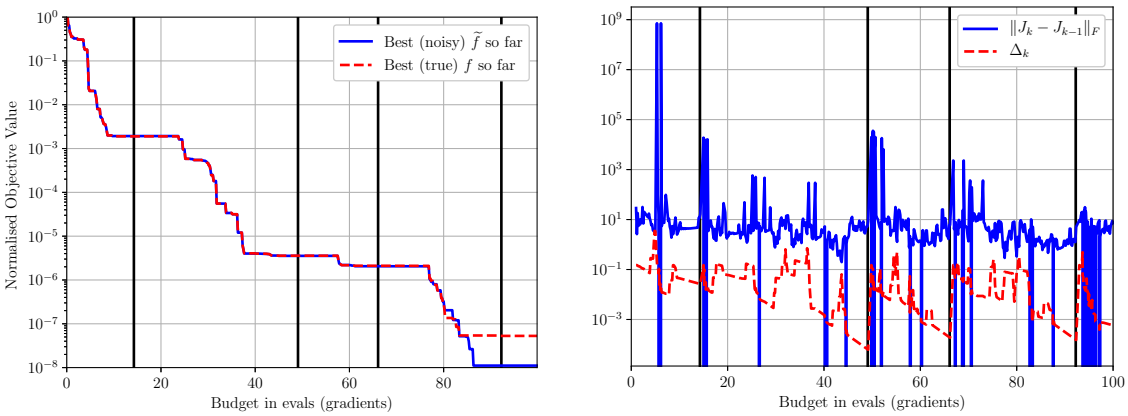
An illustrative example, revisited. In Figure 3.2, we see the same results as in Figure 3.1, but with auto-detection of when to restart. Because of the auto-detection, restarts are triggered much earlier (avoiding the iterations during which no progress was made), and we achieve accuracy $\tau = 10^{-6}$ after approximately $15(n + 1)$ evaluations, rather than approximately $80(n + 1)$ evaluations without auto-detection. □

Alternative Restart Mechanism DFO-LS has another approach available for performing soft restarts.

Soft restart (fixed $\mathbf{x}_k$): Reset the trust region radius to $\Delta_{k+1}^{\text{init}} = \rho_{k+1}^{\text{init}} = \Delta_0^{\text{init}}$, and move the $N < p$ points $\mathbf{y}_t \neq \mathbf{x}_k$ in Y_k that are closest to $\mathbf{x}_k$ to geometry-improving points in the new trust region $B(\mathbf{x}_k, \Delta_{k+1}^{\text{init}})$ as per Section 3.3.1. The iteration then continues from whichever of these N new points has the least objective value.

As we will see in Section 5.2, the soft restart (fixed $\mathbf{x}_k$), with the default value $N = \min(3, p)$, performs noticeably worse than the 'moving $\mathbf{x}_k$' version of soft restarts.

[7] This condition is similar to the noise detection mechanism from [16], which we became aware of after the condition had been chosen.



(a) Normalised objective value *(b) Convergence details*

Figure 3.2. Normalised objective value achieved by DFO-LS, measured in both the noisy and true objective, and convergence information as per Figure 3.1, but allowing auto-detection of restarts. The budget was $100(n+1)$ objective evaluations. The difference between the 'noisy' and 'true' objective reduction measures is discussed in Section 5.2.1.

3.4.3 Optional Features for Noisy Objectives

Aside from multiple restarts, DFO-LS also implements the two most common approaches for handling noisy objectives in model-based DFO: sample averaging and regression models. In Appendix C, we argue, briefly and in a simplified framework, that regression and sample averaging generate similar model error; thus, since sample averaging produces a better estimate of objective decrease for fixed noise level, we expect that overall, regression will be slightly less robust compared to sample averaging when considering large computational budgets.

In Section 5.2, we show numerically that using multiple restarts gives better performance than averaging and regression. Therefore, the latter features are not used in DFO-LS by default.

3.4.3.1 Sample Averaging

Sample averaging replaces evaluations of the noisy objective $f(\mathbf{x})$ with an average of N samples. We note that this is only a useful technique if the noise in the objective is stochastic. In the least-squares case, we replace an evaluation of the noisy objective $\overline{\mathbf{r}}$ with the sample average

$$\overline{\mathbf{r}}_N(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}(\mathbf{x}, \xi_i), \tag{3.27}$$

where $\xi_1, \ldots, \xi_N$ are different realisations of the random variable ξ defining the noise.

In convergence analysis involving sample averaging, one must usually choose N to be at least Δ_k^{-4} (e.g. [50]). However, as Δ_k can easily be of size 10^{-2} or 10^{-3}, this amount of

averaging rapidly becomes impractical, so choosing N to be of size Δ_k^{-1}, for instance, is a more sensible choice in practice (e.g. Algorithm TR-SAA in [50]).

In the implementation of DFO-LS, the use of sample averaging is governed by a user-specified function that allows for a wide range of sample averaging techniques:

$$N = \text{nsamples}(\rho_k, \Delta_k, k, n_{\text{restarts}}), \tag{3.28}$$

where $n_{\text{restarts}} \in \{0, 1, 2, \ldots\}$ is the number of restarts that the solver has performed (see Section 3.4.2 for details) and $k \in \{0, 1, 2, \ldots\}$ is the iteration number since the most recent restart. The default option for nsamples gives $N \equiv 1$ always; i.e. no sample averaging.

3.4.3.2 Regression Models for Noisy Objectives

Building regression models rather than interpolation models requires having more interpolation points than degrees of freedom in the model [58, 30, 50]. Our formulation of the DFO-LS model construction problem (3.5) allows linear regression models, when $p > n$. It remains to consider how to evolve the set Y_k at each iteration.

DFO-LS has three mechanisms for moving multiple points on successful iterations, which can be used alongside regression models:

Nothing: Replace one point in Y_k with $\mathbf{x}_{k+1}$, and nothing else;

Geometry-based: Replace one point in Y_k with $\mathbf{x}_{k+1}$, and then one-by-one[8] move the N points in Y_k that are furthest from Y_{k+1} to geometry-improving locations given by (2.52) with $B = B(\mathbf{x}_{k+1}, \Delta_{k+1})$; and

Momentum-based: Replace one point in Y_k with $\mathbf{x}_{k+1}$ to form Y_{k+1}, and then move the N points in Y_{k+1} that are furthest from $\mathbf{x}_{k+1}$ to $\mathbf{x}_{k+1} + \Delta_{k+1}\mathbf{d}$, where $\mathbf{d}$ is a random unit vector with[9] $\mathbf{d}^\top \mathbf{s}_k > 0$.

The last two mechanisms above, that replace multiple points, try to mimic/match the situation arising in sample averaging, where moving one point affects c function values, where c is the sampling rate.

The first mechanism ('nothing') is the default in DFO-LS, when regression models are used; they are compared with each other, and against sample averaging, in Section 5.2.4.

[8] That is, (2.52) is calculated sequentially, after the previous point has been moved (and Lagrange polynomials recalculated).

[9] When using bound constraints, if $\Delta_{k+1}\mathbf{d}$ gives a point outside the bounds, we instead take $\alpha\mathbf{d}$ for some $\alpha \in (0, \Delta_{k+1})$ such that the bounds are satisfied. If this requires $\alpha < 10^{-3}$, we replace $\mathbf{d}$ with $-\mathbf{d}$, sacrificing the requirement that $\mathbf{d}^\top \mathbf{s}_k > 0$.

Chapter 4

Theoretical Guarantees for DFO-LS

In this chapter, we prove global convergence of DFO-LS, and provide a worst-case complexity analysis. To do this, we first outline the connection between Λ-poisedness of Y_k and fully linear models (Section 4.1). We then prove global convergence of a simplified version of Algorithm 3.1 (i.e. convergence from any starting point $\mathbf{x}_0$) to first-order critical points in Section 4.2, and determine its worst-case complexity in Section 4.3.

Algorithm Simplification The theory developed in this chapter applies to DFO-LS (Algorithm 3.1) for solving (2.6), under the following simplifying assumptions on the algorithm:

- The flag `NOISY` is set to `False`;
- Termination conditions are not applied: $\rho_{\text{end}} = 0$ and no termination on slow objective decrease (line 30 of Algorithm 3.1); and
- Interpolation or regression models are used with no growing phase: $p_{\text{init}} = p \geq n$.

We will also require sufficient smoothness of our objective, as given by Assumption 2.4.

Novel Contributions of Theoretical Analysis The global convergence theory for DFO-LS is similar to that of DFLS [240], restricted to the case of linear residual models. However, our analysis is more closely aligned with the standard presentation of convergence for model-based trust-region DFO (e.g. [60, Chapter 10]). In addition, we weaken the assumption in [240] that the trust-region subproblem be solved to full optimality, and instead allow inexact solutions (using a standard assumption), and prove convergence of the whole sequence of gradients rather than just a subsequence. We also introduce a convergence result for the criticality phase that is stronger than the standard result (Lemma 2.28; see Section 2.3). This allows us to further extend the analysis from [240] to include a new

worst-case complexity analysis. In this analysis, we show that DFO-LS has the complexity of standard first-order methods, but with a dependency on dimension closer to that of second-order methods. This aligns with our expectations from the classical Gauss-Newton setting, where we construct models that capture some, but not all, curvature information, as per (1.20).

4.1 Linear Residual Models are Fully Linear

The first result we need is that our linear residual model (3.1) and corresponding objective model (3.2) give fully linear models for $\mathbf{r}$ and f respectively. However, we have so far only defined fully linear models in the context of scalar functions (Definition 2.7), so we need a definition that applies to vector functions. We use an analogous definition to the scalar case, as in [90], which is equivalent to the definition in [83] up to a change in constants.

Definition 4.1 (Fully linear, vector function)**.** *A model $\mathbf{m}(\mathbf{s})$ for a function $\mathbf{r}(\mathbf{x}+\mathbf{s})$ is fully linear in $B(\mathbf{x}, \Delta)$ if they are both continuously differentiable and there exist constants $\kappa_{\text{ef}}^{\text{r}}, \kappa_{\text{eg}}^{\text{r}} > 0$, independent of $\mathbf{x}$ and Δ, so that*

$$\|\mathbf{r}(\mathbf{x}+\mathbf{s}) - \mathbf{m}(\mathbf{s})\| \leq \kappa_{\text{ef}}^{\text{r}} \Delta^2, \tag{4.1}$$

$$\|J(\mathbf{x}+\mathbf{s}) - \nabla \mathbf{m}(\mathbf{s})\| \leq \kappa_{\text{eg}}^{\text{r}} \Delta, \tag{4.2}$$

for all $\|\mathbf{s}\| \leq \Delta$, where $\nabla \mathbf{m}$ is the Jacobian of $\mathbf{m}$.

We now state the connection between Λ-poisedness of Y_k and full linearity of the models $\mathbf{m}_k$ (2.6) and m_k (3.2). This result holds both for the usual case of linear interpolation models (with $p = n$, where Λ-poisedness is given by Definition 2.11) and linear regression models (with $p > n$, where Definition 2.16 applies).

Lemma 4.2. *Suppose Assumption 2.4 holds and Y_k is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$ with $|Y_k| = p \geq n$ and and $Y_k \subset B(\mathbf{x}_k, \Delta_k)$. Then $\mathbf{m}_k$ (2.6) is a fully linear model for $\mathbf{r}$ in $B(\mathbf{x}_k, \Delta_k)$ in the sense of Definition 4.1 with*

$$\kappa_{\text{ef}}^{\text{r}} = 2\kappa_{\text{eg}}^{\text{r}} \qquad \text{and} \qquad \kappa_{\text{eg}}^{\text{r}} = \frac{1}{2} L_J \left((p+1)\Lambda + 2\right). \tag{4.3}$$

Under the same hypotheses, m_k (3.2) is a fully linear model for f in $B(\mathbf{x}_k, \Delta_k)$ in the sense of Definition 2.7 with

$$\kappa_{\text{ef}} = \kappa_{\text{eg}} + L_{\nabla f}/2 + \kappa_{\text{eg}}^{\text{r}} \left(r_{\max} + \kappa_{\text{eg}}^{\text{r}} \Delta_{\max}/2\right) + (\kappa_{\text{eg}}^{\text{r}} \Delta_{\max} + J_{\max})^2, \tag{4.4}$$

$$\kappa_{\text{eg}} = L_{\nabla f} + J_{\max} \kappa_{\text{eg}}^{\text{r}} \Delta_{\max} + \kappa_{\text{eg}}^{\text{r}} r_{\max} + (\kappa_{\text{eg}}^{\text{r}} \Delta_{\max} + J_{\max})^2, \tag{4.5}$$

where $L_{\nabla f}$ is from (2.9). We also have the bound $\|H_k\| \leq (\kappa_{\text{eg}}^{\text{r}} \Delta_{\max} + J_{\max})^2$, independent of $\mathbf{x}_k$, Y_k and Δ_k.

Proof. See Appendix A. □

4.2 Global Convergence of DFO-LS

We begin with some nomenclature to describe certain iterations: we call an iteration (for which the safety phase is not called)

- 'Successful' if $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$ (i.e. $R_k \geq \eta_1$), and 'very successful' if $R_k \geq \eta_2$. Let $\mathcal{S}$ be the set of successful iterations k;
- 'Model-Improving' if $R_k < \eta_1$ and the model-improvement phase is called (i.e. Y_k is not Λ-poised in $B(\mathbf{x}_k, \Delta_k)$); and
- 'Unsuccessful' if $R_k < \eta_1$ and the model-improvement phase is not called.

Beyond Assumption 2.4—which we use throughout to ensure that making the interpolation set Λ-poised ensures m_k is a fully linear model for f, via Lemma 4.2—we will also need some further assumptions.

Assumption 4.3. *We assume that $\|H_k\| \leq \kappa_H$ for all k, for some $\kappa_H \geq 1$.*[1]

Assumption 4.4. *Our method for solving the trust-region subproblem* (1.6) *gives a step $\mathbf{s}_k$ satisfying the sufficient ('Cauchy') decrease condition*

$$m_k(\mathbf{0}) - m_k(\mathbf{s}_k) \geq c_1 \|\mathbf{g}_k\| \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)}\right), \tag{4.6}$$

for some $c_1 \in [1/2, 1]$ independent of k.

This standard condition is not onerous, and can be achieved with $c_1 = 1/2$ by one iteration of steepest descent with exact linesearch applied to the model m_k [53].

Lemma 4.5 (Lemma 4.3, [240])**.** *Suppose Assumptions 4.3 and 4.4 hold. If the model m_k is fully linear in $B(\mathbf{x}_k, \Delta_k)$ and*

$$\Delta_k \leq c_0 \|\mathbf{g}_k\|, \qquad \text{where} \qquad c_0 := \min\left(\frac{c_1(1-\eta_2)}{2\kappa_{\text{ef}}}, \frac{1}{\kappa_H}\right), \tag{4.7}$$

then either the k-th iteration is very successful or the safety phase is called.

Proof. We compute

$$|R_k - 1| = \left|\frac{(f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)) - (m_k(\mathbf{0}) - m_k(\mathbf{s}_k))}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}\right|, \tag{4.8}$$

$$\leq \frac{|f(\mathbf{x}_k + \mathbf{s}_k) - m_k(\mathbf{s}_k)|}{|m_k(\mathbf{0}) - m_k(\mathbf{s}_k)|} + \frac{|f(\mathbf{x}_k) - m_k(\mathbf{0})|}{|m_k(\mathbf{0}) - m_k(\mathbf{s}_k)|}. \tag{4.9}$$

[1] Lemma 4.2 ensures Assumption 4.3 holds whenever Y_k is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$, but we need it to hold on all iterations. However, most of our analysis holds if this assumption is removed—see Remark 4.22 for details.

By assumption, $\Delta_k \leq \|\mathbf{g}_k\|/\kappa_H \leq \|\mathbf{g}_k\|/\max(\|H_k\|, 1)$. Applying this to (4.6), we have

$$m_k(\mathbf{0}) - m_k(\mathbf{s}_k) \geq c_1 \|\mathbf{g}_k\| \Delta_k. \tag{4.10}$$

Using this and fully linearity (2.16), we get

$$|R_k - 1| \leq 2\left(\frac{\kappa_{\text{ef}} \Delta_k^2}{c_1 \|\mathbf{g}_k\| \Delta_k}\right) \leq 1 - \eta_2. \tag{4.11}$$

Thus $R_k \geq \eta_2$ and the iteration is very successful if $\|\mathbf{s}_k\| \geq \gamma_S \rho_k$, otherwise the safety phase is called. □

The next result provides a lower bound on the size of the trust region step $\|\mathbf{s}_k\|$, which we will later use to determine that the safety phase is not called when $\|\mathbf{g}_k\|$ is bounded away from zero and Δ_k is sufficiently small. Note that [240, Lemma 4.4] shows that the safety phase is not called by requiring that the trust-region subproblem (1.6) is solved to global optimality, a stronger condition than Assumption 4.4.

Lemma 4.6. *Suppose Assumption 4.4 holds. Then the step* $\mathbf{s}_k$ *satisfies*

$$\|\mathbf{s}_k\| \geq c_2 \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)}\right), \tag{4.12}$$

where $c_2 := 2c_1/(1 + \sqrt{1 + 2c_1})$.

Proof. Let $h_k := \max(\|H_k\|, 1) \geq 1$. Since $m_k(\mathbf{0}) - m_k(\mathbf{s}_k) \geq 0$ from (4.6), we have

$$m_k(\mathbf{0}) - m_k(\mathbf{s}_k) = |m_k(\mathbf{0}) - m_k(\mathbf{s}_k)| = \left|\mathbf{g}_k^\top \mathbf{s}_k + \frac{1}{2}\mathbf{s}_k^\top H_k \mathbf{s}_k\right| \leq \|\mathbf{s}_k\| \cdot \|\mathbf{g}_k\| + \frac{h_k}{2}\|\mathbf{s}_k\|^2. \tag{4.13}$$

Substituting this into (4.6), we get

$$\frac{1}{2}\|\mathbf{s}_k\|^2 + \frac{\|\mathbf{g}_k\|}{h_k} \cdot \|\mathbf{s}_k\| - c_1 \frac{\|\mathbf{g}_k\|}{h_k} \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{h_k}\right) \geq 0. \tag{4.14}$$

For (4.14) to be satisfied, we require that $\|\mathbf{s}_k\|$ is larger than (or equal to) the positive root of the left-hand side of (4.14), which gives

$$\|\mathbf{s}_k\| \geq \frac{2c_1 C_k \min(\Delta_k, C_k)}{\sqrt{C_k^2 + 2c_1 C_k \min(\Delta_k, C_k)} + C_k} \geq \frac{2c_1 C_k \min(\Delta_k, C_k)}{\sqrt{(1 + 2c_1)C_k^2} + C_k}, \tag{4.15}$$

where $C_k := \|\mathbf{g}_k\|/h_k$; from which we recover (4.12). □

Next, we bound the number of iterations in Algorithm 2.2 (criticality phase); this is a stronger version of Lemma 2.28.

Lemma 4.7. *Suppose Assumption 2.4 holds and* $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$. *Then for any* $\mu > 0$ *and* $\omega_C \in (0,1)$, *the criticality phase (Algorithm 2.2) terminates in finite time with* Y_k Λ*-poised in* $B(\mathbf{x}_k, \Delta_k)$ *and* $\Delta_k \leq \mu\|\mathbf{g}_k\|$ *for any* $\mu > 0$ *and* $\omega_C \in (0,1)$. *We also have the bound*

$$\min\left(\Delta_k^{\text{init}}, \frac{\omega_C \epsilon}{\kappa_{\text{eg}} + 1/\mu}\right) \leq \Delta_k \leq \Delta_k^{\text{init}}. \tag{4.16}$$

Proof. First, suppose Algorithm 2.2 terminates on the first iteration. Then $\Delta_k = \Delta_k^{\text{init}}$, and the result holds. Otherwise, consider some iteration i where Algorithm 2.2 does not terminate; that is, where $\omega_C^{i-1}\Delta_k^{\text{init}} > \mu\|\mathbf{g}_k^{(i)}\|$. Then since $m_k^{(i)}$ is fully linear in $B(\mathbf{x}_k, \omega_C^{i-1}\Delta_k^{\text{init}})$, we have

$$\epsilon \leq \|\nabla f(\mathbf{x}_k)\| \leq \|\nabla f(\mathbf{x}_k) - \mathbf{g}_k^{(i)}\| + \|\mathbf{g}_k^{(i)}\| \leq \left(\kappa_{\text{eg}} + \frac{1}{\mu}\right)\omega_C^{i-1}\Delta_k^{\text{init}}, \tag{4.17}$$

or equivalently $\omega_C^{i-1} \geq \dfrac{\epsilon}{(\kappa_{\text{eg}} + 1/\mu)\Delta_k^{\text{init}}}$. That is, if termination does not occur on iteration i, we must have

$$i \leq 1 + \frac{1}{|\log \omega_C|}\log\left(\frac{(\kappa_{\text{eg}} + 1/\mu)\Delta_k^{\text{init}}}{\epsilon}\right), \tag{4.18}$$

so Algorithm 2.2 terminates finitely. We also have $\omega_C^{i-1}\Delta_k^{\text{init}} \geq \dfrac{\epsilon}{\kappa_{\text{eg}} + 1/\mu}$, which gives (4.16). □

Lemma 4.8. *In all iterations,* $\|\mathbf{g}_k\| \geq \min(\epsilon_C, \Delta_k/\mu)$. *Also, if Assumption 2.4 holds and* $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$ *then*

$$\|\mathbf{g}_k\| \geq \epsilon_g := \min\left(\epsilon_C, \frac{\epsilon}{1 + \kappa_{\text{eg}}\mu}\right) > 0. \tag{4.19}$$

Proof. Firstly, if the criticality phase is not called, then we must have $\|\mathbf{g}_k\| = \|\mathbf{g}_k^{\text{init}}\| > \epsilon_C$. Otherwise, we have $\Delta_k \leq \mu\|\mathbf{g}_k\|$. Hence $\|\mathbf{g}_k\| \geq \min(\epsilon_C, \Delta_k/\mu)$.

To show (4.19), we follow the proof of [60, Lemma 10.11] and first suppose $\|\mathbf{g}_k^{\text{init}}\| \geq \epsilon_C$. Then $\mathbf{g}_k = \mathbf{g}_k^{\text{init}}$ and (4.19) holds. Otherwise, the criticality phase is called and m_k is fully linear in $B(\mathbf{x}_k, \Delta_k)$ with $\Delta_k \leq \mu\|\mathbf{g}_k\|$. In this case, we have

$$\epsilon \leq \|\nabla f(\mathbf{x}_k)\| \leq \|\nabla f(\mathbf{x}_k) - \mathbf{g}_k\| + \|\mathbf{g}_k\| \leq \kappa_{\text{eg}}\mu\|\mathbf{g}_k\| + \|\mathbf{g}_k\|, \tag{4.20}$$

and so $\|\mathbf{g}_k\| \geq \epsilon/(1 + \kappa_{\text{eg}}\mu)$ and (4.19) holds. □

Lemma 4.9. *Suppose Assumptions 2.4, 4.3 and 4.4 hold. If* $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$ *for all* k, *then* $\rho_k \geq \rho_{\min} > 0$ *for all* k, *where*

$$\rho_{\min} := \min\left(\Delta_0^{\text{init}}, \frac{\omega_C\epsilon}{\kappa_{\text{eg}} + 1/\mu}, \frac{\alpha_1\epsilon_g}{\kappa_H}, \alpha_1\left(\kappa_{\text{eg}} + \frac{2\kappa_{\text{ef}}}{c_1(1-\eta_2)}\right)^{-1}\epsilon\right). \tag{4.21}$$

Proof. From Lemma 4.8, we know $\|\mathbf{g}_k\| \geq \epsilon_g > 0$ for all k. To find a contradiction, let $k(0)$ be the first k such that $\rho_k < \rho_{\min}$. That is, we have

$$\rho_0^{\text{init}} \geq \rho_0 \geq \rho_1^{\text{init}} \geq \rho_1 \geq \cdots \geq \rho_{k(0)-1}^{\text{init}} \geq \rho_{k(0)-1} \geq \rho_{\min} \qquad \text{and} \qquad \rho_{k(0)} < \rho_{\min}. \tag{4.22}$$

We first show that

$$\rho_{k(0)} = \rho_{k(0)}^{\text{init}} < \rho_{\min}. \tag{4.23}$$

From the criticality phase of Algorithm 3.1, we know that either $\rho_{k(0)} = \rho_{k(0)}^{\text{init}}$ or $\rho_{k(0)} = \Delta_{k(0)}$. Hence we must either have $\rho_{k(0)}^{\text{init}} < \rho_{\min}$ or $\Delta_{k(0)} < \rho_{\min}$. Both cases give (4.23): the first gives it directly, and for the second case, we use Lemma 4.7 to get

$$\rho_{\min} > \Delta_{k(0)} \geq \min\left(\Delta_{k(0)}^{\text{init}}, \frac{\omega_C \epsilon}{\kappa_{\text{eg}} + 1/\mu}\right) \geq \min\left(\rho_{k(0)}^{\text{init}}, \frac{\omega_C \epsilon}{\kappa_{\text{eg}} + 1/\mu}\right). \tag{4.24}$$

Since $\rho_{\min} \leq \omega_C \epsilon/(\kappa_{\text{eg}} + 1/\mu)$, we therefore conclude that (4.23) holds.

Since $\rho_{\min} \leq \Delta_0^{\text{init}} = \rho_0^{\text{init}}$, we therefore have $k(0) > 0$ and $\rho_{k(0)-1} \geq \rho_{\min} > \rho_{k(0)}^{\text{init}}$. This reduction in ρ can only happen from a safety step or an unsuccessful step, and we must have $\rho_{k(0)}^{\text{init}} = \alpha_1 \rho_{k(0)-1}$, so $\rho_{k(0)-1} \leq \rho_{\min}/\alpha_1$. If we had a safety step, we know $\|\mathbf{s}_{k(0)-1}\| \leq \gamma_S \rho_{k(0)-1}$, but if we had an unsuccessful step, we must have $\gamma_{\text{dec}}\|\mathbf{s}_{k(0)-1}\| \leq \min(\gamma_{\text{dec}}\Delta_{k(0)-1}, \|\mathbf{s}_{k(0)-1}\|) \leq \rho_{k(0)-1}$. Hence in either case, we have

$$\|\mathbf{s}_{k(0)-1}\| \leq \min(\gamma_S, \gamma_{\text{dec}}^{-1})\rho_{k(0)-1} \leq \frac{1}{\alpha_1}\min(\gamma_S, \gamma_{\text{dec}}^{-1})\rho_{\min} = \frac{\gamma_S}{\alpha_1}\rho_{\min}, \tag{4.25}$$

since $\gamma_S < 1$ and $\gamma_{\text{dec}} < 1$. Hence by Assumption 4.4 and Lemma 4.6 we have

$$c_2 \min\left(\Delta_{k(0)-1}, \frac{\epsilon_g}{\kappa_H}\right) \leq \|\mathbf{s}_{k(0)-1}\| \leq \frac{\gamma_S}{\alpha_1}\rho_{\min}. \tag{4.26}$$

Note that $\rho_{\min} \leq \alpha_1 \epsilon_g/\kappa_H < (\alpha_1 c_2 \epsilon_g)/(\gamma_S \kappa_H)$, where in the last inequality we used the choice of γ_S in Algorithm 3.1. This inequality and the choice of γ_S, together with (4.26), also imply

$$\Delta_{k(0)-1} \leq \frac{\gamma_S \rho_{\min}}{\alpha_1 c_2} < \frac{\rho_{\min}}{\alpha_1} \leq \min\left(\frac{\epsilon_g}{\kappa_H}, \left(\kappa_{\text{eg}} + \frac{2\kappa_{\text{ef}}}{c_1(1-\eta_2)}\right)^{-1}\epsilon\right). \tag{4.27}$$

Then since $\Delta_{k_{(0)}-1} \leq \epsilon_g/\kappa_H$, Lemma 4.6 gives us $\|\mathbf{s}_{k_{(0)}-1}\| \geq c_2 \Delta_{k_{(0)}-1} > \gamma_S \rho_{k_{(0)}-1}$ and the safety phase is not called.

If m_k is not fully linear, then we must have either a successful or model-improving iteration, so $\rho_{k_{(0)}}^{\text{init}} = \rho_{k_{(0)}-1}$, contradicting (4.23). Thus m_k must be fully linear. Now suppose that

$$\Delta_{k_{(0)}-1} > \frac{c_1(1-\eta_2)\|\mathbf{g}_{k_{(0)}-1}\|}{2\kappa_{\text{ef}}}. \tag{4.28}$$

Then using full linearity, we have

$$\epsilon \leq \|\nabla f(\mathbf{x}_{k_{(0)}-1})\| \leq \kappa_{\text{eg}}\Delta_{k(0)-1} + \|\mathbf{g}_{k_{(0)}-1}\| < \left(\kappa_{\text{eg}} + \frac{2\kappa_{\text{ef}}}{c_1(1-\eta_2)}\right)\Delta_{k_{(0)}-1}. \tag{4.29}$$

contradicting (4.27). That is, (4.28) is false and so together with (4.27), we have (4.7). Hence Assumptions 4.3 and 4.4, and Lemma 4.5 imply iteration $(k_0 - 1)$ was very successful (as we have already established the safety phase was not called), so $\rho^{\text{init}}_{k_{(0)}} = \rho_{k_{(0)}-1}$, contradicting (4.23). □

Our first convergence result considers the case where we have finitely-many successful iterations.

Lemma 4.10. *Suppose Assumptions 2.4, 4.3 and 4.4 hold. If there are finitely many successful iterations, then* $\lim_{k\to\infty}\Delta_k = \lim_{k\to\infty}\rho_k = 0$ *and* $\lim_{k\to\infty}\|\nabla f(\mathbf{x}_k)\| = 0$.

Proof. We follow [60, Lemma 10.8], except we must consider safety phases. Let $k_0 = \max(\mathcal{S})$ be the last successful iteration, after which Δ_k is never increased. For any $k > k_0$, we possibly call the criticality phase, and then have either a safety phase, model-improving phase, or an unsuccessful step.

If the model is not fully linear, then either it is made fully linear by the criticality phase, or we have a safety or model-improving step. In the first case, the model is made fully linear at iteration k; in the second and third, it is fully linear at iteration $k+1$. That is, there is at most 1 iteration until the model is fully linear again. Therefore there are infinitely many $k > k_0$ where m_k is fully linear, and so we have either a safety phase or an unsuccessful step. In both of these cases, Δ_k is reduced by a factor of at least $\max(\gamma_{\text{dec}}, \alpha_2, \omega_S) < 1$, so $\Delta_k \to 0$ as $k \to \infty$. Since $\rho_k \leq \Delta_k$ at all iterations, we must also have $\rho_k \to 0$.

For each $k > k_0$, let j_k be the first iteration after k where the model is fully linear. Then from the above discussion we know $0 \leq j_k - k \leq 1$, and hence $\|\mathbf{x}_{j_k} - \mathbf{x}_k\| \leq \Delta_k \to 0$. We now compute

$$\|\nabla f(\mathbf{x}_k)\| \leq \|\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{j_k})\| + \|\nabla f(\mathbf{x}_{j_k}) - \mathbf{g}_{j_k}\| + \|\mathbf{g}_{j_k}\|. \tag{4.30}$$

As $k \to \infty$, the first term of the right-hand side of (4.30) is bounded by $L_{\nabla f}\Delta_k \to 0$, while the second term is bounded by $\kappa_{\text{eg}}\Delta_{j_k} \to 0$; thus it remains to show that the last term goes to zero.

By contradiction, suppose there exists $\epsilon > 0$ and a subsequence k_i such that $\|\mathbf{g}_{j_{k_i}}\| \geq \epsilon > 0$. Then Lemma 4.5 implies that for sufficiently small $\Delta_{j_{k_i}}$ (valid since $\Delta_{j_{k_i}} \to 0$), we get a very successful iteration or a safety step. Since $j_{k_i} \geq k_i > k_0$, this must mean we get a safety step. However, Lemma 4.6 implies that for sufficiently large i, we have $\|\mathbf{s}_{j_{k_i}}\| \geq c_2\Delta_{j_{k_i}} > \gamma_S\rho_{j_{k_i}}$, so the safety step cannot be called, a contradiction. □

Lemma 4.11. *Suppose Assumptions 2.4, 4.3 and 4.4 hold. Then* $\lim_{k\to\infty} \Delta_k = 0$ *and so* $\lim_{k\to\infty} \rho_k = 0$.

Proof. This proof follows [60, Lemma 10.9]. If $|\mathcal{S}| < \infty$, the proof of Lemma 4.10 gives the result. Thus, suppose there are infinitely many successful iterations (i.e. $|\mathcal{S}| = \infty$).

For any $k \in \mathcal{S}$, we have

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 \left(m_k(\mathbf{0}) - m_k(\mathbf{s}_k)\right) \geq \eta_1 c_1 \|\mathbf{g}_k\| \min\left(\frac{\|\mathbf{g}_k\|}{\kappa_H}, \Delta_k\right) > 0. \tag{4.31}$$

But since $\|\mathbf{g}_k\| \geq \min(\epsilon_C, \Delta_k/\mu)$ (see Lemma 4.8), this means that

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 c_1 \min(\epsilon_C, \mu^{-1}\Delta_k) \min\left(\frac{\min(\epsilon_C, \mu^{-1}\Delta_k)}{\kappa_H}, \Delta_k\right) > 0. \tag{4.32}$$

If we were to sum over all $k \in \mathcal{S}$, the left-hand side must be finite as it is bounded above by $f(\mathbf{x}_0)$, remembering that $f \geq 0$ for least-squares objectives. The right-hand side is only finite if $\lim_{k\in\mathcal{S}\to\infty} \Delta_k = 0$. The only time Δ_k is increased is if $k \in \mathcal{S}$, when it is increased by a factor of at most $\overline{\gamma}_{\text{inc}}$. For any given $k \notin \mathcal{S}$, let $j_k \in \mathcal{S}$ be the last successful iteration before k (which exists whenever k is sufficiently large). Then $\Delta_k \leq \overline{\gamma}_{\text{inc}}\Delta_{j_k} \to 0$. Lastly, $\rho_k \to 0$ since $\rho_k \leq \Delta_k$ throughout the algorithm. □

Theorem 4.12. *Suppose Assumptions 2.4, 4.3 and 4.4 hold. Then*

$$\liminf_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0. \tag{4.33}$$

Proof. If $|\mathcal{S}| < \infty$, then this follows from Lemma 4.10. Otherwise, it follows from Lemma 4.11 and Lemma 4.9. □

Theorem 4.13. *Suppose Assumptions 2.4, 4.3 and 4.4 hold. Then* $\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0$.

Proof. If $|\mathcal{S}| < \infty$, then the result follows from Lemma 4.10. Thus, suppose there are infinitely many successful iterations (i.e. $|\mathcal{S}| = \infty$). For this case, we follow the proof of [60, Theorem 10.13], except we must consider safety steps.

To find a contradiction, suppose there is a subsequence of successful iterations t_j with $\|\nabla f(\mathbf{x}_{t_j})\| \geq \epsilon_0$ for some $\epsilon_0 > 0$ (note: we do not consider any other iteration types as $\mathbf{x}_k$ does not change for these). Hence by Lemma 4.8, we must have $\|\mathbf{g}_{t_j}\| \geq \epsilon > 0$ for some ϵ, where without loss of generality we assume that

$$\epsilon < \min\left(\epsilon_C, \frac{\epsilon_0}{2 + \kappa_{\text{eg}}\mu}\right). \tag{4.34}$$

Let ℓ_j be the first iteration $\ell_j > t_j$ such that $\|\mathbf{g}_{\ell_j}\| < \epsilon$, which is guaranteed to exist by Theorem 4.12. That is, there exist subsequences $t_j < \ell_j$ satisfying

$$\|\mathbf{g}_k\| \geq \epsilon \quad \text{for } k = t_j, \ldots, \ell_j - 1, \text{ and} \quad \|\mathbf{g}_{\ell_j}\| < \epsilon. \tag{4.35}$$

Now consider the iterations $\mathcal{K} := \cup_{j\geq 0}\{t_j, \ldots, \ell_j - 1\}$.

Since $\|\mathbf{g}_k\| \geq \epsilon$ and $\Delta_k \to 0$ (Lemma 4.11), Lemma 4.5 implies that for sufficiently large $k \in \mathcal{K}$, there can be no unsuccessful steps. That is, iteration k is a safety step, or if not it must be successful or model-improving. By the same reasoning as in the proof of Lemma 4.10, since $\|\mathbf{g}_k\| \geq \epsilon$, for $k \in \mathcal{K}$ sufficiently large, Lemma 4.6 implies that $\|\mathbf{s}_k\| \geq c_2\Delta_k > \gamma_S\rho_k$, so the safety step is never called.

For each successful iteration $k \in \mathcal{K} \cap \mathcal{S}$, we have from (4.6)

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1(m_k(\mathbf{0}) - m_k(\mathbf{s}_k)) \geq \eta_1 c_1 \epsilon \min\left(\frac{\|\mathbf{g}_k\|}{\kappa_H}, \Delta_k\right) > 0, \tag{4.36}$$

and for k sufficiently large (so that $\Delta_k \leq \epsilon/\kappa_H$), we get

$$\Delta_k \leq \frac{f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})}{\eta_1 c_1 \epsilon}. \tag{4.37}$$

Since for $k \in \mathcal{K}$ sufficiently large, we either have successful or model-improving steps, and of these $\mathbf{x}_k$ is only changed on successful steps, we have (for j sufficiently large)

$$\|\mathbf{x}_{\ell_j} - \mathbf{x}_{t_j}\| \leq \sum_{k=t_j, k\in\mathcal{K}\cap\mathcal{S}}^{\ell_j-1} \|\mathbf{x}_k - \mathbf{x}_{k+1}\| \leq \sum_{k=t_j, k\in\mathcal{K}\cap\mathcal{S}}^{\ell_j-1} \Delta_k \leq \frac{f(\mathbf{x}_{t_j}) - f(\mathbf{x}_{\ell_j})}{\eta_1 c_1 \epsilon}. \tag{4.38}$$

Since $\{f(\mathbf{x}_k) : k \in \mathcal{K}\}$ is a monotone decreasing sequence by (4.36), and bounded below (as $f \geq 0$ for least-squares problems), it must converge. Thus $f(\mathbf{x}_{t_j}) - f(\mathbf{x}_{\ell_j}) \to 0$, and hence $\|\mathbf{x}_{\ell_j} - \mathbf{x}_{t_j}\| \to 0$ as $j \to \infty$.

Now, we compute

$$\|\nabla f(\mathbf{x}_{t_j})\| \leq \|\nabla f(\mathbf{x}_{t_j}) - \nabla f(\mathbf{x}_{\ell_j})\| + \|\nabla f(\mathbf{x}_{\ell_j}) - \mathbf{g}_{\ell_j}\| + \|\mathbf{g}_{\ell_j}\|. \tag{4.39}$$

Similarly to Lemma 4.10, the first term goes to zero as $j \to \infty$ since ∇f is continuous and $\|\mathbf{x}_{\ell_j} - \mathbf{x}_{t_j}\| \to 0$. Since $\|\mathbf{g}_{\ell_j}\| < \epsilon < \epsilon_C$, the criticality step is called for iteration ℓ_j, so m_{ℓ_j} is fully linear on $B(\mathbf{x}_{\ell_j}, \Delta_{\ell_j})$ for $\Delta_{\ell_j} \leq \mu\|\mathbf{g}_{\ell_j}\|$. Hence the second term is bounded by $\kappa_{\text{eg}}\Delta_{\ell_j} \leq \kappa_{\text{eg}}\mu\epsilon$. Lastly, the third term is bounded by ϵ by definition of ℓ_j.

All together, this means that for sufficiently large j,

$$\|\nabla f(\mathbf{x}_{t_j})\| \leq \epsilon + \kappa_{\text{eg}}\mu\epsilon + \epsilon = (2 + \kappa_{\text{eg}}\mu)\epsilon < \epsilon_0, \tag{4.40}$$

and we have our contradiction. □

4.3 Worst-Case Complexity

Next, we bound the number of iterations and objective evaluations until $\|\nabla f(\mathbf{x}_k)\| < \epsilon$. We know such a bound exists from Theorem 4.12. Let i_ϵ be the last iteration before $\|\nabla f(\mathbf{x}_{i_\epsilon+1})\| < \epsilon$ for the first time.

Lemma 4.14. *Suppose Assumptions 2.4, 4.3 and 4.4 hold. Let $|\mathcal{S}_{i_\epsilon}|$ be the number of successful steps up to iteration i_ϵ. Then*

$$|\mathcal{S}_{i_\epsilon}| \leq \frac{f(\mathbf{x}_0)}{\eta_1 c_1} \max\left(\kappa_H \epsilon_g^{-2}, \epsilon_g^{-1} \rho_{\min}^{-1}\right), \tag{4.41}$$

where ϵ_g is defined in (4.19)*, and $\rho_{\min}$ in* (4.21)*.*

Proof. For all $k \in \mathcal{S}_{i_\epsilon}$, we have the sufficient decrease condition

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 \left(m_k(\mathbf{0}) - m_k(\mathbf{s}_k)\right) \geq \eta_1 c_1 \|\mathbf{g}_k\| \min\left(\frac{\|\mathbf{g}_k\|}{\kappa_H}, \Delta_k\right). \tag{4.42}$$

Since $\|\mathbf{g}_k\| \geq \epsilon_g$ from Lemma 4.8 and $\Delta_k \geq \rho_k \geq \rho_{\min}$ from Lemma 4.9, this means

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 c_1 \epsilon_g \min\left(\frac{\epsilon_g}{\kappa_H}, \rho_{\min}\right). \tag{4.43}$$

Summing (4.43) over all $k \in \mathcal{S}_{i_\epsilon}$, and noting that $0 \leq f(\mathbf{x}_k) \leq f(\mathbf{x}_0)$, we get

$$f(\mathbf{x}_0) \geq |\mathcal{S}_{i_\epsilon}| \eta_1 c_1 \epsilon_g \min\left(\frac{\epsilon_g}{\kappa_H}, \rho_{\min}\right), \tag{4.44}$$

from which (4.41) follows. □

We now need to count the number of iterations of Algorithm 3.1 that are not successful. Following [83], we count each iteration of the loop inside the criticality phase (Algorithm 2.2) as a separate iteration[2]—in effect, one 'iteration' corresponds to one construction of the model m_k (3.2). We also consider separately the number of criticality phases for which Δ_k is not reduced (i.e. $\Delta_k = \Delta_k^{\text{init}}$). Counting until iteration i_ϵ (inclusive), we let

- $\mathcal{C}_{i_\epsilon}^M$ be the set of criticality phase iterations $k \leq i_\epsilon$ for which Δ_k is not reduced (i.e. the first iteration of every call of Algorithm 2.2);
- $\mathcal{C}_{i_\epsilon}^U$ be the set of criticality phase iterations $k \leq i_\epsilon$ where Δ_k is reduced (i.e. all iterations except the first for every call of Algorithm 2.2);
- $\mathcal{F}_{i_\epsilon}$ be the set of iterations where the safety phase is called;
- $\mathcal{M}_{i_\epsilon}$ be the set of iterations where the model-improving phase is called; and
- $\mathcal{U}_{i_\epsilon}$ be the set of unsuccessful iterations.

[2] Note that the analysis in [90] bounds the number of outer iterations of Algorithm 3.1; i.e. excluding $\mathcal{C}_{i_\epsilon}^M$ and $\mathcal{C}_{i_\epsilon}^U$. Instead, they prove that while $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon$, the criticality phase requires at most $|\log \epsilon|$ iterations. Thus their bound on the number of objective evaluations is a factor $|\log \epsilon|$ larger than in [83] and than here.

Lemma 4.15. *Suppose Assumptions 2.4, 4.3 and 4.4 hold. Then we have the bounds*

$$|\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}| \leq |\mathcal{S}_{i_\epsilon}| \cdot \frac{\log \overline{\gamma}_{\text{inc}}}{|\log \alpha_3|} + \frac{1}{|\log \alpha_3|} \log\left(\frac{\Delta_0^{\text{init}}}{\rho_{\min}}\right), \tag{4.45}$$

$$|\mathcal{C}_{i_\epsilon}^M| \leq |\mathcal{F}_{i_\epsilon}| + |\mathcal{S}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}|, \tag{4.46}$$

$$|\mathcal{M}_{i_\epsilon}| \leq |\mathcal{C}_{i_\epsilon}^M| + |\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{S}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}|, \tag{4.47}$$

where $\alpha_3 := \max(\omega_C, \omega_S, \gamma_{\text{dec}}, \alpha_2) < 1$ *and* $\rho_{\min}$ *is defined in* (4.21).

Proof. On each iteration $k \in \mathcal{C}_{i_\epsilon}^U$, we reduce Δ_k by a factor of ω_C. Similarly, on each iteration $k \in \mathcal{F}_{i_\epsilon}$ we reduce Δ_k by a factor of at least $\max(\omega_S, \alpha_2)$, and for iterations in $\mathcal{U}_{i_\epsilon}$ by a factor of at least $\max(\gamma_{\text{dec}}, \alpha_2)$. On each successful iteration, we increase Δ_k by a factor of at most $\overline{\gamma}_{\text{inc}}$, and on all other iterations, Δ_k is either constant or reduced. Therefore, we must have

$$\rho_{\min} \leq \Delta_{i_\epsilon} \leq \Delta_0^{\text{init}} \cdot \omega_C^{|\mathcal{C}_{i_\epsilon}^U|} \cdot \max(\omega_S, \alpha_2)^{|\mathcal{F}_{i_\epsilon}|} \cdot \max(\gamma_{\text{dec}}, \alpha_2)^{|\mathcal{U}_{i_\epsilon}|} \cdot \overline{\gamma}_{\text{inc}}^{|\mathcal{S}_{i_\epsilon}|}, \tag{4.48}$$

$$\leq \Delta_0^{\text{init}} \cdot \alpha_3^{|\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}|} \cdot \overline{\gamma}_{\text{inc}}^{|\mathcal{S}_{i_\epsilon}|}, \tag{4.49}$$

from which (4.45) follows.

After every call of the criticality phase, we have either a safety, successful or unsuccessful step, giving us (4.46). Similarly, after every model-improving phase, the next iteration cannot call a subsequent model-improving phase, giving us (4.47). □

Assumption 4.16. *The algorithm parameter* $\epsilon_C \geq c_3 \epsilon$ *for some constant* $c_3 > 0$.

Note that Assumption 4.16 can be easily satisfied by appropriate parameter choices in Algorithm 3.1.

Theorem 4.17. *Suppose Assumptions 2.4, 4.3, 4.4 and 4.16 hold. Then the number of iterations* i_ϵ *(i.e. the number of times a model* m_k (3.2) *is built) until* $\|\nabla f(\mathbf{x}_{i_\epsilon+1})\| < \epsilon$ *is at most*

$$\left\lfloor \frac{4f(\mathbf{x}_0)}{\eta_1 c_1} \left(1 + \frac{\log \overline{\gamma}_{\text{inc}}}{|\log \alpha_3|}\right) \max\left(\kappa_H c_4^{-2} \epsilon^{-2}, c_4^{-1} c_5^{-1} \epsilon^{-2}, c_4^{-1} (\Delta_0^{\text{init}})^{-1} \epsilon^{-1}\right) \right.$$
$$\left. + \frac{4}{|\log \alpha_3|} \max\left(0, \log\left(\Delta_0^{\text{init}} c_5^{-1} \epsilon^{-1}\right)\right) \right\rfloor \tag{4.50}$$

where $c_4 := \min\left(c_3, (1 + \kappa_{\text{eg}}\mu)^{-1}\right)$ *and*

$$c_5 := \min\left(\frac{\omega_C}{\kappa_{\text{eg}} + 1/\mu}, \frac{\alpha_1 c_4}{\kappa_H}, \alpha_1 \left(\kappa_{\text{eg}} + \frac{2\kappa_{\text{ef}}}{c_1(1 - \eta_2)}\right)^{-1}\right). \tag{4.51}$$

Proof. From Assumption 4.16 and Lemma 4.8, we have $\epsilon_g = c_4\epsilon$. Similarly, from Lemma 4.9 we have $\rho_{\min} = \min(\Delta_0^{\text{init}}, c_5\epsilon)$. Thus using Lemma 4.15, we can bound the total number of iterations by

$$|\mathcal{C}_{i_\epsilon}^M| + |\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{S}_{i_\epsilon}| + |\mathcal{M}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}| \tag{4.52}$$

$$\leq 4|\mathcal{S}_{i_\epsilon}| + 4\left(|\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}|\right), \tag{4.53}$$

$$\leq 4|\mathcal{S}_{i_\epsilon}|\left(1 + \frac{\log \overline{\gamma}_{\text{inc}}}{|\log \alpha_3|}\right) + \frac{4}{|\log \alpha_3|}\log\left(\frac{\Delta_0^{\text{init}}}{\rho_{\min}}\right), \tag{4.54}$$

and so (4.50) follows from this and Lemma 4.14. □

We can summarise our results as follows:

Corollary 4.18. *Suppose Assumptions 2.4, 4.3, 4.4 and 4.16 hold. Then for $\epsilon \in (0, 1]$, the number of iterations i_ϵ (i.e. the number of times a model m_k (3.2) is built) until $\|\nabla f(\mathbf{x}_{i_\epsilon+1})\| < \epsilon$ is at most $\mathcal{O}(\kappa_H\kappa_d^2\epsilon^{-2})$, and the number of objective evaluations until i_ϵ is at most $\mathcal{O}(\kappa_H\kappa_d^2 p\,\epsilon^{-2})$, where $\kappa_d := \max(\kappa_{\text{ef}}, \kappa_{\text{eg}})$.*

Proof. From Theorem 4.17, we have $c_4^{-1} = \mathcal{O}(\kappa_{\text{eg}})$ and so

$$c_5^{-1} = \mathcal{O}(\max(\kappa_{\text{eg}}, \kappa_H c_4^{-1}, \kappa_{\text{ef}} + \kappa_{\text{eg}})) = \mathcal{O}(\kappa_H\kappa_d). \tag{4.55}$$

To leading order, the number of iterations is

$$\mathcal{O}(\max(\kappa_H c_4^{-2}, c_4^{-1}c_5^{-1})\epsilon^{-2}) = \mathcal{O}(\kappa_H\kappa_d^2\epsilon^{-2}), \tag{4.56}$$

as required. In every type of iteration, we change at most $n + 1$ points, and so require no more than $p + 1$ evaluations. □

Remark 4.19. Theorem 4.17 gives us a possible termination criterion for Algorithm 3.1—we loop until k exceeds the value (4.50) or until $\rho_k \leq \rho_{\min}$. However, this would require us to know problem constants κ_{ef}, κ_{eg} and κ_H in advance, which is not usually the case. Moreover, (4.50) is a worst-case bound and so unduly pessimistic.

Remark 4.20. In [83], the authors propose a different criterion to test whether the criticality phase should be entered: $\|\mathbf{g}_k^{\text{init}}\| \leq \Delta_k/\mu$ rather than $\|\mathbf{g}_k^{\text{init}}\| \leq \epsilon_C$ as found here and in [60]. We are able to use our criterion because of Assumption 4.16. If this did not hold, we would have $\epsilon_g \ll \epsilon$ and so $\rho_{\min} \ll \epsilon$, which would worsen the result in Theorem 4.17. In practice, Assumption 4.16 is reasonable, as we would not expect a user to prescribe a criticality tolerance much smaller than their desired solution tolerance.

We can now compare the complexity of DFO-LS with the standard complexity bounds for general model-based DFO. For first-order criticality, Garmanjani, Júdice and Vicente [83] prove a complexity bound of $\mathcal{O}(\kappa_H \kappa_d^2 \epsilon^{-2})$ iterations and $\mathcal{O}(\kappa_H \kappa_d^2 n \epsilon^{-2})$ evaluations, where $\kappa_d := \max(\kappa_{\text{ef}}, \kappa_{\text{eg}})$ as per Corollary 4.18. We also estimate that, for DFO-LS, we have from Lemma 4.2 that $\kappa_d = \mathcal{O}((\kappa_{\text{eg}}^{\text{r}})^2) = \mathcal{O}(L_J^2 p^2 \Lambda^2)$ and $\kappa_H = \mathcal{O}(\kappa_d)$, at least when the model is fully linear. However, in the general first-order case, linear interpolation gives us the problem constants $\kappa_d = \mathcal{O}(L_{\nabla f} n \Lambda)$ from Theorem 2.12 and $\kappa_H = \mathcal{O}(1)$. That is, DFO-LS has the same count of iterations and evaluations as general first-order methods, but worse problem constants due to the least-squares structure and possibly from the use of regression models.

However, our model (3.2) is better than a simple linear model for f, as it captures some of the curvature information in the objective via the term $J_k^\top J_k$. This means that DFO-LS produces models that are between fully linear and fully quadratic (Definition 2.24), which is the requirement for convergence of second-order methods. It therefore makes sense to also compare the complexity of DFO-LS with the complexity of second-order methods.

Unsurprisingly, the standard bound for second-order methods is worse in general, than for first-order methods: Júdice [114] proves bounds of $\mathcal{O}(\max(\kappa_H \kappa_d^2, \kappa_d^3)\epsilon^{-3})$ iterations and $\mathcal{O}(\max(\kappa_H \kappa_d^2, \kappa_d^3) n^2 \epsilon^{-3})$ evaluations, where we require fully quadratic models, and where now $\kappa_d := \max(\kappa_{\text{ef}}, \kappa_{\text{eg}}, \kappa_{\text{eh}})$. To achieve this, we require fully quadratic interpolation, and so we have $\kappa_d = \mathcal{O}(L_H n^3 \Lambda)$ from Theorem 2.25, assuming sufficient smoothness for f, and if $\|\nabla^2 f(\mathbf{x})\|$ is uniformly bounded, then we would expect $\kappa_H = \mathcal{O}(\kappa_{\text{eh}}) = \mathcal{O}(\kappa_d)$.

All together, we may conclude that DFO-LS has the iteration and evaluation complexity of a first-order method, but the problem constants (i.e. dependency on p or n) closer to a second-order method. Specifically, DFO-LS achieves first-order convergence in $\mathcal{O}(p^6 \epsilon^{-2})$ iterations or $\mathcal{O}(p^7 \epsilon^{-2})$ evaluations. By comparison, general first-order methods require $\mathcal{O}(n^2 \epsilon^{-2})$ iterations or $\mathcal{O}(n^3 \epsilon^{-2})$ evaluations and second-order methods require $\mathcal{O}(n^9 \epsilon^{-3})$ iterations or $\mathcal{O}(n^{11} \epsilon^{-3})$ evaluations.

Remark 4.21. In [83, 114], the estimation of κ_d and κ_H in terms of n was based on uniform bounds on the singular values of the interpolation linear system, and so they show all methods with a weaker dependency on n. For instance, for first-order models they estimate $\kappa_d = \mathcal{O}(L_{\nabla f} \sqrt{n})$ using (2.22). A similar approach in the proof of Lemma 4.2 would conclude that $\kappa_d = \mathcal{O}(L_J^2 p)$ for DFO-LS. Either way, we may conclude that the complexity of DFO-LS lies between first- and second-order methods.

Remark 4.22 (*Discussion of Assumption 4.3*). It is also important to note that when m_k is fully linear, we have an explicit bound $\|H_k\| \leq \widetilde{\kappa}_H = \mathcal{O}(\kappa_d)$ from Lemma 4.2. This means that Assumption 4.3, which typically necessary for first-order convergence (e.g. [60, 83]), is

not required for Theorem 4.12 and our complexity analysis. To remove the assumption, we need to change Algorithm 3.1 in two places:

1. Replace the test for entering the criticality phase with

$$\min\left(\|\mathbf{g}_k^{\text{init}}\|, \frac{\|\mathbf{g}_k^{\text{init}}\|}{\max(\|H_k^{\text{init}}\|, 1)}\right) \leq \epsilon_C; \qquad \text{and} \tag{4.57}$$

2. Require the criticality phase to output m_k fully linear and Δ_k satisfying

$$\Delta_k \leq \mu \min\left(\|\mathbf{g}_k\|, \frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)}\right). \tag{4.58}$$

With these changes, the criticality phase still terminates, but instead of (4.16) we have

$$\min\left(\Delta_k^{\text{init}}, \frac{\omega_C \epsilon}{\kappa_{\text{eg}} + 1/\mu}, \frac{\omega_C \epsilon}{\kappa_{\text{eg}} + \widetilde{\kappa}_H/\mu}\right) \leq \Delta_k \leq \Delta_k^{\text{init}}. \tag{4.59}$$

We can also augment Lemma 4.8 with the following, which can be used to arrive at a new value for $\rho_{\min}$.

Lemma 4.23. *In all iterations,* $\|\mathbf{g}_k\|/\max(\|H_k\|, 1) \geq \min(\epsilon_C, \Delta_k/\mu)$. *If Assumption 2.4 holds and* $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$ *then*

$$\frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)} \geq \epsilon_H := \min\left(\epsilon_C, \frac{\epsilon}{(1 + \kappa_{\text{eg}}\mu)\widetilde{\kappa}_H}\right) > 0. \tag{4.60}$$

Ultimately, we arrive at complexity bounds that match Corollary 4.18, but replacing κ_H with $\widetilde{\kappa}_H$. However, Assumption 4.3 is still necessary for Theorem 4.13 to hold. We note that we do not enforce Assumption 4.3 in the implementation of DFO-LS, but have not observed large values of $\|H_k\|$ in practice.

Chapter 5

Numerical Results for DFO-LS

In this chapter, we show the results of our numerical testing of DFO-LS.[1] The most structurally significant novel feature in DFO-LS compared to existing methods is its use of linear residual models rather than quadratic, which is the standard approach in [240, 225]. Hence, we begin with a discussion on the impact of this change (Section 5.1). We then provide several numerical studies examining the other novel features of DFO-LS (Section 5.2), and conclude by benchmarking DFO-LS against other solvers (Section 5.3).

5.1 Impact of Linear Residual Models

As has been discussed at length, there are many similarities between DFO-LS and DFBOLS from Zhang et al. [240]. The theoretical algorithm described in [240] (called DFLS) allows linear residual models in principle, and its convergence theory covers this case. However, the implementation of this algorithm, DFBOLS, does not, strictly speaking, allow linear residual models; instead, it either uses underdetermined or fully quadratic models for each r_i, with between $n+2$ and $(n+1)(n+2)/2$ interpolation points. Furthermore, there are no numerical results showing how linear residual models, or models with $n+2$ points, perform compared to (underdetermined or fully) quadratic residual models. Aside from this, there are several respects in which DFO-LS is simpler than DFLS/DFBOLS:

- The use of linear models for each residual (2.6) means we require only $n+1$ interpolation points, as opposed to between $n+2$ and $(n+1)(n+2)/2$ points needed by DFBOLS. This results in both a smaller interpolation system and a lower startup cost (where an initial Y_0 of the correct size is constructed, and $\mathbf{r}$ evaluated at each of these points);
- As a result of using linear residual models, there is no ambiguity in how to construct the full model m_k (3.2). In DFLS and DFBOLS, simply taking a sum of squares of

[1] All results use version 1.0.1.

each residual's model gives a quartic. The authors drop the cubic and quartic terms, and choose the quadratic term from one of three possibilities (1.30). This requires the introduction of three new algorithm parameters, each of which may require calibration; and

- DFO-LS's method for choosing a point to replace when doing model updating, as discussed in Section 3.3.2, yields a unification of the geometric ('optimal Λ-poisedness') and algebraic ('stable update') perspectives on this update. In DFBOLS, the connection exists but is less direct, as it uses the same method as BOBYQA (3.15) with $\sigma_t \geq \ell_t(\mathbf{y}^+)^2$. As discussed in [186], this bound may sometimes be violated as a result of rounding errors, and thus requires an extra geometry-improving routine to 'rescue' the algorithm from this problem. DFO-LS does not need or have this routine.

By comparison, the first point above does not apply to POUNDERS [225], which allows linear models initially, and at each iteration constructs models using $n+1 \leq p \leq p_{\max}$ points, where $p_{\max} \in [n+2, (n+1)(n+2)/2]$ is a user input and p is chosen dynamically each time by selecting points from the full history of iterates using the method from [227].[2] The method of model construction is to use underdetermined quadratic residual models (with minimal change to the model Hessian), and so while linear models are used initially, as soon as $p > n+1$ for some iteration, quadratic residual models are used in all subsequent iterations. The second point above, however, does not apply, as POUNDERS uses a model for the full objective that is equivalent to a full quadratic approximation (i.e. including all available second-order information). Similarly to existing literature for DFBOLS [240], no numerical results for using only linear residual models in POUNDERS are available; in fact, we are not aware of any existing work showing extensive numerical results or comparisons for POUNDERS.

5.1.1 Numerical Study of Linear Residual Models

In this section, we present numerical results demonstrating the impact of using linear models for each residual compared to quadratic models. Hence, here we compare DFO-LS against

- DFBOLS, the Fortran implementation from [240], provided by H. Zhang;[3]
- Py-DFBOLS, our own implementation of DFBOLS, designed to be as similar to DFO-LS in structure as possible. In particular, it is implemented in Python and uses the factorisation approach to solving the interpolation system (see Section 3.3.4). As a

[2] POUNDERS can also choose from a set of points where the user knows the objective value a priori, an optional input.

[3] Private correspondence, May 2015.

result, comparing the runtime of DFO-LS with Py-DFBOLS represents a like-for-like test of algorithm speed;

- BOBYQA [186], as originally implemented in Fortran by Powell, available from [242]; and,
- POUNDERS [225], which uses adaptive interpolation models for each residual, and is incorporated into PETSc. Testing was performed using the Python package petsc4py 3.10.1 and the default setting of using at most $p_{\max} = 2n + 1$ interpolation points at each iteration.

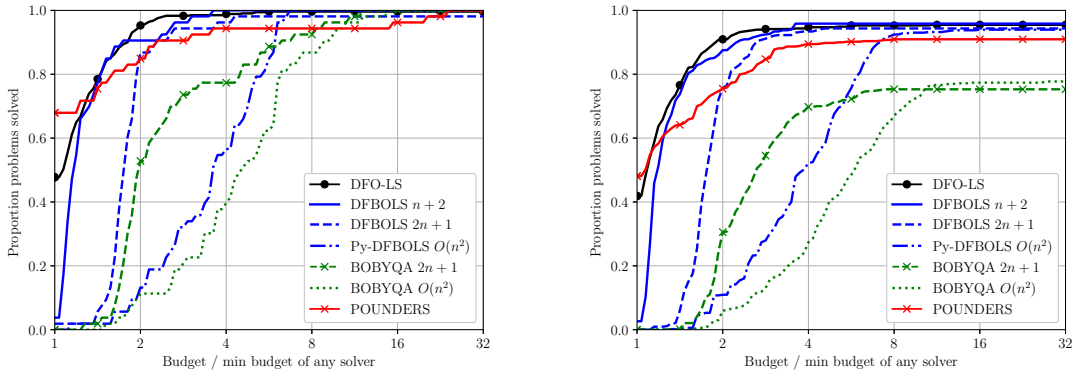
To more fairly compare linear and quadratic models, we do not use any other novel features that are default in DFO-LS, such as alternative trust-region parameter values as described in Section 3.4.1, or multiple restarts (Section 3.4.2). In addition, we only use DFO-LS with interpolation models (i.e. $p = n$ in (3.6)), to align with the approach of DFBOLS and POUNDERS—we test both reduced initialisation cost (using $p < n$) and regression models (with $p > n$) in Section 5.2.

For all solvers, we use an initial trust-region radius of $\Delta_0^{\text{init}} = 0.1 \max(\|\mathbf{x}_0\|_\infty, 1)$ (the default for DFO-LS; see Remark 3.2) and final trust-region radius $\rho_{\text{end}} = 10^{-10}$ where possible,[4] to avoid this being the termination condition as often as possible. All other parameters are set to their default values.

We tested BOBYQA and (Py-)DFBOLS with $n + 2$, $2n + 1$ and $(n + 1)(n + 2)/2$ interpolation points. All these solvers use quadratic interpolation models, and do not allow the use of $n + 1$ interpolation points. Here, we show the $n + 2$ and $2n + 1$ cases for DFBOLS and the $(n + 1)(n + 2)/2$ case for Py-DFBOLS. These were chosen because Py-DFBOLS performs very similarly to DFBOLS in the case of $n + 2$ and $2n + 1$ points, and outperforms DFBOLS in the $(n + 1)(n + 2)/2$ case. Similarly, we show the best-performing $2n + 1$ and $(n + 1)(n + 2)/2$ cases for BOBYQA.

The testing is on the (MW) problem collection, with and without stochastic noise, with a budget of $200(n + 1)$ evaluations. For noisy problems we run 10 instances for every problem and every solver; we also do this for DFO-LS for noiseless problems, since it initialises using random directions. As described in Section 2.4.2, for data and performance profile construction we always treat each of these 10 problem instances as a separate problem to be solved.

[4] POUNDERS does not have this as a user input. Instead we set all gradient tolerances to zero.



(a) Smooth objective *(b) Mult. Gaussian noise $\sigma = 10^{-2}$*

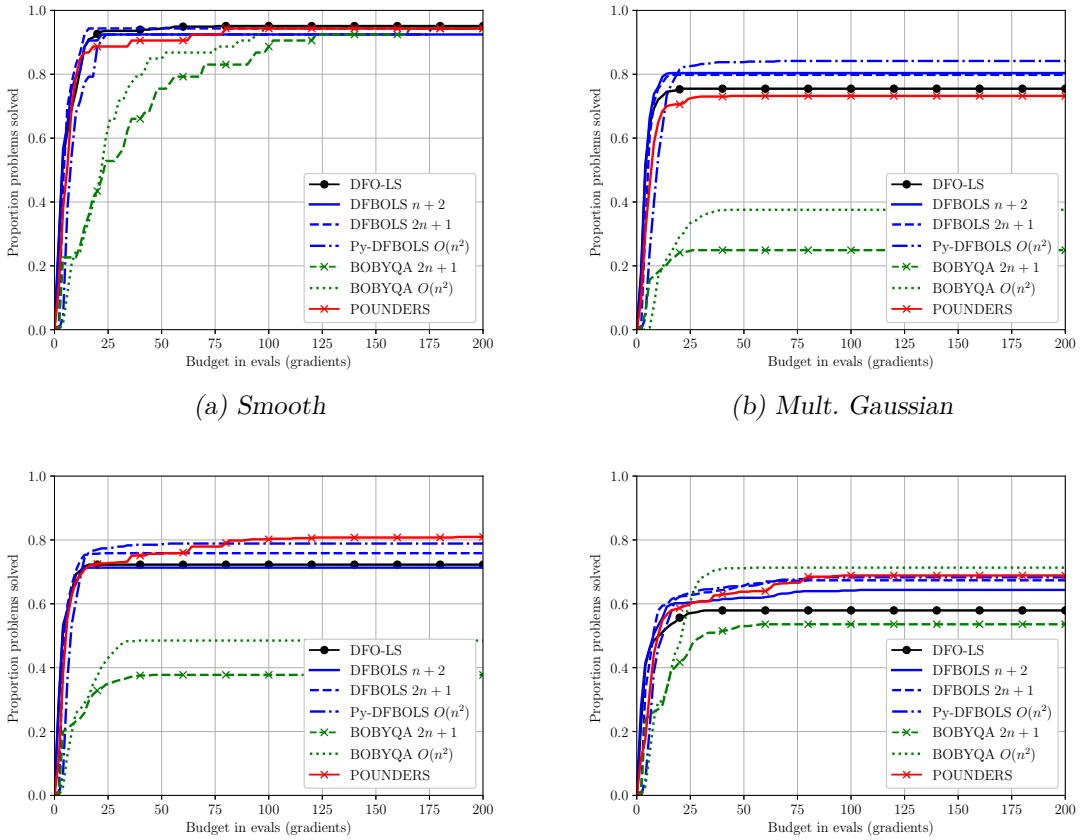
Figure 5.1. Performance profile comparison of DFO-LS with BOBYQA, DFBOLS and POUNDERS for low accuracy $\tau = 10^{-1}$. For the BOBYQA and DFBOLS runs, n+2, 2n+1 and $\mathcal{O}(n^2) = (n+1)(n+2)/2$ are the number of interpolation points. For DFO-LS and all noisy problems, results are an average of 10 runs for each solver. The problem collection is (MW).

Low accuracy setting Firstly, Figure 5.1 shows two performance profiles under the low accuracy requirement $\tau = 10^{-1}$. Here we see an important benefit of DFO-LS and POUNDERS compared to BOBYQA and DFBOLS—allowing a smaller interpolation set means that they can begin the main iteration and make progress sooner. This is reflected in Figure 5.1, where DFO-LS and POUNDERS are the fastest solvers more frequently than the others, both with smooth and noisy objective evaluations. However, the performance of POUNDERS is less strong than DFO-LS for larger performance ratios (i.e. $\alpha \geq 2$ in (2.57)). In line with the results from [240], BOBYQA does not perform as well as POUNDERS, DFBOLS or DFO-LS, as it does not exploit the least-squares problem structure.

The low accuracy requirement often corresponds in practice to the typical case when the objective/residual evaluations are very expensive, more so than the linear algebra and storage costs. The limiting factor then is the (small) evaluation budget, and hence we generally expect objective improvement rather than accurate optimisation from the solver.

High accuracy setting Next, Figure 5.2a shows results for accuracy $\tau = 10^{-5}$ and smooth objective evaluations. Note that our simplification from quadratic to linear residual models has not led to a loss of performance for obtaining high-accuracy solutions, and produces essentially identical long-budget performance. At this level, the advantage from the smaller startup cost is no longer seen, but particularly in the performance profile, we can still see the substantially higher startup cost of using $(n+1)(n+2)/2$ interpolation points.

Similarly, the remainder of the plots in Figure 5.2 shows results for noisy problems. Here, DFO-LS suffers a small performance penalty (of approximately 5–10%) compared



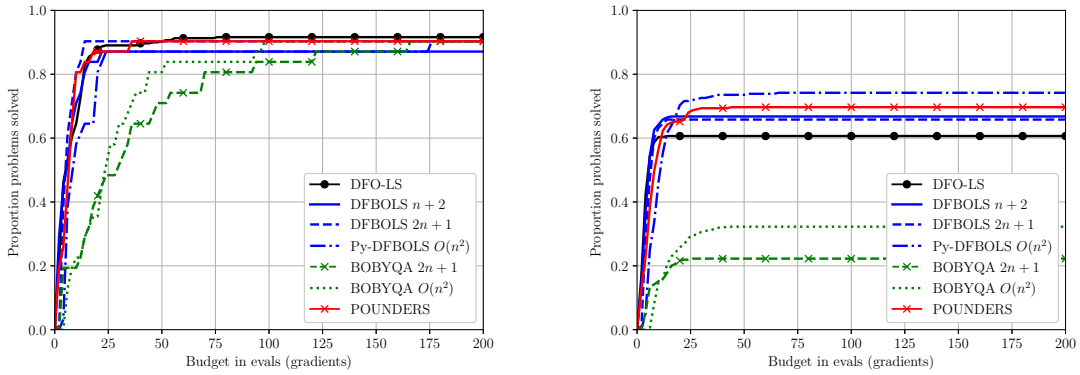
(a) Smooth *(b) Mult. Gaussian*

(c) Add. Gaussian *(d) Add. χ^2*

Figure 5.2. Comparison of data profiles for DFO-LS with BOBYQA, DFBOLS and POUNDERS for smooth and noisy objectives (with $\sigma = 10^{-2}$, to accuracy $\tau = 10^{-5}$). For the BOBYQA and DFBOLS runs, $n+2$, $2n+1$ and $\mathcal{O}(n^2) = (n+1)(n+2)/2$ are the number of interpolation points. For DFO-LS and all noisy problems, results are an average of 10 runs for each solver. The problem collection is (MW).

to DFBOLS, particularly when using $2n+1$ and $(n+1)(n+2)/2$ interpolation points, suggesting that the extra curvature and evaluation information in DFBOLS has some benefit for noisy problems. Also, the performance penalty is larger in the case of additive noise than multiplicative, and here there is also a similar performance penalty compared to POUNDERS. Note that additive noise makes all our test problems nonzero residual (i.e. $f(\mathbf{x}^*) > 0$ for the true minimum $\mathbf{x}^*$); however in the next section we show that this is not a key driver of this differential.

Note also that, although BOBYQA suffers a substantial performance penalty when moving from smooth to noisy problems, this penalty (compared to DFO-LS and DFBOLS) is much less for additive χ^2 noise. This is likely because this noise model makes each residual function more complicated by taking square roots, but the change to the full objective is relatively benign—simply adding χ^2 random variables.



(a) Smooth objective *(b) Mult. Gaussian noise $\sigma = 10^{-2}$*

Figure 5.3. Data profile comparison of DFO-LS with BOBYQA, DFBOLS and POUNDERS for nonzero residual problems only, to accuracy $\tau = 10^{-5}$. For the BOBYQA and DFBOLS runs, $n+2$, $2n+1$ and $\mathcal{O}(n^2) = (n+1)(n+2)/2$ are the number of interpolation points. For DFO-LS and all noisy problems, results are an average of 10 runs for each solver. The problem collection is (MW).

Nonzero residual problems We saw above that DFO-LS suffered a higher—but still small— loss of performance, compared to DFBOLS and POUNDERS, for problems with additive noise. To ascertain if this is because Gauss-Newton methods are known to have slower asymptotic convergence rates for nonzero residual problems (see Section 1.1.3), we extract the performance of the nonzero residual problems only from the test set results we already presented; Figure 5.3 shows the resulting data profiles for accuracy $\tau = 10^{-5}$, for smooth objectives and multiplicative Gaussian noise ($\sigma = 10^{-2}$).[5] For multiplicative Gaussian noise, we see for all solvers a worse performance on nonzero residual problems (compared to all problems), however, in all cases, DFO-LS performs similarly well against DFBOLS and POUNDERS compared to looking at all problems.

Conclusions to evaluation comparisons for linear residual models The numerical results in this section show that DFO-LS performs comparably to DFBOLS and POUNDERS in terms of evaluation counts, and outperforms BOBYQA, in both smooth and noisy settings, and for low and high accuracy. DFO-LS exhibits a slight performance loss compared to DFBOLS and POUNDERS for additive noisy problems. We may explain the similar performance of DFO-LS to DFBOLS and POUNDERS, despite their use of higher order models, as being due to the general effectiveness of Gauss-Newton-like frameworks for nonlinear least-squares, especially for zero-residual problems; and furthermore, by the usual remit of DFO algorithms in which the asymptotic regimes are not or cannot really be observed or targeted by the accuracy at which the problems are solved.

[5] For both additive noise models, all problems are, in expectation, nonzero residual, even if the underlying smooth problem is zero residual.

Solver	Measure	Smooth	Mult. Gaussian	Add. Gaussian	Add. χ^2
DFO-LS	Runtime	47s [1x]	10s [1x]	10s [1x]	13s [1x]
	Total evals	16028 [1x]	4417 [1x]	4639 [1x]	6238 [1x]
Py-DFBOLS $n+2$	Runtime	402s [8.5x]	70s [7.2x]	61s [6.3x]	97s [7.4x]
	Total evals	16832 [1.1x]	5213 [1.2x]	5240 [1.1x]	9369 [1.5x]
Py-DFBOLS $2n+1$	Runtime	705s [14.9x]	207.6s [21.4x]	191.1s [19.8x]	224.2s [17.1x]
	Total evals	14777 [0.9x]	7800 [1.8x]	8171 [1.8x]	11632 [1.9x]
Py-DFBOLS $O(n^2)$	Runtime	3042s [64.2x]	3698s [380.6x]	3848s [399.4x]	4173s [318.4x]
	Total evals	16802 [1.0x]	20922 [4.7x]	23587 [5.1x]	28390 [4.6x]

Table 5.1. Runtimes and total evaluations of all objectives (until solver termination, not necessarily a specific accuracy tolerance), for DFO-LS and Py-DFBOLS. All runs used a maximum budget of $200(n+1)$ objective evaluations, and all noisy results are an average of running 10 instances of each problem with noise level $\sigma = 10^{-2}$; similarly for DFO-LS with smooth objectives. Values are raw (runtime in seconds) and ratio compared to DFO-LS. For Py-DFBOLS, $\mathcal{O}(n^2) = (n+1)(n+2)/2$ interpolation points. The problem collection is (MW).

For smooth problems, DFO-LS is still able to solve essentially the same proportion of problems as (Py-)DFBOLS. For noisy problems, the results are more mixed: overall, DFO-LS does slightly worse for Gaussian noise, but slightly better for χ^2 noise. These results are the same when looking at all problems, or just nonzero residual problems. This reinforces our previous conclusions, and gives us confidence that a Gauss-Newton framework for DFO is a suitable choice, and is robust to the level of accuracy required for a given problem.

Runtime Comparison The use of linear models in DFO-LS also leads to a reduced linear algebra cost. As described in Section 3.3.4, the interpolation system for DFO-LS is of size n, compared to (Py-)DFBOLS, where the system is of size $p+n+2$ for $p+1 \geq n+2$ interpolation points; i.e. the (Py-)DFBOLS interpolation system is at least twice the size of the DFO-LS system. Depending on which preprocessing step is used (factorisation in Py-DFBOLS or low-rank updates in DFBOLS) and the size of m, we would therefore expect the computational cost of solving the (Py-)DFBOLS system to be at least 4–8 times larger that of DFO-LS (depending on whether the preprocessing or the solve step dominates the cost). To verify this, in this section we compare the runtime of DFO-LS with Py-DFBOLS. Since DFBOLS is implemented in Fortran, a runtime comparison against the Python implementation of DFO-LS will not produce meaningful results. As Py-DFBOLS uses the more expensive factorisation-based preprocessing step, we would expect the performance penalty to be nearer the upper end of the 4–8 times range (as a minority of the Moré & Wild test problems have $m \geq p+n$).

The wall time required by each solver to run the above testing (with a budget of $200(n+1)$ objective evaluations) on a Lenovo ThinkCentre M900 (with one 64-bit Intel i5 processor,

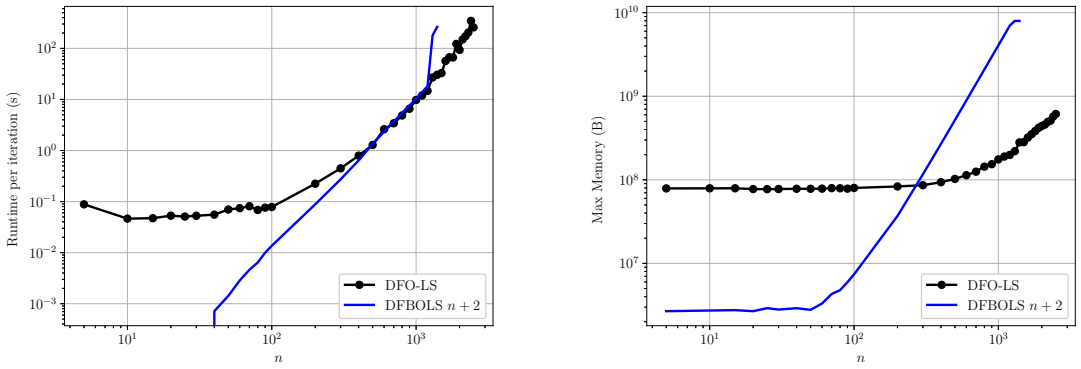
8GB of RAM), is shown in Table 5.1 for both smooth and noisy evaluations. We find that DFO-LS is 6–9 times faster than Py-DFBOLS with $n + 2$ points, 14–22 times faster than Py-DFBOLS with $2n + 1$ points, and 64–400 times faster than Py-DFBOLS with $(n + 1)(n + 2)/2$ points. In all cases, this is a substantial improvement, particularly given the small difference in performance (measured in function evaluations) between DFO-LS and (Py-)DFBOLS described above.

In Table 5.1, we also show the total number of objective evaluations required by each solver (summed over all instances of all problems). We see that DFO-LS uses a similar number of evaluations as Py-DFBOLS for smooth problems, and fewer evaluations (in some cases substantially so) for noisy problems. Firstly, the fact that this does not correlate with runtime indicates that objective evaluation cost is not affecting the runtime results. Secondly, since we saw above that DFO-LS and (Py-)DFBOLS require a similar number of evaluations to achieve a given accuracy level, this is perhaps evidence that there is scope to implementing more sophisticated termination criteria in both solvers.

5.1.2 Scalability Features

We saw above that DFO-LS runs faster than Py-DFBOLS due to the lower cost of solving the interpolation linear system. Another important benefit is that storing the interpolation models for each residual requires only $\mathcal{O}(mn)$ memory, rather than $\mathcal{O}(mn^2)$ for quadratic models. These two observations together suggest that DFO-LS should scale to large problems better than DFBOLS—in this section we demonstrate this. We consider Problem 29 from Moré, Garbow & Hillstrom [155] (which is Problem 33 (INTEGREQ) in the (CR) set of test problems). This is a zero-residual least-squares problem (with $m = n$) for solving a one-dimensional integral equation, using an n-point discretisation of $(0, 1)$. We compare DFO-LS and DFBOLS with $n + 2$ interpolation points, as it has the smallest memory usage and runtime of all possible values.

In Figure 5.4 we compare the per-iteration runtime and peak memory usage of DFBOLS and DFO-LS as n increases. Note that we are comparing DFO-LS (implemented in Python) against DFBOLS (implemented in Fortran) rather than Py-DFBOLS, as used for the runtime comparisons above, to put ourselves at a substantial disadvantage. We see that for small n, DFBOLS has significantly lower runtime and memory requirements than DFO-LS (which is unsurprising, since it is implemented in Fortran rather than Python). However, as expected, both the runtime and memory usage increases much faster for DFBOLS than for DFO-LS as n is increased. For $n > 1200$, DFBOLS exceeds the memory capacity of the system. At this point, it has to store data on disk, and as a result the runtime increases very quickly. DFO-LS does not suffer from this issue, and can continue solving problems quickly



(a) Runtime per iteration *(b) Peak memory usage*

Figure 5.4. Comparison of runtime and peak memory usage of DFBOLS (original Fortran implementation with $n+2$ interpolation points) and DFO-LS for solving the discretised integral equation, as problem dimension n increases. The largest values tested were $n = 1400$ for DFBOLS and $n = 2500$ for DFO-LS.

for substantially larger n. For instance, DFO-LS solves the $n = 2500$ problem faster per iteration than DFBOLS solves the much smaller $n = 1400$ problem.

5.2 Numerical Studies of New DFO-LS Features

In this section, we test the new features of DFO-LS and showcase the successful ones that are chosen as defaults, with the remaining features available as options. Section 5.3 then compares DFO-LS with its default settings against state-of-the-art DFO least-squares solvers.

5.2.1 Testing Methodology

In the following sections, we provide numerical comparisons of different features of DFO-LS, and compare it to other solvers. Our methodology for measuring performance with data and performance profiles follows Section 2.4, including the adaptive choice accuracy level for noisy problems from Section 2.4.2. The collections of test problems are (MW) and (CR) from Section 2.4.1; for noisy problems we ran 10 instances of each problem[6] with noise level $\sigma = 10^{-2}$, and treat each of these problem instances as a separate problem to be solved when constructing profiles. We used a maximum budget of $10^4(n+1)$ evaluations for the Moré and Wild problems (MW). Particularly for noisy problems, we are interested in a regime where objective evaluations are cheap, and we are concerned with the robustness of each solver—how many problems can it solve, if budget were not an issue. Since this budget is much larger than is often used for testing (e.g. [240, 47]), we show data profiles

[6] By default, DFO-LS chooses its initial set Y_0 using random orthogonal directions, so we also run 10 instances of DFO-LS for noiseless problems.

with a log-scale for budget, so we can easily compare solvers both for large budgets (to check robustness) and for realistically small budgets. For the CUTEst problems (CR), we used a much smaller budget of $50(n+1)$ evaluations, to represent the other regime, where objectives are expensive to evaluate.

The default parameter values for DFO-LS are given in Remark 3.2, although unlike in Section 5.1, for noisy problems we set the alternative defaults as specified in Section 3.4.1. For all solvers, we use an initial trust-region radius of $\Delta_0^{\text{init}} = 0.1 \max(\|\mathbf{x}_0\|_\infty, 1)$ (the default for DFO-LS) and final trust region radius $\rho_{\text{end}} = 10^{-8}$ where possible. Other parameters are all set to their default values.

5.2.2 Reduced Initialisation Cost

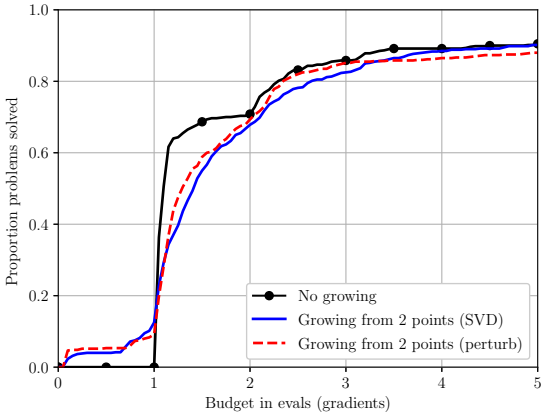
In Figure 5.5, we consider the CUTEst problems (CR) with noiseless evaluations. We compare the basic implementation of DFO-LS against DFO-LS with a reduced initialisation cost of 2, $n/4$ and $n/2$ function evaluations (growing the direction space via both mechanisms described in Section 3.2.2 above: modifying J_k using its SVD, and perturbing the trust-region step). Using our small budget of $50(n+1)$ evaluations, we show data profiles in two settings: for a small budget ($5(n+1)$ evaluations) with low accuracy $\tau = 10^{-1}$, and the full budget with high accuracy $\tau = 10^{-5}$.

In the short budget, low accuracy plots, we see the benefit of a reduced initialisation cost—we are able to solve a notable fraction of the problems to low accuracy with very few evaluations; less than the number required to perform a single gradient evaluation. We also see the tradeoff of this benefit, which is a lower performance at small budgets (1–3 gradients). However, the long budget, high accuracy plots in Figure 5.5 show that we do not lose robustness regardless of the initialisation cost, as the small-budget performance loss does not perpetuate to longer budgets. The difference between initialising with 2, $n/4$ and $n/2$ points is not substantial. Comparing the two mechanisms for increasing the model dimensionality, we find that the SVD approach performs similarly to the perturbed trust-region approach for small budgets, but better matches DFO-LS with a full initialisation set. We note that all our test problems are least-squares or nonlinear systems, so $m \geq n$ in all cases.

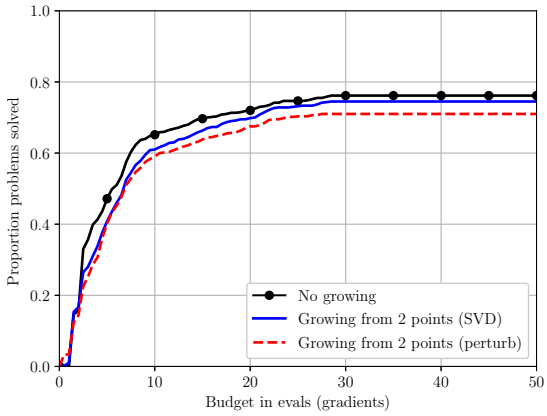
5.2.3 Sample Averaging

To demonstrate the impact of using sample averaging, Figure 5.6 shows data profiles with averaging strategies $N \in \{1, 2, 5, 10, 30, \max(1, \lfloor \Delta_k^{-1} \rfloor)\}$ in (3.28). Each of the N samples for a given $\mathbf{x}_k$ are counted towards the maximum computational budget of $10^4(n+1)$ evaluations.

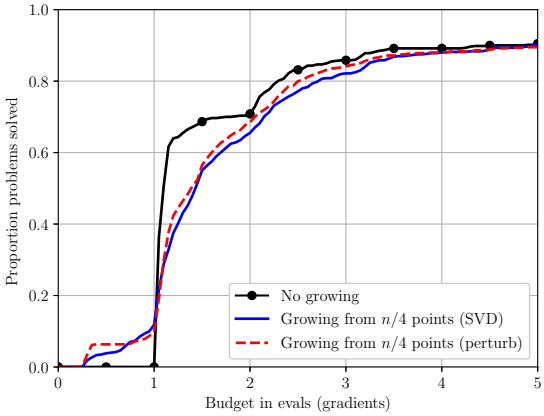
Unsurprisingly, we see that using a larger number of samples can improve the robustness of DFO-LS. Of course, to achieve this robustness, a proportionally larger number of evaluations



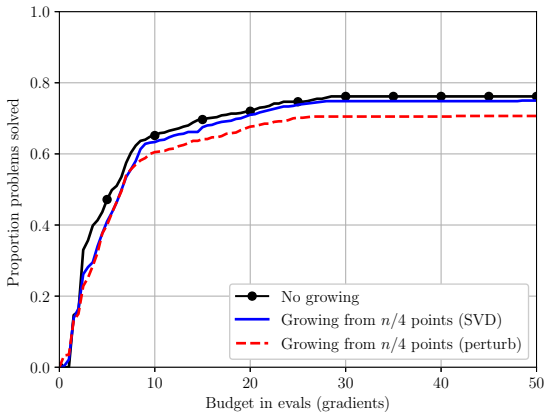
(a) Short budget, 2 starting points, $\tau = 10^{-1}$



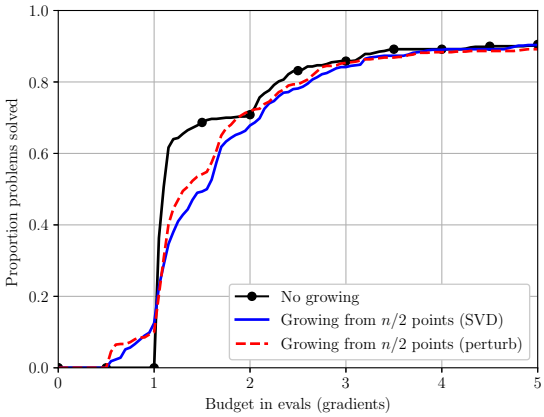
(b) Long budget, 2 starting points, $\tau = 10^{-5}$



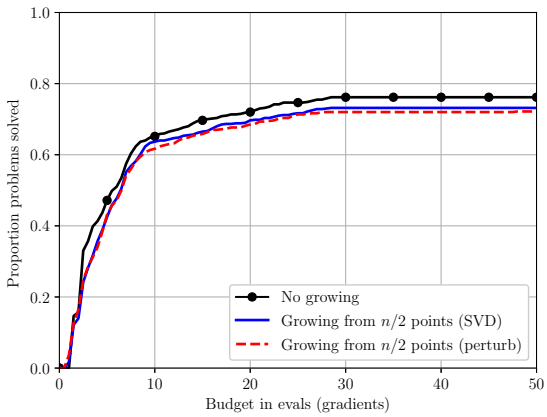
(c) Short budget, $n/4$ starting points, $\tau = 10^{-1}$



(d) Long budget, $n/4$ starting points, $\tau = 10^{-5}$

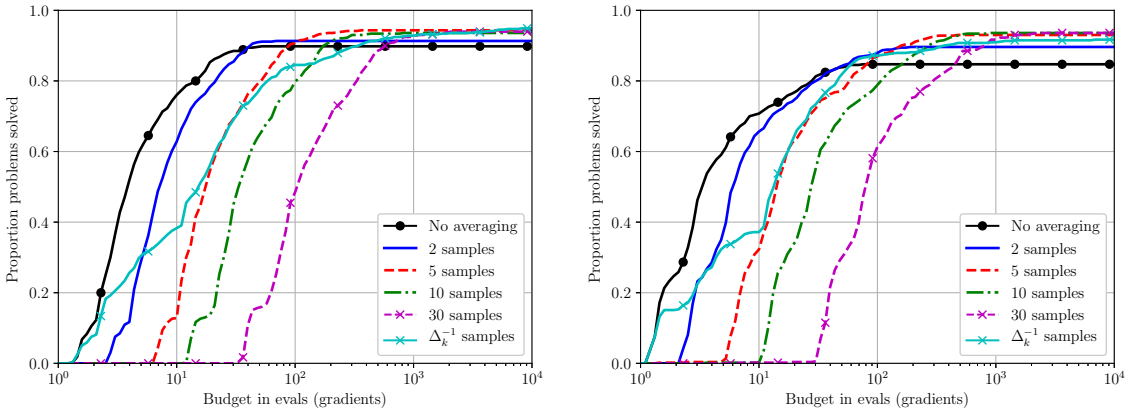


(e) Short budget, $n/2$ starting points, $\tau = 10^{-1}$



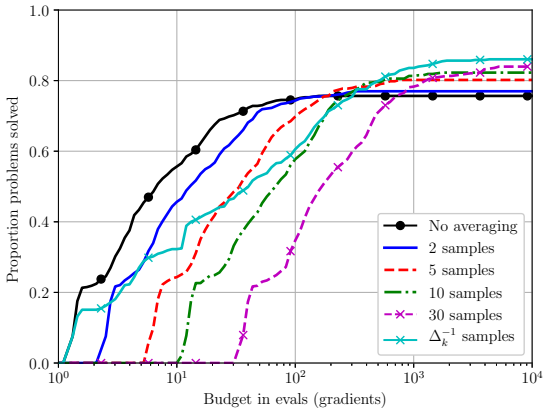
(f) Long budget, $n/2$ starting points, $\tau = 10^{-5}$

Figure 5.5. Data profiles showing the impact of the reduced initialisation cost of DFO-LS (using $n+1$ interpolation points) against using the full initial set, for smooth objectives. Results an average of 10 runs in each case. The problem collection is (CR).



(a) Multiplicative Gaussian noise

(b) Additive Gaussian noise



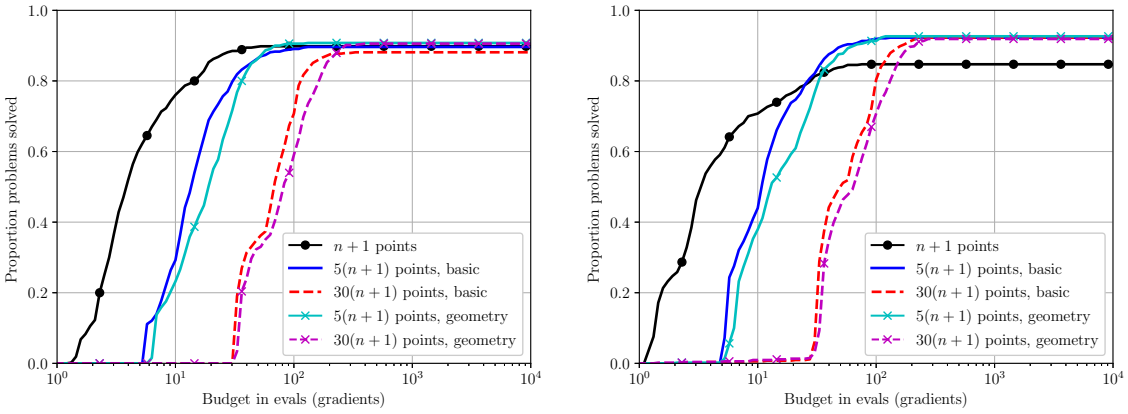
(c) Additive χ^2 noise

Figure 5.6. Comparison of different sample averaging methods for DFO-LS (using $n+1$ interpolation points). We are using noisy objective evaluations with $\sigma = 10^{-2}$, high accuracy $\tau = 10^{-5}$, and an average of 10 runs for each solver. The problem collection is (MW).

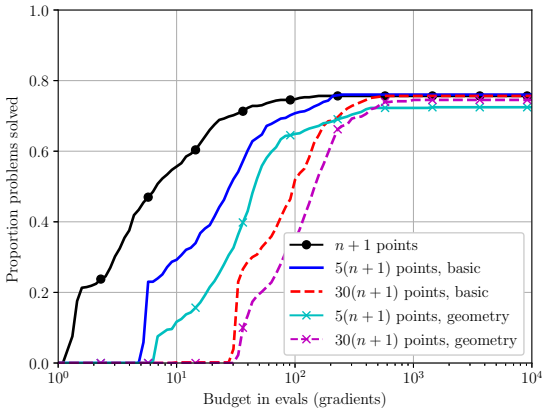
are required, so for small-to-medium budgets (in serial) we lose in performance. This does not take into account the benefits of parallelisation that may be available when sample averaging is used. We also notice that using $N = \mathcal{O}(\Delta_k^{-1})$ can provide a compromise—it still makes progress for small budgets, but manages to achieve a reasonable level of robustness overall.

5.2.4 Regression Models

In practice, we find that the geometry-based moves perform similarly to or slightly better than momentum-based ones (see Section 3.4.3.2 for details), so we do not show results for the latter mechanism. Figure 5.7 compares the remaining two techniques with varying numbers of regression points ($p+1 = c(n+1)$ for $c \in \{5, 30\}$) against interpolation models ($p+1 = n+1$). We see that using a larger sample set improves the robustness of DFO-LS, particularly for additive Gaussian noise, and this improvement (for $p+1 = c(n+1)$) is



(a) Multiplicative Gaussian noise *(b) Additive Gaussian noise*



(c) Additive χ^2 noise

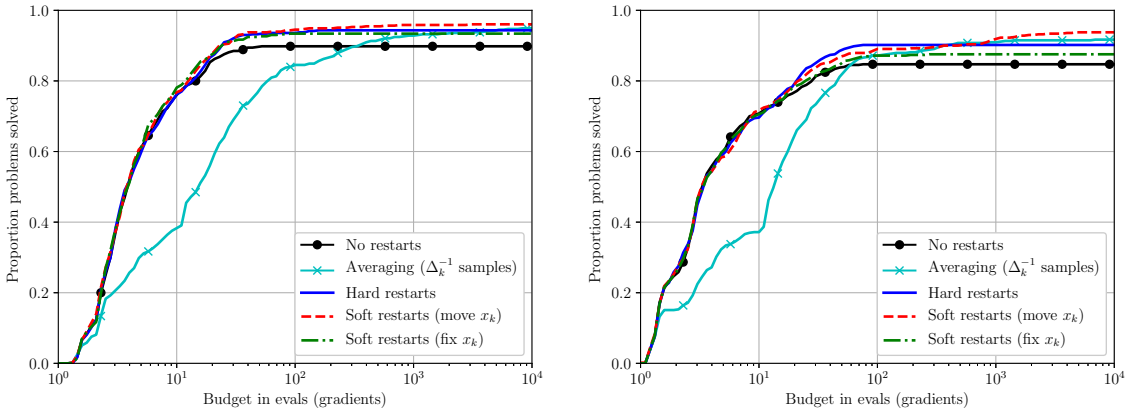
Figure 5.7. Comparison of different regression methods for DFO-LS. We are using noisy objective evaluations with $\sigma = 10^{-2}$, high accuracy $\tau = 10^{-5}$, and an average of 10 runs for each solver. The problem collection is (MW).

generally comparable to, or slightly worse than, the use of sample averaging (with c samples at each point). This aligns with the argument in Appendix C that we would expect regression to be slightly less robust than sample averaging. The geometry-based mechanism for moving multiple points makes the algorithm progress more slowly, as indicated by the performance profiles, while at times providing a slight improvement over the 'basic' approach (moving one point per iteration).

5.2.5 Multiple Restarts

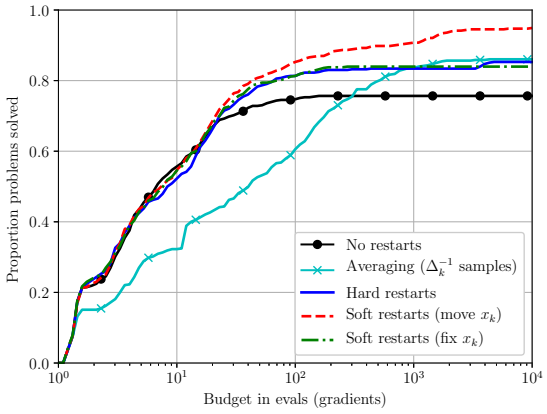
Figure 5.8 compares the different restart methods against sample averaging (Δ_k^{-1} samples at every point). All runs use auto-detection of restarts and the optional noise-based termination criterion (3.25).

We see that soft restarts (moving $\mathbf{x}_k$) is the most successful restarts mechanism, followed by hard restarts, then soft restarts (fixing $\mathbf{x}_k$), and that all these mechanisms are better



(a) Multiplicative Gaussian noise

(b) Additive Gaussian noise



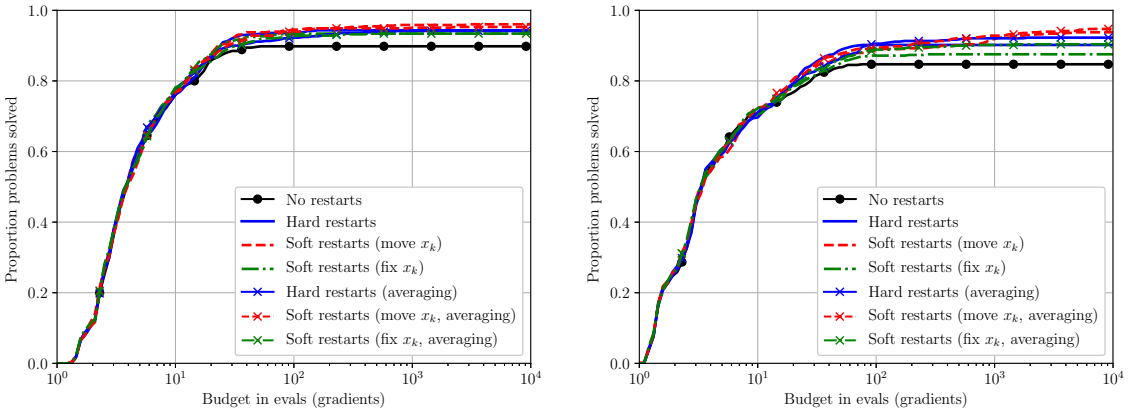
(c) Additive χ^2 noise

Figure 5.8. Comparison of different multiple restart strategies for DFO-LS (using $n+1$ interpolation points). We are using noisy objective evaluations with $\sigma = 10^{-2}$, high accuracy $\tau = 10^{-5}$, and an average of 10 runs for each solver. The problem collection is (MW).

than DFO-LS without any noise-based features. Compared to the case of using averaging with Δ_k^{-1} samples at every point, soft restarts (moving $\mathbf{x}_k$) achieve a similar or better level of robustness with many fewer objective evaluations—this is most clearly observed for small budgets, indicating that using Δ_k^{-1} samples is more pessimistic than necessary (even though it is well below the $\mathcal{O}(\Delta_k^{-4})$ evaluations required in theory [50]).

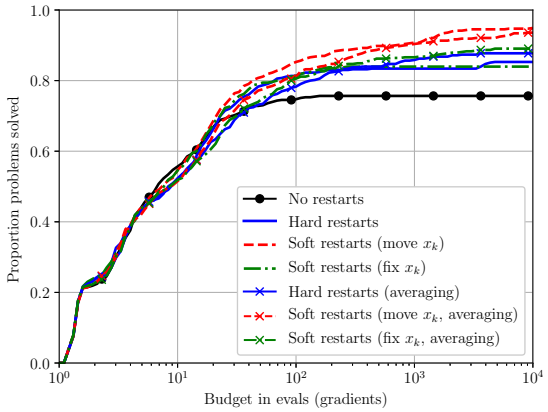
The improvements in robustness from using multiple restarts are obvious at the end of the full budget of 10^4 gradients, but there are still benefits to be found at much smaller budgets (e.g. $\mathcal{O}(100)$ gradients). As a result of these benefits, the soft restarts (moving $\mathbf{x}_k$) mechanism is activated by default in DFO-LS for noisy problems.

Next, we consider the impact of using increased levels of sample averaging with every restart. The reason for this is that after every restart, we hope to be closer to the desired solution, so an increased amount of averaging may help distinguish points near to this



(a) Multiplicative Gaussian noise

(b) Additive Gaussian noise



(c) Additive χ^2 noise

Figure 5.9. Comparison of multiple restarts with and without sample averaging for DFO-LS (using $n+1$ interpolation points). We are using noisy objective evaluations with $\sigma = 10^{-2}$, high accuracy $\tau = 10^{-5}$, and an average of 10 runs for each solver. The problem collection is (MW).

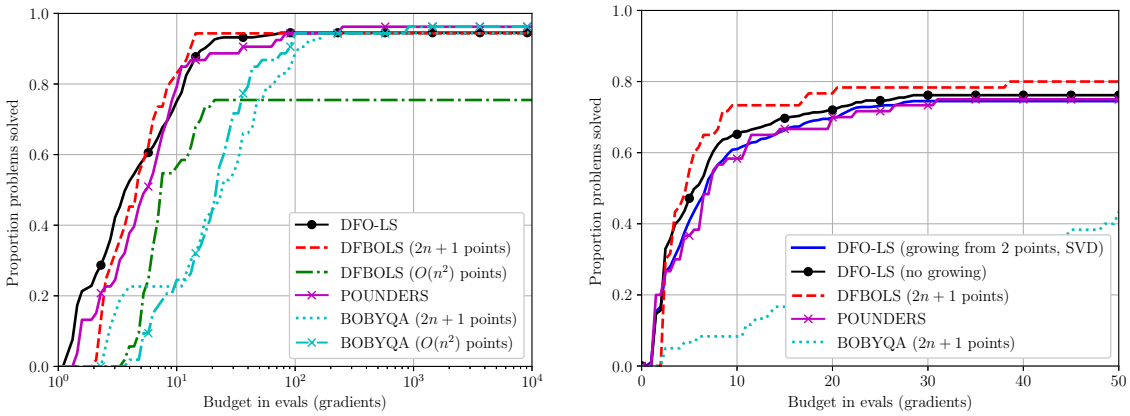
optimum. To achieve this, in (3.28) we use

$$N = \text{nsamples}(\rho_k, \Delta_k, k, n_{\text{restarts}}) = \min\{n_{\text{restarts}} + 1, 30\}. \tag{5.1}$$

Figure 5.9 shows that augmenting multiple restarts with sample averaging improves the robustness of hard and soft restarts (fixing $\mathbf{x}_k$), but not for the default mechanism (soft restarts moving $\mathbf{x}_k$). Ultimately, using soft restarts (moving $\mathbf{x}_k$) is better than the other two restart mechanisms, with or without sample averaging. Hence, we do not use any sample averaging in DFO-LS by default.

5.3 Benchmark Comparison of DFO-LS

In this section we compare the performance of DFO-LS against DFBOLS with $2n+1$ and $(n+1)(n+2)/2$ points, POUNDERS, and BOBYQA with $2n+1$ and $(n+1)(n+2)/2$



(a) Problem collection (MW) *(b) Problem collection (CR)*

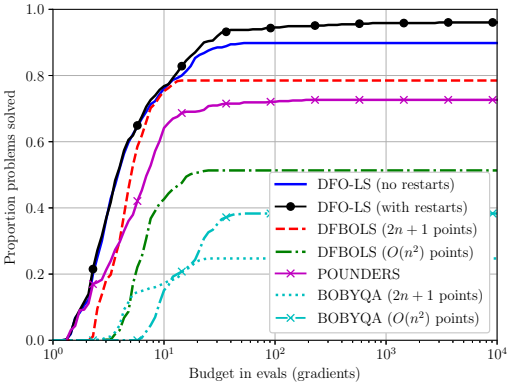
Figure 5.10. Comparison of the basic implementation of DFO-LS (using $n+1$ interpolation points) with DFBOLS for smooth objective evaluations and high accuracy $\tau = 10^{-5}$. For DFBOLS and BOBYQA, $2n+1$ and $\mathcal{O}(n^2) = (n+1)(n+2)/2$ are the number of interpolation points. For DFO-LS, results are an average of 10 runs. In (b), we show results using the full initialisation cost of $n+1$ evaluations, and a reduced cost of 2 evaluations (using the SVD method).

points. DFO-LS uses $p_{\text{init}} = p = n$ interpolation points and the default values for all other parameters, unless otherwise specified. For all solvers, we use the computational budget, and initial and final trust region radii as in Section 5.2.1, with accuracy level $\tau = 10^{-5}$.

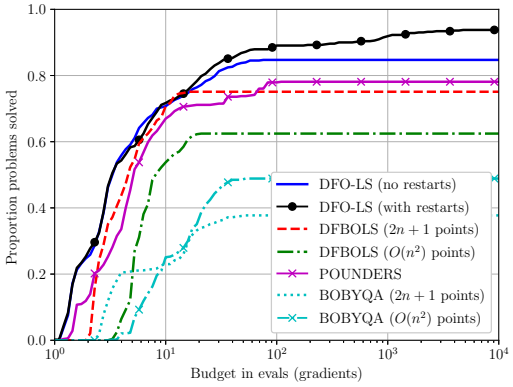
Figure 5.10 shows results for smooth (noiseless) objective functions for both the (MW) and (CR) test sets. Since DFO-LS uses randomised initial points, we show an average result over 10 runs. For the (CR) set, we do not show DFBOLS or BOBYQA with $(n+1)(n+2)/2$ points, as in most cases the initialisation cost will use almost all of the available budget. DFO-LS performs similarly to both DFBOLS and POUNDERS, which is to be expected given the similarity of these algorithms. All these solvers outperform BOBYQA, particularly at low budgets, as they can exploit the nonlinear least-squares structure of the problem. Note that for (CR), initialising DFO-LS with only 2 evaluations yields only slightly worse performance, but gains the benefit of decreasing the objective at a very low cost (as shown in Section 5.2.2).

Similarly, for noisy functions (from the (MW) set), Figure 5.11 shows DFO-LS with and without restarts compared to the same solvers as above. It is in this scenario that the flexibility of DFO-LS becomes evident—its ability to adjust the default algorithm parameters in the presence of noisy evaluations allows it to solve a larger proportion of problems than both DFBOLS and POUNDERS—as well as BOBYQA—and this robustness is further improved, by a significant margin, by the use of multiple restarts.

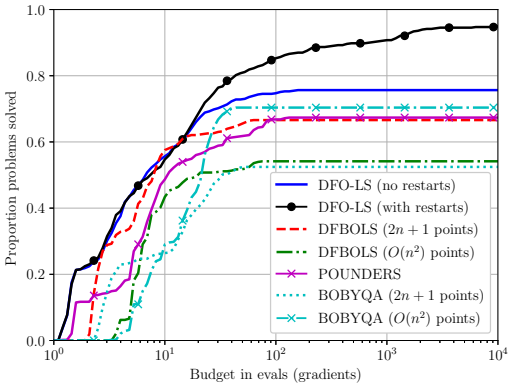
We observed in Section 5.1.1 that linear residual models to have a small performance penalty compared to quadratic residual models for noisy problems. Thus, that DFO-LS performs better than DFBOLS and POUNDERS here gives strong weight to the benefit of the novel algorithmic features in DFO-LS.



(a) Multiplicative Gaussian noise

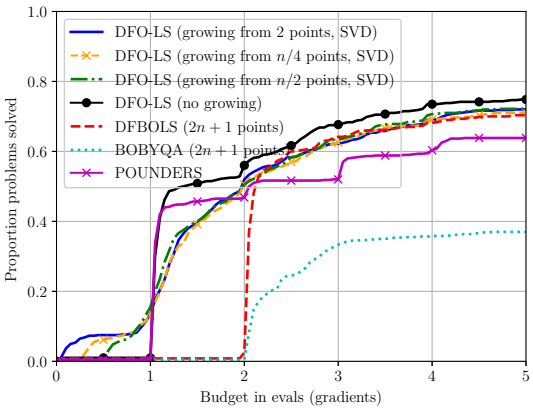


(b) Additive Gaussian noise

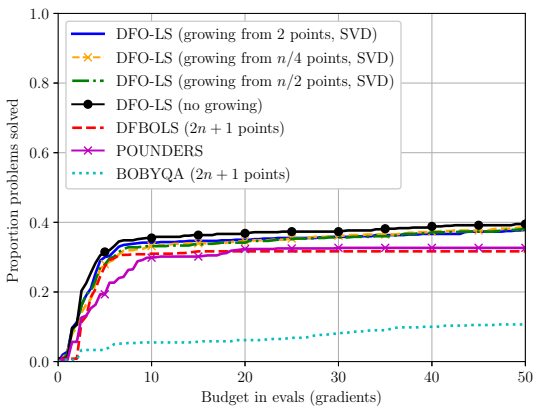


(c) Additive χ^2 noise

Figure 5.11. Comparison of the basic implementation of DFO-LS (using $n+1$ interpolation points) with DFBOLS, POUNDERS and BOBYQA for noisy objective evaluations with $\sigma = 10^{-2}$ and high accuracy $\tau = 10^{-5}$. For DFBOLS and BOBYQA, $2n+1$ and $\mathcal{O}(n^2) = (n+1)(n+2)/2$ are the number of interpolation points. Results shown are an average of 10 runs for each solver. The problem collection is (MW).



(a) Short budget, $\tau = 10^{-1}$



(b) Long budget, $\tau = 10^{-5}$

Figure 5.12. Comparison of the reduced initialisation cost of DFO-LS (using $n+1$ interpolation points, SVD method) against using the full initial set and DFBOLS, POUNDERS and BOBYQA, for objectives with additive Gaussian noise, $\sigma = 10^{-2}$. For DFBOLS and BOBYQA, $2n+1$ is the number of interpolation points. Results are an average of 10 runs in each case. The problem collection is (CR).

Expensive & Noisy Problems As discussed in Chapter 1, the two regimes that have been the focus of our attention—'expensive' and 'noisy'—are not mutually exclusive. Here, we demonstrate that DFO-LS continues to perform well when the two regimes coincide. In Figure 5.12, we run all solvers on the (CR) problem set with additive Gaussian noise. The DFO-LS runs use the default settings for noisy problems (i.e. slower trust region decrease parameters, multiple restarts). The results are very similar to the smooth case (see Figures 5.5 and 5.10b): the reduced initialisation cost allows progress to be made within $n + 1$ objective evaluations for some problems, at the cost of reduced small-budget performance, but achieves similar overall robustness, with performance for long budgets at high accuracy levels. In addition, DFO-LS with full initialisation cost outperforms DFBOLS, POUNDERS and BOBYQA, in both the low and high accuracy/budget regimes.

Chapter 6

DFO for General Minimisation

In this chapter we consider the case of general objective problems (2.1), as opposed to nonlinear least-squares problems (2.6). The main goal of this chapter is to translate some of the novel algorithmic features from DFO-LS (Chapter 3) to Powell's BOBYQA algorithm [186] for solving (2.1) (or in practice the bound-constrained version (2.5)). In Section 6.1 we introduce our general minimisation solver, which we call Py-BOBYQA[1] as it is a Python-based solver that is built on the original (Fortran) software [242]. Then, in Section 6.2, we present global convergence results and a worst-case complexity analysis for Py-BOBYQA, based on the approach from Chapter 4. In Section 6.3, we show numerical results for this algorithm for both smooth and noisy problems; here, we also show that the multiple restarts mechanism from Section 3.4.2 can also be useful for escaping local minima. Lastly, in Section 6.4 we extend this idea and consider the performance of our multiple restarts framework on global optimisation problems.

6.1 The Py-BOBYQA Algorithm

The overall algorithmic structure of (Py-)BOBYQA is the same as Algorithm 3.1: we construct an interpolation-based model for f, calculate a step to minimise this model inside a trust region, and perform one of several phases (safety, successful, model-improving, unsuccessful) depending on the outcome. The most important difference is that the model $m_k(\mathbf{s}) \approx f(\mathbf{x}_k + \mathbf{s})$ is built by directly interpolating $f(\mathbf{y}_t)$ for $\mathbf{y}_t \in Y_k := \{\mathbf{y}_0 = \mathbf{x}_k, \dots, \mathbf{y}_p\}$ and is typically quadratic. Specifically, for an interpolation set of size $p+1 \in \{n+1, \dots, (n+1)(n+2)/2\}$, we construct

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) = c_k + \mathbf{g}_k^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_k \mathbf{s}, \tag{6.1}$$

[1] Available from `https://github.com/numericalalgorithmsgroup/pybobyqa`

satisfying the interpolation (not regression) conditions

$$m_k(\mathbf{y}_t - \mathbf{x}_k) = f(\mathbf{y}_t), \quad \forall t = 0, \ldots, p. \tag{6.2}$$

If $p + 1 = n + 1$ we have linear interpolation and hence a linear local model (with $H_k = 0$), otherwise we have underdetermined or fully-determined quadratic interpolation, and find c_k, $\mathbf{g}_k$ and H_k by selecting the solution given by (2.36); i.e. with minimal change in the Hessian. The full Py-BOBYQA algorithm is given in Algorithm 6.1, where the size of the interpolation set $p+1$ is a user-specified input (defaulting to $2n+1$ for smooth problems and $(n+1)(n+2)/2$ for noisy problems), which can be set depending on the number of initial evaluations the user can afford. Additionally, the implementation of Py-BOBYQA, like DFO-LS and BOBYQA, uses simplified geometry management (as described in Section 3.3.1)

Remark 6.1. Compared to DFO-LS (Algorithm 3.1), Py-BOBYQA is the same, except it does not allow for a reduced initialisation cost, has a reduced range of allowable values for the input p, and when we evaluate the objective in line 19 of Algorithm 6.1, we receive just $f(\mathbf{x}_k + \mathbf{s}_k)$ rather than the vector $\mathbf{r}(\mathbf{x}_k + \mathbf{s}_k)$. The default values of the trust-region parameters in Py-BOBYQA are the same as DFO-LS (listed in Remark 3.2).

Improvements from original BOBYQA The goal of implementing Py-BOBYQA is to endow it with some of the key features of DFO-LS in order to improve its robustness to noise. Given the already-strong performance of BOBYQA, we aim to enhance—rather than redesign—BOBYQA by incorporating features from DFO-LS that naturally translate into the setting of quadratic interpolation models. We now describe the ways in which Py-BOBYQA is an improvement over BOBYQA.

Firstly, the user is able to specify $p + 1 = n + 1$, compared to $p + 1 \geq n + 2$ as required by BOBYQA. However, Py-BOBYQA uses the default option $2n + 1$ for smooth problems and $(n + 1)(n + 2)/2$ for noisy problems; see Section 6.3 for a discussion of this choice. Py-BOBYQA also has a larger range of termination conditions than BOBYQA, including:

1. Small objective value: we allow termination when $f(\mathbf{x}_k) \leq f_{\text{abs}}$, for user-specified parameter f_{abs} (default $f_{\text{abs}} = -10^{20}$; i.e. effectively inactive);

2. Small trust region: we terminate when $\rho_k \leq \rho_{\text{end}}$ (default $\rho_{\text{end}} = 10^{-8}$);

3. Computational budget: we terminate after a given number of evaluations of the objective (default $\min(100(n + 1), 1000)$);

4. Slow decrease in the objective: this is defined in Section 3.3.3, but we use $f(\mathbf{x}_{k_{(i-K)}}) - f(\mathbf{x}_{k_i})$ in (3.24) rather than log-decrease (since we may no longer have $f \geq 0$); and

Algorithm 6.1 Py-BOBYQA: Python Implementation of BOBYQA.

Input: Starting point $\mathbf{x}_0 \in \mathbb{R}^n$, initial trust region radius $\Delta_0^{\text{init}} > 0$ and integer p, the size of the interpolation set, where $n+1 \leq p+1 \leq (n+1)(n+2)/2$.

Parameters: maximum trust-region radius $\Delta_{\max} \geq \Delta_0^{\text{init}}$, criticality threshold $\epsilon_C > 0$, criticality scaling $\mu > 0$, minimum trust-region radius $0 < \rho_{\text{end}} < \Delta_0^{\text{init}}$, trust-region radius scalings $0 < \gamma_{\text{dec}} < 1 < \gamma_{\text{inc}} \leq \overline{\gamma}_{\text{inc}}$ and $0 < \alpha_1 < \alpha_2 < 1$, acceptance thresholds $0 < \eta_1 \leq \eta_2 < 1$, safety reduction factor $0 < \omega_S < 1$, safety step threshold $0 < \gamma_S < 2c_1/(1+\sqrt{1+2c_1})$, poisedness constant $\Lambda \geq 1$, and Boolean flag `NOISY` indicating the presence of noise in the objective.

1: Build an initial interpolation set $Y_0 \subset B(\mathbf{x}_0, \Delta_0^{\text{init}})$ of size $p+1$, with $\mathbf{x}_0 \in Y_0$. Set $\rho_0^{\text{init}} = \Delta_0^{\text{init}}$.
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: **if** `NOISY` **and** *all values* $\{f(\mathbf{y}) : \mathbf{y} \in Y_k\}$ *are within noise level of* $f(\mathbf{x}_k)$ **then**
4: Call restart: set $\Delta_{k+1}^{\text{init}} = \rho_{k+1}^{\text{init}} = \Delta_0^{\text{init}}$ and build Y_{k+1} as per Section 3.4.2, then **goto** line 2.
5: **end if**
6: Given $\mathbf{x}_k$ and Y_k, construct the model m_k^{init} (6.1) by solving the interpolation problem (2.36).
7: **if** $\|\mathbf{g}_k^{\text{init}}\| \leq \epsilon_C$ **then**
8: Criticality Phase: using Algorithm 2.2, modify Y_k and find $\Delta_k \leq \Delta_k^{\text{init}}$ such that Y_k is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$ and $\Delta_k \leq \mu\|\mathbf{g}_k\|$, where $\mathbf{g}_k$ is the gradient of the new m_k. Set $\rho_k = \min(\rho_k^{\text{init}}, \Delta_k)$.
9: **else**
10: Set $m_k = m_k^{\text{init}}$, $\Delta_k = \Delta_k^{\text{init}}$ and $\rho_k = \rho_k^{\text{init}}$.
11: **end if**
12: Approximately solve the trust-region subproblem (1.6) to get a step $\mathbf{s}_k$ satisfying Assumption 4.4.
13: **if** $\|\mathbf{s}_k\| < \gamma_S \rho_k$ **then**
14: Safety Phase: Set $\mathbf{x}_{k+1} = \mathbf{x}_k$ and $\Delta_{k+1}^{\text{init}} = \max(\rho_k, \omega_S \Delta_k)$, and form Y_{k+1} by making Y_k Λ-poised in $B(\mathbf{x}_{k+1}, \Delta_{k+1}^{\text{init}})$.
15: If $\Delta_{k+1}^{\text{init}} = \rho_k$, set $(\rho_{k+1}^{\text{init}}, \Delta_{k+1}^{\text{init}}) = (\alpha_1\rho_k, \alpha_2\rho_k)$, otherwise set $\rho_{k+1}^{\text{init}} = \rho_k$.
16: If $\rho_{k+1}^{\text{init}} \leq \rho_{\text{end}}$: call restart if `NOISY`, else **terminate**.
17: **goto** line 2.
18: **end if**
19: Evaluate $f(\mathbf{x}_k + \mathbf{s}_k)$ and calculate ratio R_k (3.3).
20: Accept/reject step and update trust region radius: set

$$\mathbf{x}_{k+1} = \begin{cases} \mathbf{x}_k + \mathbf{s}_k, & R_k \geq \eta_1, \\ \mathbf{x}_k, & R_k < \eta_1, \end{cases} \quad \text{and} \quad \Delta_{k+1}^{\text{init}} = \begin{cases} \min(\max(\gamma_{\text{inc}}\Delta_k, \overline{\gamma}_{\text{inc}}\|\mathbf{s}_k\|), \Delta_{\max}), & R_k \geq \eta_2, \\ \max(\gamma_{\text{dec}}\Delta_k, \|\mathbf{s}_k\|, \rho_k), & \eta_1 \leq R_k < \eta_2, \\ \max(\min(\gamma_{\text{dec}}\Delta_k, \|\mathbf{s}_k\|), \rho_k), & R_k < \eta_1. \end{cases} \tag{6.3}$$

21: **if** $R_k \geq \eta_1$ **then**
22: Successful Phase: Form $Y_{k+1} = Y_k \cup \{\mathbf{x}_{k+1}\} \setminus \{\mathbf{y}\}$ for some $\mathbf{y} \in Y_k$ and set $\rho_{k+1}^{\text{init}} = \rho_k$.
23: If objective decrease is too slow: call restart if `NOISY`, else **terminate**.
24: **else if** `NOISY` **and** *restart auto-detected* **then**
25: Restart Auto-Detection: Call restart.
26: **else if** Y_k *is not* Λ*-poised in* $B(\mathbf{x}_k, \Delta_k)$ **then**
27: Model Improvement Phase: Form Y_{k+1} by making Y_k Λ-poised in $B(\mathbf{x}_{k+1}, \Delta_{k+1}^{\text{init}})$ and set $\rho_{k+1}^{\text{init}} = \rho_k$.
28: **else**
29: Unsuccessful Phase: Set $Y_{k+1} = Y_k$, and if $\Delta_{k+1}^{\text{init}} = \rho_k$, set $(\rho_{k+1}^{\text{init}}, \Delta_{k+1}^{\text{init}}) = (\alpha_1\rho_k, \alpha_2\rho_k)$, otherwise set $\rho_{k+1} = \rho_k$.
30: If $\rho_{k+1}^{\text{init}} \leq \rho_{\text{end}}$: call restart if `NOISY`, else **terminate**.
31: **end if**
32: **end for**

5. All objective values are within noise level (if `NOISY=True` in Algorithm 6.1).

Finally, Py-BOBYQA includes several strategies aimed to improve its robustness to noise. Specifically, it implements different default parameter values (as per Section 3.4.1) and multiple restarts (as per Section 3.4.2). In this setting, however, we must modify the automatic detection of restarts to be based on the slope and correlation coefficients of both $\{(k, \log \|\mathbf{g}_k - \mathbf{g}_{k-1}\|)\}$ and $\{(k, \log \|H_k - H_{k-1}\|_F)\}$, compared to $\{(k, \log \|J_k - J_{k-1}\|_F)\}$ for DFO-LS. Py-BOBYQA also implements flexible sample averaging using (3.28), as per Section 3.4.3.1, but—as in DFO-LS—this feature is not used by default.

Simplifications from original BOBYQA For the purposes of a more streamlined code, we simplify the model construction process in Py-BOBYQA as compared to its original implementation in [186]. As in DFO-LS (Section 3.3.4), we solve the interpolation system by factorising the interpolation linear system resulting from (2.36) at each iteration, rather than updating the inverse of the associated matrix via low-rank updating (as described in Section 2.2.3). Following Section 3.3.2, this naturally allows us to select the interpolation point to replace with $\mathbf{x}_k + \mathbf{s}_k$ by $\mathbf{y}_t$, where

$$t = \underset{j=0,\ldots,p}{\arg\max} |\ell_j(\mathbf{x}_k + \mathbf{s}_k)| \cdot \max\left(\frac{\|\mathbf{y}_j - \mathbf{x}_k\|^4}{\Delta_k^4}, 1\right). \tag{6.4}$$

Finally, also following DFO-LS, we initialise Py-BOBYQA using random orthogonal directions rather than coordinate directions (see Section 3.3.5).

6.2 Theoretical Guarantees for Py-BOBYQA

Here, we demonstrate how the theoretical results from DFO-LS (Chapter 4) apply to Py-BOBYQA. When Algorithm 6.1 refers to making the interpolation set Λ-poised, it refers to the appropriate definition of poisedness from Chapter 2, depending on whether linear or quadratic models are used. As in Chapter 4, we prove results for a simplified Py-BOBYQA, called with the following settings:

- No noise is present in the objective (i.e. `NOISY=False`); and
- Termination conditions are not applied: $\rho_{\text{end}} = 0$ and no termination on slow objective decrease (line 30 of Algorithm 3.1).

In addition, we require that our objective is sufficiently smooth, as per Assumption 2.1, and that the model Hessians are uniformly bounded.

Assumption 6.2. *We assume that $\|H_k\| \leq \kappa_H$ for all k, for some $\kappa_H \geq 1$.*

We note that Assumption 6.2 is automatically satisfied for linear models, where $H_k = 0$. For quadratic models, we only have bounded Hessians—from (2.44)—when the interpolation set is Λ-poised for minimum Frobenius norm interpolation and we use $H_{\text{prev}} = 0$ in (2.36). In this case, the discussion in Remark 4.22 would apply here.

Under Assumptions 2.1 and 6.2, we conclude that whenever Algorithm 3.1 makes the interpolation set Λ-poised, we arrive at a fully linear model as per Definition 2.7; this follows from Theorems 2.12 and 2.23.

In this setting, the main results from DFO-LS hold for Py-BOBYQA.

Theorem 6.3. *Suppose Assumptions 2.1, 4.4 and 6.2 hold. Then Py-BOBYQA produces iterates $\{\mathbf{x}_k\}$ satisfying $\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0$. Moreover, if Assumption 4.16 holds then the number of iterations i_ϵ (i.e. the number of times a model m_k is built via (6.2)) until $\|\nabla f(\mathbf{x}_{i_\epsilon+1})\| < \epsilon$ is at most*

$$\left\lfloor \frac{4[f(\mathbf{x}_0) - f_{\text{low}}]}{\eta_1 c_1} \left(1 + \frac{\log \overline{\gamma}_{\text{inc}}}{|\log \alpha_3|}\right) \max\left(\kappa_H c_4^{-2}\epsilon^{-2}, c_4^{-1}c_5^{-1}\epsilon^{-2}, c_4^{-1}(\Delta_0^{\text{init}})^{-1}\epsilon^{-1}\right) + \frac{4}{|\log \alpha_3|} \max\left(0, \log\left(\Delta_0^{\text{init}} c_5^{-1}\epsilon^{-1}\right)\right)\right\rfloor \tag{6.5}$$

where c_1 is defined in Assumption 4.4, α_3 is defined in Lemma 4.15, and c_3, c_4 and c_5 are defined in Theorem 4.17.

Proof. The proofs for the corresponding results in Sections 4.2 and 4.3 apply without modification, since they only rely on fully linear models, rather than the explicit method used to achieve them. The only change is the proof of Lemma 4.14, where instead of (4.44), which was based on $f(\mathbf{x}_k) \geq 0$ for nonlinear least-squares problems, we use $f(\mathbf{x}_k) \geq f_{\text{low}}$ to conclude

$$f(\mathbf{x}_0) - f_{\text{low}} \geq |\mathcal{S}_{i_\epsilon}| \eta_1 c_1 \epsilon_g \min\left(\frac{\epsilon_g}{\kappa_H}, \rho_{\min}\right), \tag{6.6}$$

which gives us the constant factor $[f(\mathbf{x}_0) - f_{\text{low}}]$ in (6.5) rather than $f(\mathbf{x}_0)$ in (4.50). □

As in Corollary 4.18, we can summarise this result as follows:

Corollary 6.4. *Suppose Assumptions 2.1, 4.4, 4.16 and 6.2 hold. Then for $\epsilon \in (0, 1]$, the number of iterations i_ϵ (i.e. the number of times a model m_k is built via (6.2)) until Py-BOBYQA achieves $\|\nabla f(\mathbf{x}_{i_\epsilon+1})\| < \epsilon$ is at most $\mathcal{O}(\kappa_H \kappa_d^2 \epsilon^{-2})$, and the number of objective evaluations until i_ϵ is at most $\mathcal{O}(\kappa_H \kappa_d^2 p \epsilon^{-2})$, where $\kappa_d := \max(\kappa_{\text{ef}}, \kappa_{\text{eg}})$ and $p+1$ is the size of the interpolation set at each iteration.*

That is, Py-BOBYQA has the same iteration and evaluation complexity—including dependence on dimension (provided the interpolation set is of size $p = \mathcal{O}(n)$)—as the first-order complexity results from [83].

6.3 Numerical Results

We now present numerical results for Py-BOBYQA.[2] In our comparisons, we compare Py-BOBYQA with the original BOBYQA [186, 242], and (S)NOWPAC [16, 17, 18] (as described in Section 1.1.4)—we use NOWPAC for smooth problems and SNOWPAC for noisy problems, in line with their intended use. SNOWPAC requires both evaluations of the (noisy) objective and its standard deviation, and so receives more information than the other solvers. For noisy problems, we also compare against STORM for unbiased noise [50, Algorithm 3].[3] For STORM, theoretical convergence requires Δ_k^{-4} samples to check objective decrease, and Δ_k^{-4} regression points for model construction—however, for tractability, we follow [50] and use Δ_k^{-1} for both.

Our testing methodology is the same as Section 5.2.1, using the problem collections (MW) and (CFMR). Specifically, we use an initial trust-region radius of $\Delta_0^{\text{init}} = 0.1 \max(\|\mathbf{x}_0\|_\infty, 1)$ (the default for Py-BOBYQA) and final trust region radius $\rho_{\text{end}} = 10^{-8}$, and all other parameters set to their default values. We allow a budget of $10^4(n+1)$ evaluations for (MW) and $50(n+1)$ evaluations for (CFMR), but with a maximum runtime of 12 hours per problem instance. We run 10 problem instances for all solvers whenever we have noisy problems, and for Py-BOBYQA with smooth problems due to its initialisation using random directions.

6.3.1 Smooth Regime

Figure 6.1 compares the basic implementation of Py-BOBYQA (no sample averaging or restarts) with BOBYQA and NOWPAC, for both (MW) and (CFMR) problem collections.

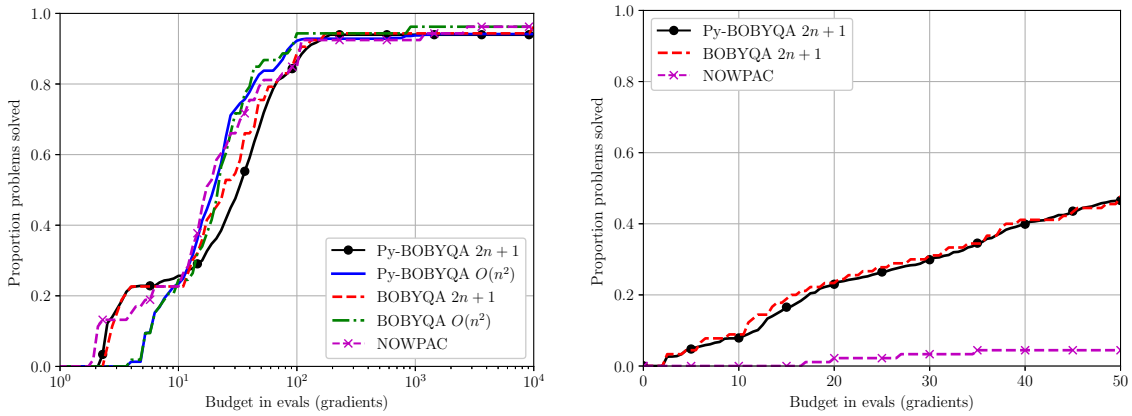
For (Py-)BOBYQA applied to (MW), we show results for the default choice $p+1 = 2n+1$, as well as the maximum value $p+1 = (n+1)(n+2)/2$, which is Py-BOBYQA's default choice for noisy problems. We do not show the $p+1 = (n+1)(n+2)/2$ results for (CFMR), because the small budget and high dimension means that almost all of the budget would be used by the initialisation phase. We see that Py-BOBYQA has comparable performance with BOBYQA and NOWPAC for smooth problems, which we expect given the similarity of the algorithms.

6.3.2 Noisy Regime

In Figure 6.2, we compare Py-BOBYQA with BOBYQA, STORM and SNOWPAC. Similar to our results for DFO-LS, we see that using multiple restarts gives a substantial improvement

[2] These results use version 1.0.1.

[3] As mentioned in Section 1.1.2, there are several variants of STORM proposed in [50]. We chose this version, based on highly overdetermined quadratic regression models, because it showed better performance than other variants.



(a) Problem collection (MW) *(b) Problem collection (CFMR)*

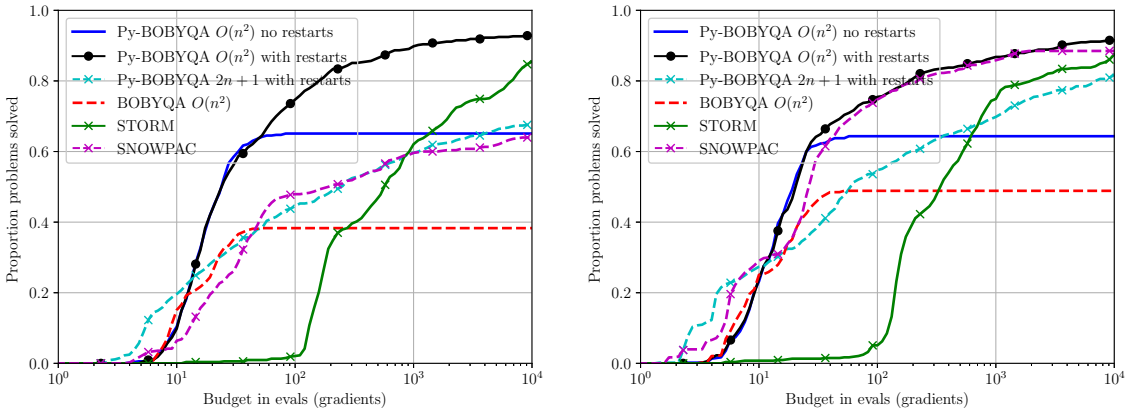
Figure 6.1. Comparison of the basic implementation of Py-BOBYQA with the original Fortran BOBYQA and NOWPAC for smooth objective evaluations and high accuracy $\tau = 10^{-5}$. For (Py-)BOBYQA, $2n+1$ and $\mathcal{O}(n^2) = (n+1)(n+2)/2$ are the number of interpolation points. For Py-BOBYQA, results are an average of 10 runs. The (MW) results have a budget of $10^4(n+1)$ evaluations and the (CFMR) results have a budget of $50(n+1)$ evaluations, and both have a 12 hour runtime limit.

in the robustness of Py-BOBYQA, and it performs substantially better than BOBYQA. We also see the benefit of using a larger number of interpolation points for Py-BOBYQA, for noisy problems; as observed in [200], choosing a smaller interpolation set is useful for small budgets, but it ultimately leads to a reasonable reduction in robustness. This validates our choice of the maximal value $p+1 = (n+1)(n+2)/2$ as the default in Py-BOBYQA for noisy problems, but $2n+1$ for smooth problems, where there is no difference in robustness (see discussion of Figure 6.1 above).

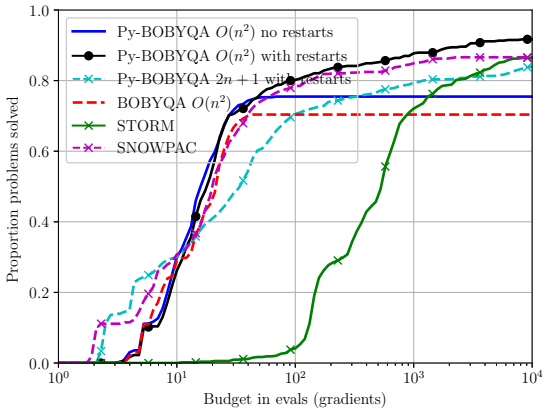
In our experiments, Py-BOBYQA can solve more problems than STORM within the computational budget, and can solve many problems much more efficiently. We note that STORM relies on constructing models that are entirely independent at each iteration, so it takes many more evaluations to begin seeing the desired objective reductions. Compared to SNOWPAC, which requires both (noisy) objective values and standard deviation, Py-BOBYQA performs either comparably or better, with the difference most noticeable for multiplicative noise. The multiple restarts approach in Py-BOBYQA has the advantage of not requiring extra user input, and being cheap to implement compared to constructing a surrogate model.

To illustrate the relative cost of multiple restarts compared to building surrogate models, Figure 6.3 shows the runtime[4] for Py-BOBYQA and SNOWPAC for two different noisy problems from (MW), using the large budget of $10^4(n+1)$ objective evaluations and additive

[4] CPU time, measured on a Lenovo ThinkCentre M900 (with one 64-bit Intel i5 processor, 8GB of RAM).



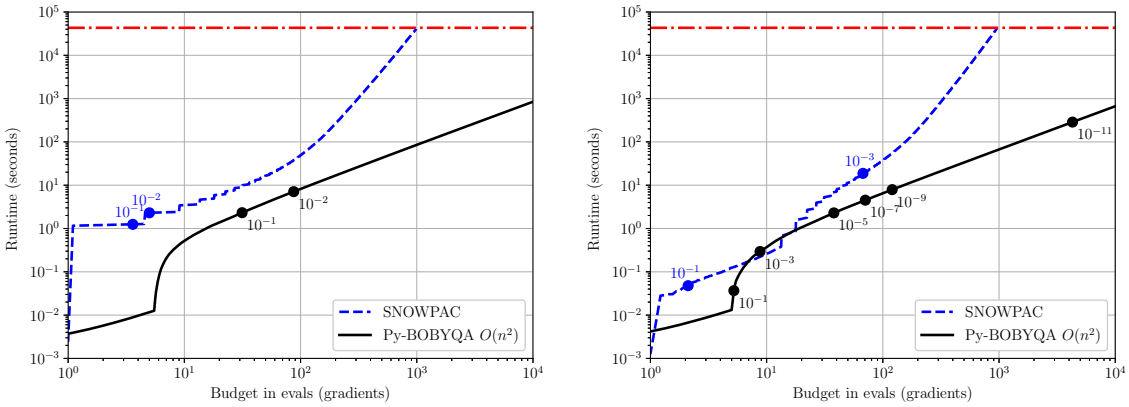
(a) Multiplicative Gaussian noise *(b) Additive Gaussian noise*



(c) Additive χ^2 noise

Figure 6.2. Comparison of the basic implementation of Py-BOBYQA with the original Fortran BOBYQA, STORM and SNOWPAC for noisy objective evaluations with $\sigma = 10^{-2}$ and high accuracy $\tau = 10^{-5}$. For (Py-)BOBYQA, $2n+1$ and $\mathcal{O}(n^2) = (n+1)(n+2)/2$ are the number of interpolation points. Results shown are an average of 10 runs for each solver. The problem collection is (MW).

Gaussian noise. For each problem, we mark when each solver achieves the particular objective reduction $\tau_{\text{crit}}(p)$ until it reaches the 12 hour timeout; see (2.69). For both problems, the runtime of Py-BOBYQA grows linearly with the number of objective evaluations, after the initial setup cost of $\mathcal{O}(n^2)$ evaluations. However SNOWPAC's runtime starts to grow much more quickly for large budgets. In SNOWPAC, the number of points used to build the surrogate model—which drives the cost of surrogate model construction—depends on the number of evaluated points in the entire run that are sufficiently close to $\mathbf{x}_k$. In many cases, this means the rapid increase in runtime occurs in the asymptotic regime, when a good solution has already been found (i.e. accuracy $\tau_{\text{crit}}(p)$ has been achieved), and SNOWPAC is trying to improve the quality of the solution using a more accurate surrogate. This occurs in problem 1, for instance, where $\tau_{\text{crit}}(p) = 10^{-2}$, and SNOWPAC achieves this accuracy well before the runtime starts to grow quickly. However, problem 53 is an example where the



(a) Problem 1, $\tau_{\text{crit}}(p) = 10^{-2}$ *(b) Problem 53, $\tau_{\text{crit}}(p) = 10^{-13}$*

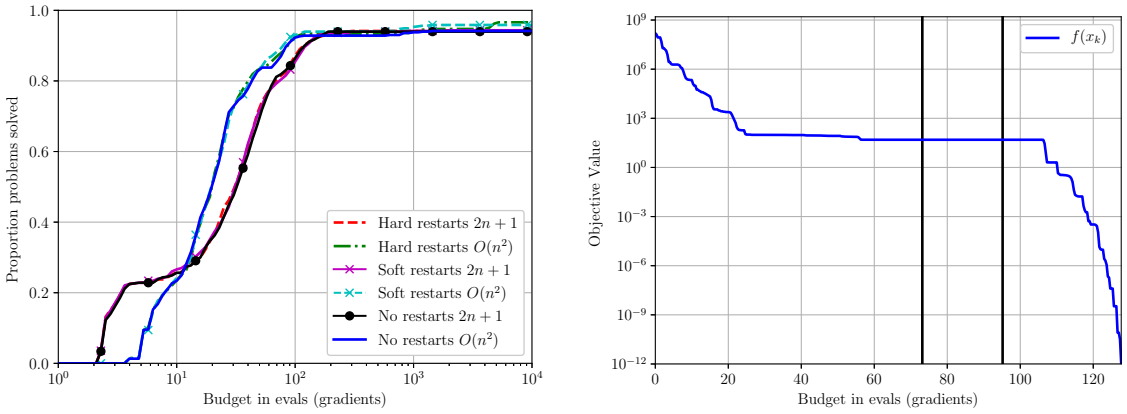
Figure 6.3. Comparison of average runtimes—up to a maximum of 12 hours (horizontal dot-dash line)—for Py-BOBYQA (with $(n+1)(n+2)/2$ interpolation points and multiple restarts) and SNOWPAC, for two problems from (MW). The marked points are average budget/runtime when each solver achieved the labelled objective reduction τ. Both problems had additive Gaussian noise with $\sigma = 10^{-2}$. Results shown are an average of 10 runs for each solver.

increase in runtime comes before this high accuracy regime: we have $\tau_{\text{crit}}(p) = 10^{-13}$, and SNOWPAC terminates (from the timeout) without achieving accuracy 10^{-4}. By comparison, on this problem, Py-BOBYQA terminates on maximum budget after reaching the much higher accuracy level $\tau = 10^{-11}$ before terminating (on budget). Overall, the use of a surrogate model is beneficial for achieving robustness to noise, but may result in reduced performance in order to realise this benefit.

6.3.3 Multiple Restarts for Noiseless Problems

We conclude by illustrating that there may also be some benefit in using multiple restarts when running Py-BOBYQA on smooth problems. As before, since the first run of Py-BOBYQA with restarts is the same as the full solver run without restarts, there is no performance loss from using multiple restarts (although more of the computational budget is used). In Figure 6.4a, we compare Py-BOBYQA without restarts against soft (moving $\mathbf{x}_k$) and hard restarts for the (MW) collection. As expected, at this accuracy level, multiple restarts either gives the same or slightly better robustness than no restarts—the improvement is larger when using $(n+1)(n+2)/2$ interpolation points.

However, one benefit of multiple restarts is being able to escape local minima. In Figure 6.4b, we show the objective value $f(\mathbf{x}_k)$ for one run of Py-BOBYQA with soft restarts and $2n+1$ interpolation points for problem 14 in (MW); the vertical lines show where restarts occurred. This problem has two local minima, with $f(\mathbf{x}^*) \approx 48.98$ and $f(\mathbf{x}^*) = 0$ [155]. We see that when the first restart occurs, we have found the local minimum with



(a) Data Profile, $\tau = 10^{-5}$, (MW) collection *(b) Objective value, problem 14 (soft restarts)*

Figure 6.4. Illustration of the impacts of multiple restarts for Py-BOBYQA on noiseless problems. For (a), we show Py-BOBYQA without restarts, and with soft (moving $\mathbf{x}_k$) and hard restarts; because the problem has no noise, we do not use autodetection of restarts. Figure (b) shows the objective value $f(\mathbf{x}_k)$ using Py-BOBYQA with soft restarts and $2n+1$ interpolation points, for (MW) problem 14; the vertical lines indicate where restarts occurred.

higher objective value—this is when Py-BOBYQA would usually terminate. However, if we allow Py-BOBYQA to perform two soft restarts, it manages to find the other local minimum (which is also the global minimum).

6.4 Py-BOBYQA for Escaping Local Minima

The results in Section 6.3.3 provide some preliminary evidence that the multiple restarts feature can help Py-BOBYQA to escape local minima. Motivated by this, in this section we further modify the restart procedure in a way that is (even more) advantageous to global optimisation, and test both the usual Py-BOBYQA (with restarts) and its modification against state-of-the-art global optimisation software—on standard global optimisation test problems and on a parameter tuning problem from machine learning—with encouraging results, as we describe below.

Adaptive restarts The new restart set up in Py-BOBYQA, to help achieve (faster) escape from local minima proceeds as follows. Instead of setting $\Delta_{k+1}^{\text{init}} = \rho_{k+1}^{\text{init}} = \Delta_0^{\text{init}}$ every time a restart is triggered (which in this section we refer to as a fixed restart), we modify Py-BOBYQA to allow $\Delta_{k+1}^{\text{init}} = \rho_{k+1}^{\text{init}} = \Delta_{\text{reset}}$, where Δ_{reset} is increased by a constant factor (default is 1.1) if the previous restart failed to produce a reduction in the objective. Thus Δ_{reset} is still set to Δ_0^{init} for the first restart, but will be progressively increased on restarts that do not make progress. The motivation for this approach is that the restart helps Py-BOBYQA to build models in a larger region of the domain. For global optimisation

problems, if we are not making progress after a given restart, we should gradually expand how much of the domain we explore. To use the language of global optimisation (see Section 6.4.1 below), we may view one run of Py-BOBYQA (until a restart is called) as an 'exploitation' step, and the adaptive restart mechanism as a proxy for an 'exploration' step.

An illustrative example We use the well-known Ackley function, that has a global minimum surrounded by a high number of local minima, and illustrate the impact of using fixed and adaptive restarts in Py-BOBYQA on finding the global minimum. In Figure 6.5, Py-BOBYQA with $2n + 1$ interpolation points is applied to the two-dimensional Ackley function with bound constraints $\mathbf{x} \in [-26, 26]^2$ and no noise, from starting point $\mathbf{x}_0 = [20,\ 20]^\top$ and with an evaluation budget of 10^3 gradients. Both usual (fixed) restarts and adaptive restarts are tested. Figure 6.5 shows that both fixed and adaptive restarts help Py-BOBYQA reach the global minimum to high accuracy (while the no restarts Py-BOBYQA does not). Furthermore, by comparison to fixed restarts, adaptive restarts both achieve a larger reduction in the cost function within fewer evaluations and require fewer restarts to achieve this improvement.

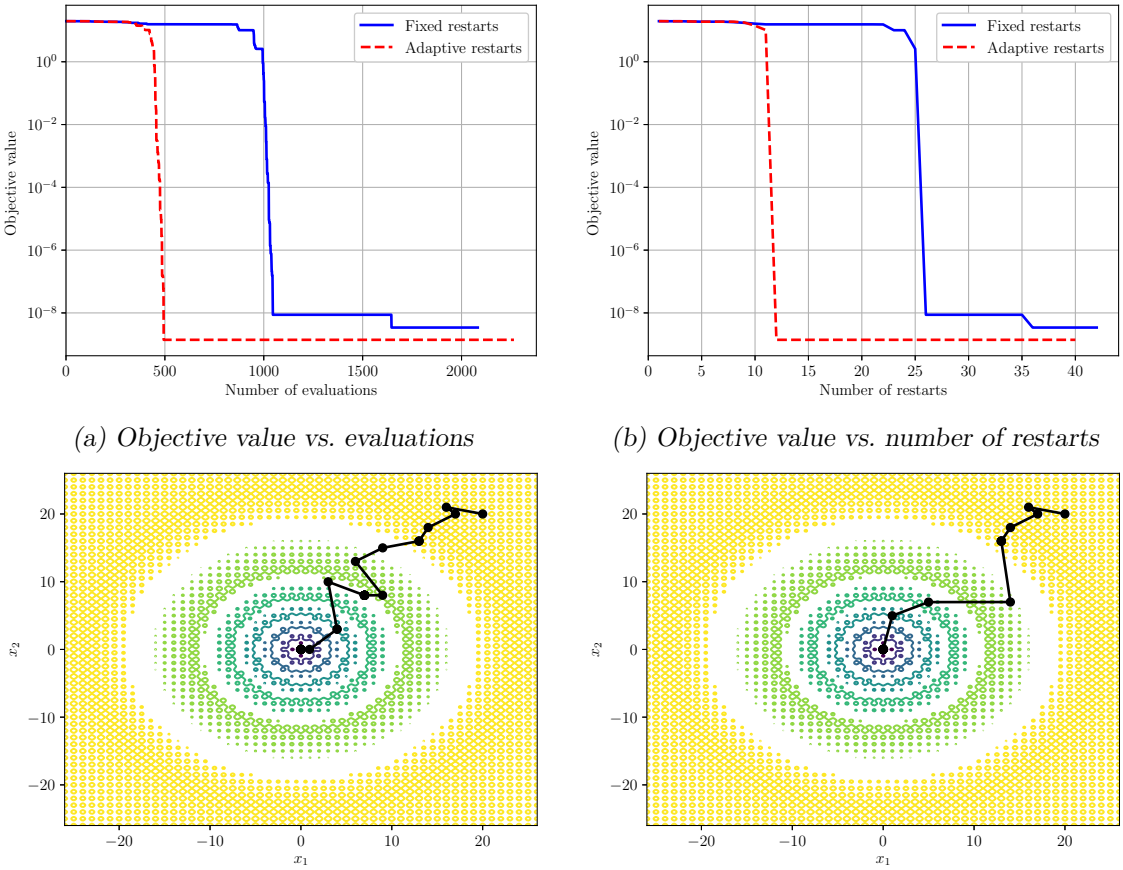
6.4.1 Brief Survey of DFO Methods for Global Optimisation

Due to the numerical comparisons to follow, we now briefly review different DFO methods designed specifically for global optimisation; more details can be found in [140, 199], for instance. Since this now requires the solver to fully explore the feasible region (which we assume to be bounded, usually defined by box constraints (2.5)), the techniques here are quite different to those for local optimisation. In particular, they need to balance 'exploitation' (searching in regions where the objective is known to be small) and 'exploration' (searching in unexplored regions).

Bayesian/Surrogate Optimisation Bayesian and surrogate optimisation methods are designed for the framework where the objective is expensive to evaluate, and so a large computational effort should go into selecting the next evaluation point in an informed way. Both types of method build global models for the objective (e.g. via Gaussian Processes/radial basis functions [192]) based on all available evaluations.[5] The next evaluation point is selected by minimising a particular function (based on the global model) aiming to balance exploitation and exploration.[6] This requires a call to another global optimisation algorithm,

[5] It is usual to begin by sampling the objective at a number of points distributed throughout the feasible region, called an 'initial design'.

[6] We note that simply minimising the current global model over-emphasises exploitation, and can lead to only finding a local minimum [112].



(a) Objective value vs. evaluations

(b) Objective value vs. number of restarts

(c) Path of iterates after each restart (fixed)

(d) Path of iterates after each restart (adaptive)

Figure 6.5. Demonstration of Py-BOBYQA with multiple restarts on the Ackley function. This function has a local minimum at each integer-valued coordinate, and a global minimum of 0 at the origin.

however this is easier than solving the original problem as this new function is cheap to evaluate. More details on these methods and their applications, particularly in machine learning, can be found in [194, 35, 82, 205], for instance.

Dividing Rectangles The DIRECT (*di*viding *rect*angles) method [113] is an extension of Lipschitz global optimisation methods, which minimise a piecewise-linear lower bound for the objective, constructed using its (known) Lipschitz constant.[7] DIRECT extends this to unknown Lipschitz constants. It partitions the feasible region into boxes, and the objective is evaluated at the centre of each. At each iteration, boxes with either small objective value (exploitation) and/or large width (exploration) are subdivided.[8] This process is generalised

[7] This is the Lipschitz constant of f, not ∇f (which is an important quantity for nonconvex optimisation theory; e.g. Assumption 2.1).

[8] This is done by estimating linear lower bounds for the objective, hence the relationship with Lipschitz global optimisation.

in multilevel coordinate search [108], where the evaluated point is not necessarily the centre of each box.

Evolutionary Algorithms In evolutionary, or genetic, algorithms, we maintain a set of points, and at each iteration they combine these to generate a new set, and select the best points from both.[9] In the context of global optimisation, the most well-known method is CMA-ES (Covariance Matrix Adaptation Evolutionary Strategy) [101, 99]. Here, the set of points is modelled as samples from a multivariate normal distribution, and the new set consists random samples from this distribution. We select the points with the lowest objective value, and update the mean and covariance of the distribution based on these.

Branch and Fit In SNOBFIT [109], we subdivide the domain, and construct local (linear or quadratic) interpolation models in each region based on nearby evaluated points. We then minimise each local model to determine a set of candidate points for evaluation, and ensure exploration by also evaluating points in regions far from existing points.

Multi-Start Local Optimisation An alternative heuristic approach for global optimisation is to perform several runs of a local optimisation method, each from a different starting point [140, 103]. The hope is that one of the starting points lies in the basin of attraction for the global minimum. This is different to our multiple restarts approach in Py-BOBYQA, which only ever uses a single initialisation point for a run, and employs the final iterate of a run as the starting point for the restarted run.

6.4.2 Testing Setup

For our comparison, we use the (AKZ) collection of test problems (see Section 2.4.1), and consider problems without noise, and multiplicative and additive Gaussian noise with $\sigma = 10^{-2}$ (as defined in Section 2.4.1 for general-objective problems).

All solvers are allowed a budget of at most $10^4(n+1)$ objective evaluations for an n-dimensional problem, and a runtime of at most 12 hours per problem instance. In addition, Py-BOBYQA, CMA-ES, PySMAC and SNOBFIT require a starting point; for each solver, problem, and instance, we choose this randomly from a uniform distribution over the entire feasible region (independently for each solver/problem/instance). Given this, we run 10 instances of each solver on each problem (where, each instance has different realisations of the stochastic noise in the objective, and, if applicable, a different starting point).

[9] In analogy to natural evolution, we call the first set a 'population', the new set 'offspring'—generated by 'mutations'—and select the best points according to a 'fitness function'.

Solvers tested We compare Py-BOBYQA with a selection of global optimisation routines, all implemented or wrapped in Python.[10] Here, we list each solver tested and any non-standard parameters set—in some cases, these choices depended on whether we were testing an objective with or without noise:

CMA-ES v2.5.7 [101, 100] (Evolutionary): Using an initial standard deviation $0.2 \min_i(b_i - a_i)$, where the feasible region is $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$;

DIRECT v1.0.1 [113, 110]: Using all defaults, except for increasing the maximum number of objective evaluations to be sufficiently large for our testing;

GPyOpt v1.2.1 [213] (Bayesian): Using an initial design of $2n$ random points, and the `exact_feval` flag is set if the objective is noisy;

HyperOpt v0.1 [28, 29] (Bayesian): Using a uniform search space over the feasible region, Tree of Parzen Estimators algorithm, with and without providing noise variance for the objective evaluation;

PySMAC v0.9.1 [107, 19] (Bayesian): We set the flag `deterministic` according to whether the objective had noise or not;

PySOT v0.1.36 [76] (Surrogate): Using a radial basis function surrogate with an initial design of $2n + 1$ Latin Hypercube points and a DYCORS [195] strategy with $100n$ points for selecting new points;

SNOBFIT v1.0.0 [109, 238] (Branch & Fit): Using all default parameters.

For Py-BOBYQA, we used version 1.1, with options: scale feasible region to $[0, 1]^n$; $\Delta_0^{\text{init}} = 0.1$ and $\rho_{\text{end}} = 10^{-8}$ (both in scaled variables); on unsuccessful restarts, increase trust region radius Δ_{reset} by a factor[11] of 1.1; and terminate after 10 consecutive unsuccessful restarts or 20 total unsuccessful restarts.

This collection of solvers has a large focus on Bayesian (GPyOpt, HyperOpt, PySMAC) and surrogate (PySOT) methods, since, like Py-BOBYQA, they rely on minimising models for the objective.

[10] We use Python 2.7 with NumPy 1.12.1 and SciPy 1.0.1, and Py-BOBYQA version 1.1.

[11] The constant factor 1.1 was not decided based on testing or tuning; it was simply considered to be a sensible choice.

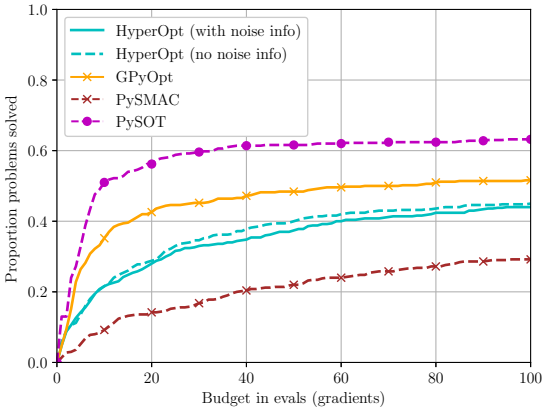
6.4.3 Numerical Results

Selecting the best-performing Bayesian and surrogate solver We begin by comparing only the Bayesian solvers (GPyOpt, HyperOpt and PySMAC) and surrogate algorithm PySOT. Figure 6.6 shows that, for both smooth and noisy problems, PySOT is consistently the best performing solver compared to all Bayesian solvers that we tested, in both low and high accuracy regimes; the next best solver is GPyOpt for low accuracy regime, but when large budgets are available, HyperOpt and PySMAC improve their performance (compared to GPyOpt). Thus in our comparisons between all solvers, we select only PySOT to 'represent' the best behaviour of Bayesian and surrogate solvers.

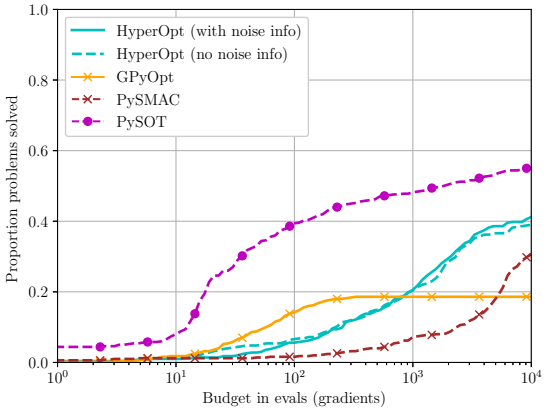
Selecting the best-performing Py-BOBYQA variant First, we compare the different restart mechanisms for Py-BOBYQA using data profiles. We consider Py-BOBYQA with both $2n+1$ and $(n+1)(n+2)/2$ interpolation points (the default choices for smooth and noisy problems respectively), compare no restarts, and soft and hard restarts, and look at both fixed and adaptive variants of each restart mechanism.

For both interpolation models, and different noise models (including no noise), Figure 6.7 shows that, with restarts, Py-BOBYQA is able to use the full budget to continue making progress, while the no-restarts variant can no longer improve its performance beyond the achievements in the low accuracy regime, despite the substantially larger budget. In particular, Py-BOBYQA without restarts is the worst-performing variant, failing to solve the largest number of problems within the given (large) budget. In terms of restarts, the soft restart variant with adaptive restart radius (adaptive Δ_{reset}) is the most efficient and economical across all models and restart types, followed by standard soft restarts (with fixed radius size) and matched sometimes by hard restarts with adaptive radius size (such as for $p+1 = 2n+1$ and noisy problems; and $p+1 = (n+1)(n+2)/2$ and no noise).

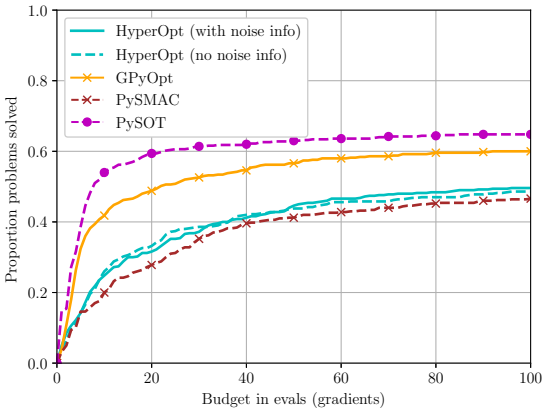
Similar to Section 6.3, Py-BOBYQA with $\mathcal{O}(n^2)$ interpolation points, soft restarts and adaptive radius on restarts gives the best performance overall, and is particularly effective on noisy problems. For smooth problems, we get similar results with $2n+1$ as with $\mathcal{O}(n^2)$ interpolation points, but with the benefit of reduced initialisation and linear algebra cost. However, in this accuracy/budget regime, the benefit of the reduced interpolation cost is not seen.



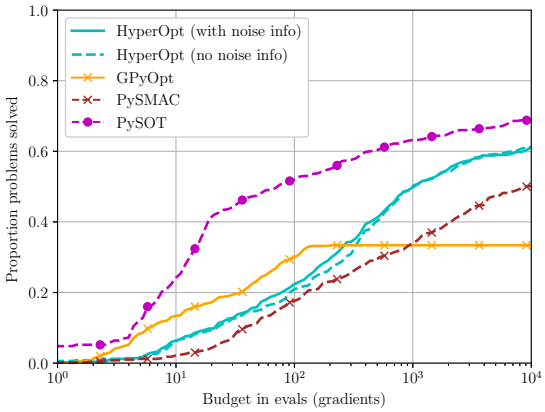
(a) Low accuracy ($\tau = 10^{-2}$), smooth.



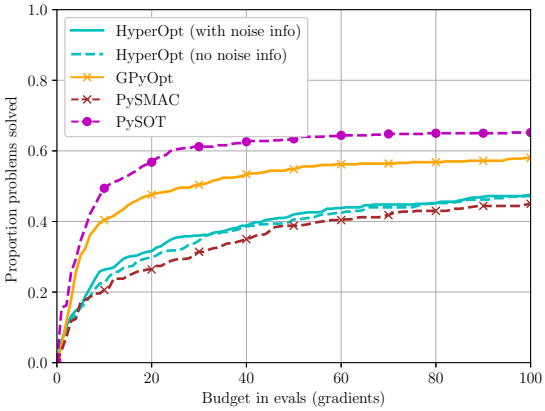
(b) High accuracy ($\tau = 10^{-5}$), smooth.



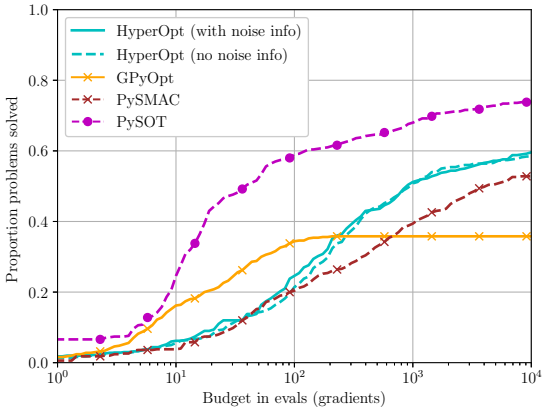
(c) Low accuracy ($\tau = 10^{-2}$), multiplicative noise.



(d) High accuracy ($\tau = 10^{-5}$), multiplicative noise.

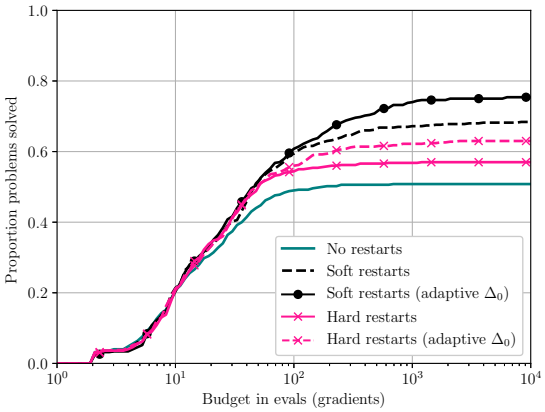


(e) Low accuracy ($\tau = 10^{-2}$), additive noise.

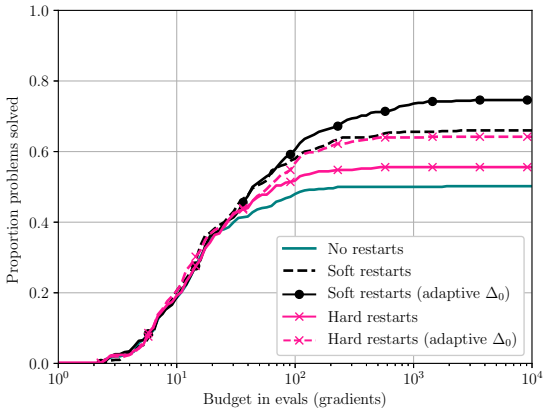


(f) High accuracy ($\tau = 10^{-5}$), additive noise.

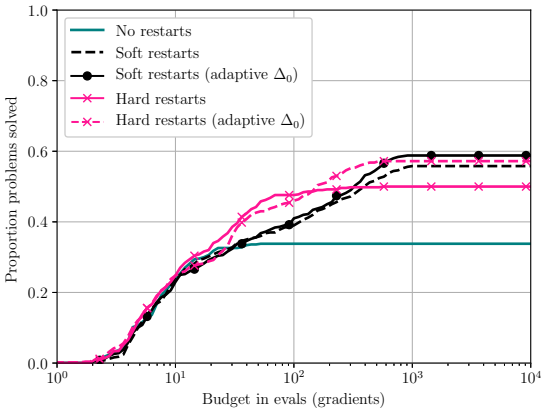
Figure 6.6. Comparison of Bayesian and surrogate solvers on the global optimisation test set (AKZ), with different choices for the interpolation set size and noise model. These results use a budget of $10^4(n+1)$ evaluations and a 12 hour runtime limit.



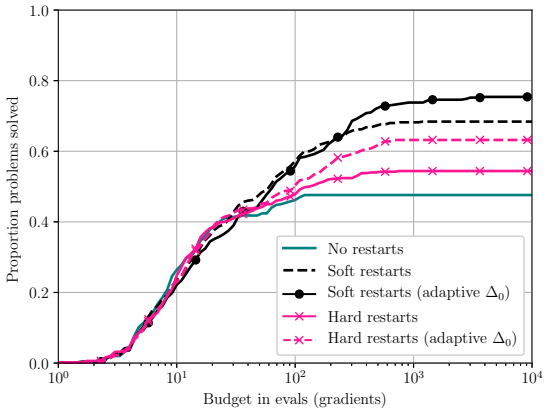
(a) Smooth, $2n + 1$ points.



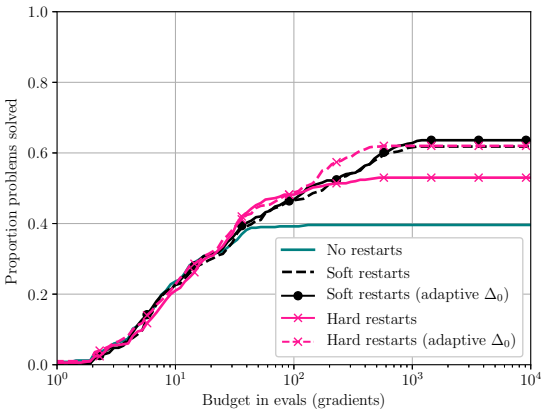
(b) Smooth, $\mathcal{O}(n^2)$ points.



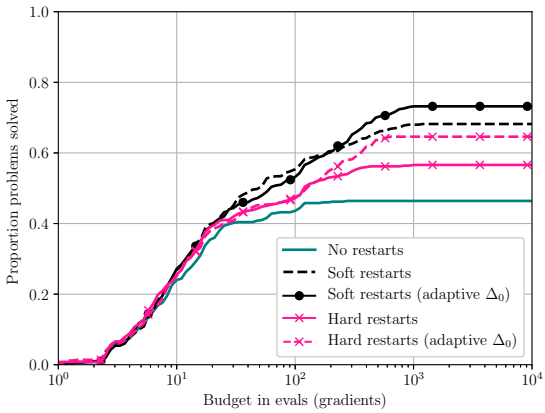
(c) Multiplicative noise, $2n + 1$ points.



(d) Multiplicative noise, $\mathcal{O}(n^2)$ points.

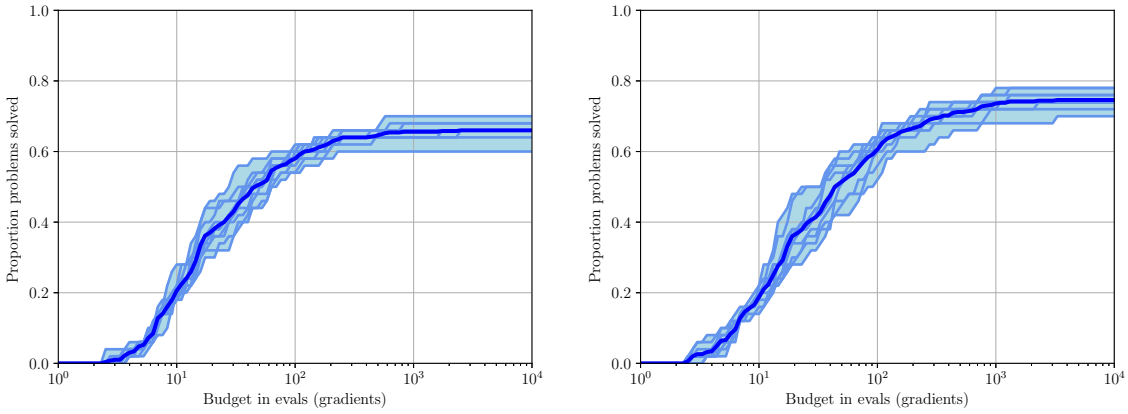


(e) Additive noise, $2n + 1$ points.



(f) Additive noise, $\mathcal{O}(n^2)$ points.

Figure 6.7. Comparison of Py-BOBYQA restart variants on the global optimisation test set (AKZ), with different choices for the interpolation set size and noise model. These results use a budget of $10^4(n + 1)$ evaluations and a 12 hour runtime limit, and have accuracy level $\tau = 10^{-5}$.



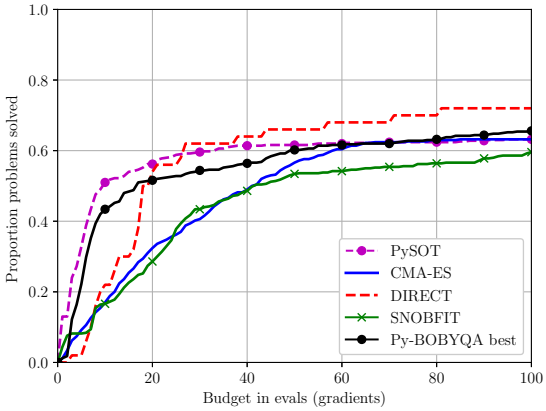
(a) Fixed restart radius. *(b) Adaptive restart radius.*

Figure 6.8. Demonstration of impact of random starting point for Py-BOBYQA ($p+1 = \mathcal{O}(n^2)$ with soft restarts). We use accuracy level $\tau = 10^{-5}$ and a budget of $10^4(n+1)$ evaluations. The dark lines are the average of all runs. The problem set is (AKZ).

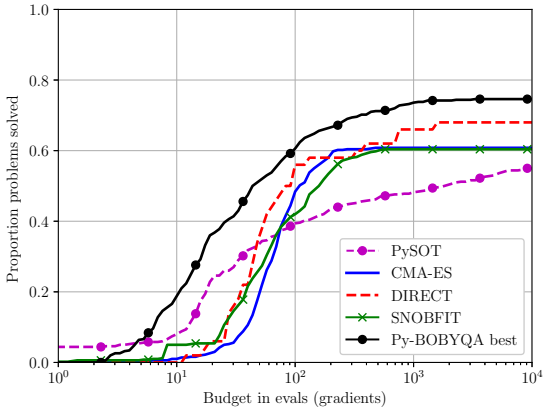
Py-BOBYQA variance test As Py-BOBYQA is a local solver, and as we are using randomly chosen starting points per problem instance, we investigate here the variation in performance over the 10 different runs of Py-BOBYQA over the test set with different starting points each time. This illustrates the variance in our previous results, and how much the performance is influenced by a particular starting point, a key question for local solvers. Given the good performance of Py-BOBYQA with full quadratic models ($p+1 = \mathcal{O}(n^2)$) and soft restarts across all test problem variants, we focus on this variant, with fixed and adaptive restart trust-region radius on restarts. We are only looking at smooth problems (with no noise), to make sure the only randomness present is due to choice of starting point. In Figure 6.8, we show separate data profiles for the ten runs of Py-BOBYQA (i.e. each profile shows one instance of each problem with a random starting point), and the average data profile.

The variation in starting point usually leads to between 5%–10% variation in performance for all budgets and both restart mechanisms, except in the low budget and adaptive trust-region radius regime where we see a variation in performance of about 20%. If we compare with Figure 6.7, we see that almost all adaptive radius runs are perform better than the fixed radius variant.

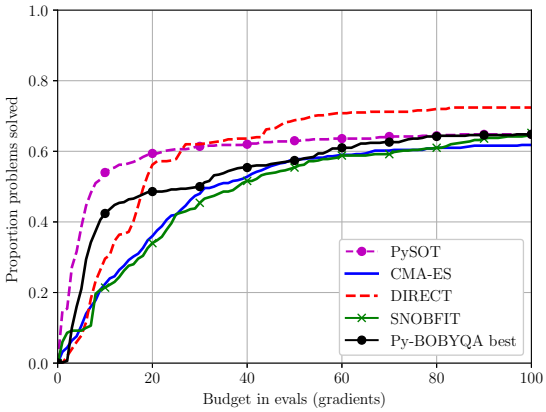
We also note that these conclusions are similar for Py-BOBYQA with $2n+1$ interpolation points; see [49] for these results.



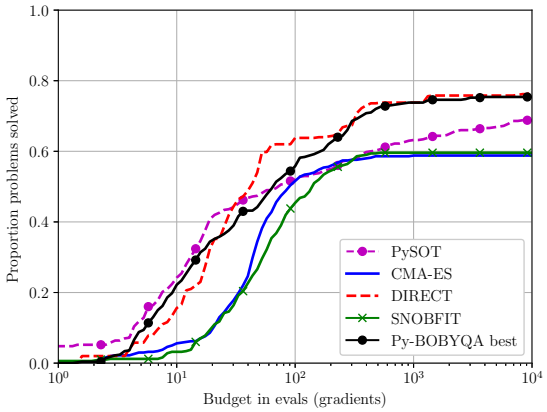
(a) Low accuracy ($\tau = 10^{-2}$), smooth.



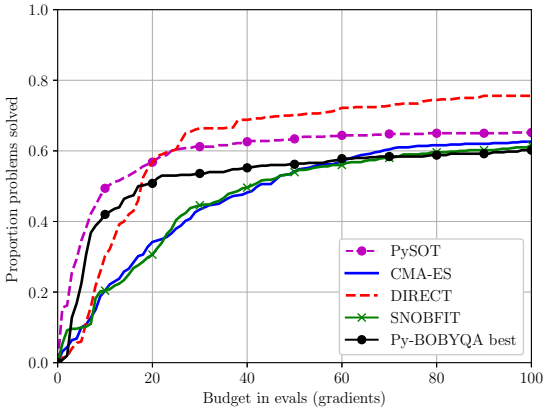
(b) High accuracy ($\tau = 10^{-5}$), smooth.



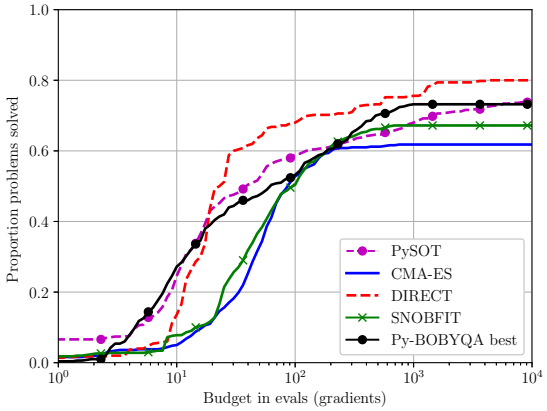
(c) Low accuracy ($\tau = 10^{-2}$), multiplicative noise.



(d) High accuracy ($\tau = 10^{-5}$), multiplicative noise.



(e) Low accuracy ($\tau = 10^{-2}$), additive noise.



(f) High accuracy ($\tau = 10^{-5}$), additive noise.

Figure 6.9. Comparison of all solvers on the global optimisation test set (AKZ) for different accuracy/budget choices and noise models. These results use a budget of $100(n+1)$ or $10^4(n+1)$ evaluations and a 12 hour runtime limit.

Comparisons of all solvers In Figure 6.9, we compare Py-BOBYQA against the global solvers. The solver "Py-BOBYQA best" represents the variant with fully quadratic interpolation ($p = \mathcal{O}(n^2)$), soft restarts and adaptive trust region radius on restarts, which we found above to be the best variant for global optimisation (see the discussion around Figure 6.7).

Low accuracy/budget ($\tau = 10^{-2}$ and budget 10^2 gradients) The first column in Figure 6.9 shows that on average, DIRECT performs best in this regime, and the remaining solvers are essentially similar at the end of the budget. In the earlier phases of budget, PySOT and then the best Py-BOBYQA variant have best performance, being able to solve more problems on small budgets than the remaining solvers, while CMA-ES and SNOBFIT seem to need longer budgets to reach a similar amount of solved problems at the end of the low accuracy budget.

High accuracy/budget ($\tau = 10^{-5}$ and budget 10^4 gradients) For smooth problems, the best Py-BOBYQA variant is the best-performing solver followed by DIRECT, while for multiplicative noise, these two solvers have comparably good performance. For additive noise, DIRECT has superior performance, followed by similarly good performance from PySOT and Py-BOBYQA.

6.4.4 Case Study: Hyperparameter Tuning (MNIST)

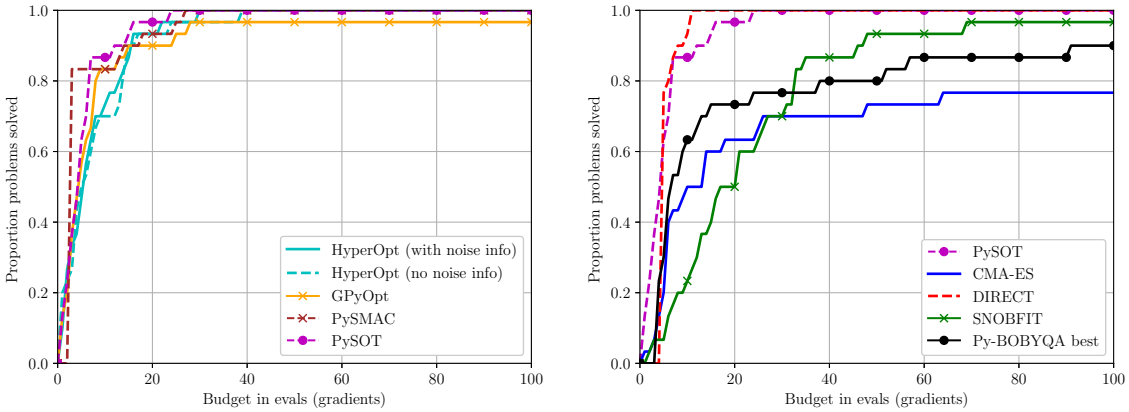
We now consider a case study of hyperparameter tuning for machine learning, for which developers have often used Bayesian optimisation solvers. Here, we follow the approach from Snoek, Larochelle, and Adams [207], and consider the problem of training a multi-class logistic classifier for MNIST[12] using stochastic gradient descent with a fixed learning rate. The 4 hyperparameters we optimise in our testing are:

- $\log_{10}$ of learning rate, in $[-10, 0]$;
- $\log_{10}$ of ℓ_2 regularisation parameter, in $[-10, 0]$;
- Number of epochs, in $[1, 100]$; and
- Batch size, in $[1, 2000]$.

We convert real-valued inputs for the last two parameters to integer values by rounding up to the nearest integer. The objective function we select is the (top-1) error rate on the test set.[13] We run the solvers for a maximum of $100(n + 1) = 500$ objective evaluations or 36

[12] The multi-class classifier came from by training 10 one-versus-all classifiers, with prediction by selecting the class with highest probability. The training set had 60,000 images, the test set had 10,000 images.

[13] That is, the proportion of problems where the true class did not match the class with the highest probability from the 10 trained one-versus-all classifiers.



(a) Bayesian and surrogate solvers. *(b) Main solver comparison.*

Figure 6.10. Comparison of solvers on the MNIST problem, for low accuracy ($\tau = 10^{-2}$). These results use a budget of $100(n+1)$ evaluations and a 36 hour runtime limit.

hours. For each solver, we run 30 instances of the problem, and select starting points (if needed) in the same way as above (i.e. uniformly at random).

Selecting the 'optimum' value for MNIST To construct data profiles for noisy problems, we need estimates of $\mathbb{E}[\widetilde{f}(\mathbf{x}_0)]$, $\mathbb{E}[\widetilde{f}(\mathbf{x}^*)]$ and $\sigma(\mathbf{x}^*)$ in order to evaluate (2.69) in Section 2.4.2. However, unlike before, we do not have access to the true values for our objective, and so we need to provide sensible estimates of these values. For $\mathbb{E}[\widetilde{f}(\mathbf{x}_0)]$, we use the value of $\widetilde{f}(\mathbf{x}_0)$ from each instance of each solver. To determine $\mathbf{x}^*$, and hence $\mathbb{E}[\widetilde{f}(\mathbf{x}^*)]$ and $\sigma(\mathbf{x}^*)$, we do the following:

1. Take the 20 points found by all instances of all solvers with the smallest (noisy) objective value;
2. Evaluate the (noisy) objective $\widetilde{f}$ independently 30 times at each of these points $\mathbf{x}$, and use this to estimate $\mathbb{E}[\widetilde{f}(\mathbf{x})]$ and $\sigma(\mathbf{x})$;
3. Select as $\mathbf{x}^*$ the point that gives the smallest estimate value of $\mathbb{E}[\widetilde{f}(\mathbf{x})]$.

This process yields $\mathbf{x}^* = [-5.2706, -2.2505, 100, 411.02]^\top$, with estimates $\mathbb{E}[\widetilde{f}(\mathbf{x}^*)] = 0.0818067$ and $\sigma(\mathbf{x}^*) = 1.01388 \times 10^{-3}$. This equates to a learning rate of 5.36×10^{-6}, ℓ_2 regularisation parameter 5.62×10^{-3}, 100 epochs and batch size 412, and corresponds to a test set error rate of $8.1\% \pm 0.1\%$.

Numerical Results Since this is a problem where Bayesian optimisation is frequently used, we first compare just the Bayesian and surrogate solvers in Figure 6.10a; these results are similar to those for the main test problem collection for this accuracy level ($\tau = 10^{-2}$),

namely, the solvers are comparable with PySOT being slightly better. When comparing the remaining solvers with the best Py-BOBYQA and PySOT in Figure 6.10b, we find that DIRECT and PySOT are best performing, and Py-BOBYQA outperforms CMA-ES, and SNOBFIT for small budgets.

Chapter 7

Scalability of Model-Based DFO for Nonlinear Least-Squares

In this final chapter, we consider how to improve the scalability of model-based DFO methods for nonlinear least-squares problems. DFO-LS is designed for small- or medium-scale problems, but the linear algebra cost of each iteration, largely due to the cost of constructing the interpolation model, means that its runtime increases rapidly for large problems. There are several settings where scalable DFO algorithms may be useful, such as data assimilation (e.g. for weather forecasting) [27, 6], machine learning [84], generating adversarial examples for deep neural networks [3, 210], and possibly as a proxy for global optimisation methods.

To address this, in this chapter we introduce a scalable model-based DFO algorithm for nonlinear least-squares problems. In essence, at each iteration we select a low-dimensional subspace of $\mathbb{R}^n$, build and minimise a model to compute a step in this space, then change the subspace at the next iteration. Our approach of building a low-dimensional model from fewer interpolation points is similar to the reduced initialisation cost mechanism in DFO-LS (Section 3.2.1), except we do not perturb our model to expand the search space across iterations. This allows us to control the linear algebra cost. Instead, we maintain a constant subspace dimension across iterations, and sample new points to explore new subspaces between iterations.

We note here the works mentioned in Section 1.1.2 related to exploiting problem sparsity, which provide an alternative route for improving the scalability of model-based DFO. Specifically, [52] exploits the partial separability (1.14) of the objective to build models for the functions f_i, which only depend on a subset of coordinates. This requires the user to provide the solver with the evaluation of each f_i separately, as well as the partial separability pattern (i.e. specifying which variables each f_i depends on). In effect, this relies on knowing substantial problem structure information in advance, which may be unavailable

in a DFO context. On the other hand, [21] uses randomised techniques to build models with sparse Hessians, for general objective problems. This can approximate objectives with sparse Hessians with relatively few interpolation points—without knowing the sparsity structure—making it a viable approach for improving scalability. Here, our goal is to have a flexible framework for nonlinear least-squares problems that can be extended to general objective problems, and does not impose extra requirements on the problem such as partial separability or Hessian sparsity.

In this chapter, we begin by reviewing existing subspace methods (Section 7.1). Then, we illustrate numerically that model construction is the key issue limiting the scalability of DFO-LS (Section 7.2). To address this issue, in Section 7.3 we describe our proposed algorithm, which we call DFBGN (Derivative-Free Block Gauss-Newton). We compare the features of DFBGN with those of DFO-LS in Section 7.4, and present our main numerical results in Section 7.5.

7.1 Existing Work

We begin by reviewing existing work on subspace optimisation methods, where iterates are generated in reduced subspaces (rather than in the full ambient space). We note that there is a large body of work on randomised search techniques (see surveys [140, Chapter 3] and [129], for instance), but here—in line with our algorithmic approach—we consider methods that handle scalability by coupling (possibly random) subspace projection with rigorous model-based optimisation methods.

Block Coordinate Descent Largely motivated by machine learning problems, block coordinate descent (BCD) methods are those where only subsets of variables are perturbed in each iteration. They may also be viewed as (derivative-based) extensions of coordinate search methods (see Section 1.1.1, or [231] for a more detailed survey). BCD methods for nonconvex optimisation target problems with a block structure:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(s)}), \tag{7.1}$$

where $\mathbf{x}$ is partitioned into blocks of variables $\mathbf{x} = [\mathbf{x}^{(1)} \ \cdots \ \mathbf{x}^{(s)}]^\top$. In BCD methods, at each iteration we select a block $\mathbf{x}^{(i)}$ and optimise over these variables.

Different BCD methods vary in their rule for selecting a block at each iteration (e.g. cyclic [233] or randomly [80]), and how to optimise over the chosen block. A popular choice for single-block optimisation is proximal gradient descent, where we update $\mathbf{x}^{(i)}$ by linearising f around $\mathbf{x}^{(i)}$, and minimising a model that includes a quadratic regularisation term [235]. This leads to a steepest descent-style iteration for smooth problems, or a proximal gradient

iteration for nonsmooth problems (e.g. when f has a nonsmooth regulariser for each block of variables, a common assumption for BCD methods).

Global convergence results for (7.1) with f nonconvex and proximal gradient descent were shown in [235] for 'essentially cyclic' block selection (i.e. each block is chosen at least once every $N \geq s$ iterations) and for a nonmonotone variant with randomised block selection in [141]. This approach was extended to inexact gradients (specifically stochastic gradients where f is a sum of functions) in [234], and to more general iterations (beyond proximal gradient descent) in [237]. A worst-case complexity analysis in the case of f convex and possibly nonsmooth is given in [197]. An earlier convergence analysis was given in [218] for (7.1) with essentially cyclic block selection, but solving the block subproblem to full optimality (rather than one iteration of proximal gradient descent, for instance).

Another work with some similarity to coordinate descent methods is [215], which develops a model-based DFO method for bound-constrained problems. In each iteration, the constraints are handled by fixing the variables with active bounds, then solving the resulting lower-dimensional problem—optimising only the variables with inactive bounds—via a recursive call to the same algorithm.

Block Coordinate Gauss-Newton BCD methods have been specialised to nonlinear least-squares problems. In this setting, we select a block of variables and construct the Jacobian associated with those variables (i.e. selecting columns of the full Jacobian) to build a Gauss-Newton model for the given low-dimensional block. This was initially proposed in the context of parameter fitting for climate models [212], where an implicit filtering approach was used to handle noise in function evaluations. In a derivative-based setting, quadratic regularisation/proximal and trust-region variants of this method are analysed theoretically and tested in [40], where an adaptive mechanism for selecting how many blocks/variables to minimise in each iteration is also proposed.

Randomised Direct Search There has been some work in direct search DFO (see Section 1.1.1) on pattern search methods where the poll set $\mathcal{D}_k$ is replaced by a random set of directions, which does not need to be a positive spanning set (and in particular may only search in a randomly-generated subspace) at any iteration [92, 94]. The works [92, 94] provide convergence and worst-case complexity results, for both unconstrained and linearly-constrained problems.

Random Projections An alternative approach for high-dimensional DFO methods is to, at each iteration, project the problem into a random low-dimensional subspace, then solve this low-dimensional problem with standard methods. This approach has been considered in

[190] for problems with low effective dimension (i.e. there exists a low-dimensional subspace $V \subset \mathbb{R}^n$ such that $|f(\mathbf{x}) - f(\text{proj}_V \mathbf{x})| \leq \epsilon$ for all $\mathbf{x}$), and in [222] for convex functions depending only on a small number of input variables (i.e. $\nabla f(\mathbf{x})$ is sparse), but where the objective evaluation is noisy.

Sketching In the context of nonlinear least-squares problems (and related problems such as sum of functions), scalability issues may also arise from having many residuals (not necessarily having large numbers of variables). Here, sketching methods are a common technique, which involve approximating the sum over r_i in (2.6) with a random subsample of terms, or random linear combinations of terms. Different sketching methods have been analysed for Newton's method [170, 201], quasi-Newton methods [88], and linear systems [89], for instance.

7.2 Factors Limiting Scalability

We first show that the linear algebra costs associated with model construction are a key performance bottleneck of our model-based DFO methods for large problems. This motivates our use of subspace methods for improving their scalability, as they are designed to reduce the linear algebra cost in each iteration.

To this end, we run DFO-LS on the generalised Rosenbrock function

$$f(\mathbf{x}) := \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2, \tag{7.2}$$

which we formulate as a least-squares problem (2.6) with $m = 2(n-1)$ residuals. We start from $\mathbf{x}_0 = [-1.2, 1, -1.2, \ldots, 1]^\top \in \mathbb{R}^n$, and where the underlying dimension n is variable. We allow DFO-LS to run until termination or $3(n+1)$ evaluations, with all default settings, for dimensions $n \in [100, 1000]$. Running on a Lenovo ThinkCentre M900 (with one 64-bit Intel i5 processor, 8GB of RAM), we measure the total runtime that DFO-LS spends in different parts of the algorithm.[1]

In Figure 7.1, we show the total runtime split into the time taken for interpolation model construction, Lagrange polynomial construction, and all other parts of the algorithm (e.g. trust-region subproblem). We see that the runtime increases rapidly as the underlying dimension n increases, and that this increase is largely driven by model and Lagrange polynomial construction. This suggests to us that we should focus our attention on methods that have a low cost for solving the relevant interpolation linear system (i.e. (3.6)), even as

[1] We achieve this using Python's built-in code profiler, cProfile. Since (7.2) is simple to implement, the cost of objective evaluation here is negligible.

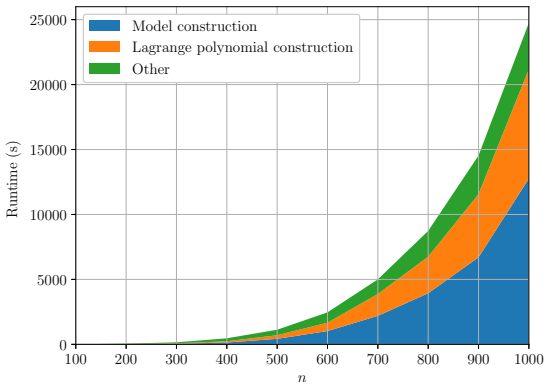


Figure 7.1. Runtime breakdown for DFO-LS for minimising the generalised Rosenbrock function, as problem dimension n increases, with a budget of $3(n+1)$ evaluations.

the underlying dimension n grows. Our approach, where we construct models in a reduced subspace, will give Lagrange polynomials that are effectively low-dimensional. Hence, our approach will reduce the cost of both interpolation and Lagrange polynomial construction.

We note that our approach will not reduce the cost in terms of the number of residuals m. Applying sketching methods to DFO-LS is delegated to future work (see Chapter 8).

Linear Algebra Costs Currently, the interpolation linear systems are solved using the approach described in Section 3.3.4, namely: factorise the interpolation matrix, then back-solve for each right-hand side. Thus, if we have an interpolation set of size $n+1$—the usual case for DFO-LS—the cost of the linear algebra is:

1. Model construction costs $\mathcal{O}(n^3)$ to compute the factorisation plus $\mathcal{O}(mn^2)$ for the back-substitution with m right-hand sides; and

2. Lagrange polynomial construction costs $\mathcal{O}(n^3)$ in total, coming from one backsolve for each of $n+1$ polynomials (using the factorisation computed in the model construction step).

7.3 Subspace Model-Based DFO for Nonlinear Least-Squares

The core idea behind the methods in this chapter is to construct interpolation models approximating the objective in a subspace, rather than in the full space $\mathbb{R}^n$. This allows us to use interpolation sets with fewer than $n+1$ points, since we do not have to capture the objective's behaviour outside our subspace, which in turn reduces the linear algebra cost of model construction.

DFO-LS with reduced initialisation cost also allows fewer than $n+1$ interpolation points (Section 3.2.1), where by default our model only approximates the objective in a subspace

(Lemma 3.4). However, in that setting we modify the model so that it varies over the whole space $\mathbb{R}^n$, so that our interpolation set eventually has $n+1$ points and our model is full-dimensional. Here, our goal is scalability, so we keep our model low-dimensional, and instead change the subspace at each iteration.

In this section, we outline this process for nonlinear least-squares problems (2.6).[2]

7.3.1 Reduced Subspace Interpolation Models

Similar to Section 3.2.1, we assume that, at iteration k, our interpolation set has $p+1$ points $\{\mathbf{x}_k, \mathbf{y}_1, \dots, \mathbf{y}_p\} \subset \mathbb{R}^n$ with $1 \leq p \leq n$. We seek an interpolating model

$$\mathbf{r}(\mathbf{x}_k + \mathbf{s}) \approx \mathbf{m}_k(\mathbf{s}) := \mathbf{r}(\mathbf{x}_k) + J_k \mathbf{s}, \tag{7.3}$$

where we have already imposed the interpolation condition $\mathbf{m}_k(\mathbf{0}) = \mathbf{r}(\mathbf{x}_k)$. The other interpolation conditions $\mathbf{m}_k(\mathbf{y}_t - \mathbf{x}_k) = \mathbf{r}(\mathbf{y}_t)$ for $t = 1, \dots, p$ yield the $p \times n$ linear system

$$L_k \, \mathbf{g}_{k,i} := \begin{bmatrix} (\mathbf{y}_1 - \mathbf{x}_k)^\top \\ \vdots \\ (\mathbf{y}_p - \mathbf{x}_k)^\top \end{bmatrix} \mathbf{g}_{k,i} = \begin{bmatrix} r_i(\mathbf{y}_1) - r_i(\mathbf{x}_k) \\ \vdots \\ r_i(\mathbf{y}_p) - r_i(\mathbf{x}_k) \end{bmatrix}, \tag{7.4}$$

for $i = 1, \dots, m$, where $\mathbf{g}_{k,i}^\top$ is the i-th row of J_k. This is the same as the model-construction process for DFO-LS (3.6), but with $p < n$. As in the case of DFO-LS with reduced initialisation cost (Section 3.2.1), we seek the minimal-norm solution to the underdetermined system (7.4):

$$\mathbf{g}_{k,i} = \arg\min_{\mathbf{g}_i} \|\mathbf{g}_i\|^2 \qquad \text{s.t.} \qquad L_k \, \mathbf{g}_i = \begin{bmatrix} r_i(\mathbf{y}_1) - r_i(\mathbf{x}_k) \\ \vdots \\ r_i(\mathbf{y}_p) - r_i(\mathbf{x}_k) \end{bmatrix}, \tag{7.5}$$

for $i = 1, \dots, m$.

Calculation of Interpolation Model To solve (7.5), assuming that $\{\mathbf{y}_1 - \mathbf{x}_k, \dots, \mathbf{y}_p - \mathbf{x}_k\}$ are linearly independent, we follow [85, Algorithm 5.7.2] and compute the QR factorisation

$$L_k^\top = Q_k R_k, \tag{7.6}$$

where $Q_k \in \mathbb{R}^{n \times p}$ has orthonormal columns and $R_k \in \mathbb{R}^{p \times p}$ is upper-triangular. This gives us

$$\mathbf{g}_{k,i} = Q_k \hat{\mathbf{g}}_{k,i}, \qquad \text{where} \qquad \hat{\mathbf{g}}_{k,i} := R_k^{-\top} \begin{bmatrix} r_i(\mathbf{y}_1) - r_i(\mathbf{x}_k) \\ \vdots \\ r_i(\mathbf{y}_p) - r_i(\mathbf{x}_k) \end{bmatrix}, \tag{7.7}$$

[2] Our implementation of DFBGN does not include bound constraints (2.15), as bound constraints become linear inequality constraints when projected into the subspace at each iteration.

for $i = 1, \ldots, m$. Thus $J_k = \hat{J}_k Q_k^\top$, where $\hat{J}_k \in \mathbb{R}^{m \times p}$ has rows $\hat{\mathbf{g}}_{k,i}^\top$ (i.e. J_k has column rank at most p, as per Lemma 3.4). This tells us that $\mathbf{m}_k$ is non-constant only for directions in $\mathcal{Y}_k := \text{col}(Q_k) \cong \mathbb{R}^p$. That is, our model is effectively p-dimensional, and we have

$$\mathbf{m}_k(Q_k \hat{\mathbf{s}}) = \hat{\mathbf{m}}_k(\hat{\mathbf{s}}) := \mathbf{r}(\mathbf{x}_k) + \hat{J}_k \hat{\mathbf{s}}, \qquad \forall \hat{\mathbf{s}} \in \mathbb{R}^p. \tag{7.8}$$

Using (7.8), we can then define a local quadratic model for the full objective in this reduced space, given by

$$f(\mathbf{x}_k + Q_k \hat{\mathbf{s}}) \approx \hat{m}_k(\hat{\mathbf{s}}) := \frac{1}{2} \|\hat{\mathbf{m}}_k(\hat{\mathbf{s}})\|^2, \qquad \forall \hat{\mathbf{s}} \in \mathbb{R}^p, \tag{7.9}$$

and calculate a tentative step as

$$\mathbf{s}_k = Q_k \hat{\mathbf{s}}_k \in \mathbb{R}^n, \qquad \text{where} \qquad \hat{\mathbf{s}}_k = \operatorname*{arg\,min}_{\hat{\mathbf{s}} \in \mathbb{R}^p} \hat{m}_k(\hat{\mathbf{s}}), \quad \text{s.t.} \quad \|\hat{\mathbf{s}}\| \leq \Delta_k. \tag{7.10}$$

Linear Algebra Cost This approach yields substantial reductions in the required linear algebra costs compared to DFO-LS:

- Model construction now costs $\mathcal{O}(np^2)$ for factorisation (7.6) and $\mathcal{O}(mp^2)$ for back-substitution (7.7), rather than $\mathcal{O}(n^3)$ and $\mathcal{O}(mn^2)$ respectively; and
- Lagrange polynomial construction costs $\mathcal{O}(p^3)$ rather than $\mathcal{O}(n^3)$.[3]

As well as these reductions, we also get a smaller trust-region subproblem (7.10)—in $\mathbb{R}^p$ rather than $\mathbb{R}^n$—and smaller memory requirements for storing the model Jacobian: we only store $\hat{J}_k$ and Q_k, requiring $\mathcal{O}((m+n)p)$ memory rather than $\mathcal{O}(mn)$ for storing the full $m \times n$ Jacobian. However, in (7.10), we do have the extra cost of projecting $\hat{\mathbf{s}}_k \in \mathbb{R}^p$ into the full space $\mathbb{R}^n$, which requires a multiplication by Q_k, costing $\mathcal{O}(np)$. In addition to the reduced linear algebra costs, the smaller interpolation set means we have a lower evaluation cost to construct the initial model of $p+1$ evaluations (rather than $n+1$).

No particular choice of p is needed for this method, and anything from $p = 1$ (i.e. coordinate search) to $p = n$ (i.e. full space search) is allowed. However, unsurprisingly, we shall see that larger values of p give better performance in terms of evaluations, except for the very low-budget phase, where smaller values of p benefit from a lower initialisation cost. Hence, we expect that our approach with small p is useful when the $\mathcal{O}(mn^2 + n^3)$ per-iteration linear algebra cost of DFO-LS is too great, and reducing the linear algebra cost is worth (possibly) needing more objective evaluations to achieve a given accuracy. As a result, p should in general be set as large as possible, given the linear algebra costs the user is willing to bear.

[3] Here, we find the (linear) Lagrange polynomials using (7.5) and the usual interpolation conditions $\ell_s(\mathbf{y}_t) = \delta_{s,t}$. Thus these polynomials vary only in $\mathcal{Y}_k$, and are effectively low-dimensional.

Algorithm 7.1 DFBGN: Derivative-Free Block Gauss-Newton.

Input: Starting point $\mathbf{x}_0 \in \mathbb{R}^n$, initial trust region radius $\Delta_0 > 0$, block size $p \in [1, n]$, and number of points to drop at each iteration $p_{\text{drop}} \in [1, p]$.

Parameters: maximum and minimum trust-region radii $\Delta_{\max} \geq \Delta_0 > \Delta_{\text{end}} > 0$, trust-region radius scalings $0 < \gamma_{\text{dec}} < 1 < \gamma_{\text{inc}} \leq \overline{\gamma}_{\text{inc}}$, and acceptance thresholds $0 < \eta_1 \leq \eta_2 < 1$.

Select random orthonormal directions $\mathbf{s}_1, \dots, \mathbf{s}_p \in \mathbb{R}^n$ using Algorithm 7.2, and build initial interpolation set $Y_0 := \{\mathbf{x}_0, \mathbf{x}_0 + \Delta_0 \mathbf{s}_1, \dots, \mathbf{x}_0 + \Delta_0 \mathbf{s}_p\}$.
for $k = 0, 1, 2, \dots$ **do**
    Given $\mathbf{x}_k$ and Y_k, build subspace models $\hat{\mathbf{m}}_k : \mathbb{R}^p \to \mathbb{R}^m$ (7.8) and $\hat{m}_k : \mathbb{R}^p \to \mathbb{R}$ (7.9).
    Approximately solve the subspace trust-region subproblem in $\mathbb{R}^p$ (7.10) and project into the full space to get a step $\mathbf{s}_k \in \mathbb{R}^n$.
    Evaluate $\mathbf{r}(\mathbf{x}_k + \mathbf{s}_k)$ and calculate ratio R_k (3.3).
    Accept/reject step and update trust region radius: set

$$\mathbf{x}_{k+1} = \begin{cases} \mathbf{x}_k + \mathbf{s}_k, & R_k \geq \eta_1, \\ \mathbf{x}_k, & R_k < \eta_1, \end{cases} \quad \text{and} \quad \Delta_{k+1} = \begin{cases} \min(\max(\gamma_{\text{inc}} \Delta_k, \overline{\gamma}_{\text{inc}} \|\mathbf{s}_k\|), \Delta_{\max}), & R_k \geq \eta_2, \\ \max(\gamma_{\text{dec}} \Delta_k, \|\mathbf{s}_k\|), & \eta_1 \leq R_k < \eta_2, \\ \min(\gamma_{\text{dec}} \Delta_k, \|\mathbf{s}_k\|), & R_k < \eta_1. \end{cases} \tag{7.11}$$

    if $\Delta_{k+1} \leq \Delta_{\text{end}}$, **terminate**.
    if $p < n$ **then**
        Set $Y_{k+1}^{\text{init}} = Y_k \cup \{\mathbf{x}_k + \mathbf{s}_k\}$.
        Remove $\min(\max(p_{\text{drop}}, 2), p)$ points from Y_{k+1}^{init} (without removing $\mathbf{x}_{k+1}$) using Algorithm 7.3.
    else
        Set $Y_{k+1}^{\text{init}} = Y_k \cup \{\mathbf{x}_k + \mathbf{s}_k\} \setminus \{\mathbf{y}\}$ for some $\mathbf{y} \in Y_k \setminus \{\mathbf{x}_{k+1}\}$.
        Remove $\min(\max(p_{\text{drop}}, 1), p)$ points from Y_{k+1}^{init} (without removing $\mathbf{x}_{k+1}$) using Algorithm 7.3.
    end if
    Let $q := p + 1 - |Y_{k+1}^{\text{init}}|$, and generate random orthonormal vectors $\{\mathbf{d}_1, \dots, \mathbf{d}_q\}$ that are also orthogonal to $\{\mathbf{y} - \mathbf{x}_{k+1} : \mathbf{y} \in Y_{k+1}^{\text{init}} \setminus \{\mathbf{x}_{k+1}\}\}$, using Algorithm 7.2.
    Set $Y_{k+1} = Y_{k+1}^{\text{init}} \cup \{\mathbf{x}_{k+1} + \Delta_{k+1} \mathbf{d}_1, \dots, \mathbf{x}_{k+1} + \Delta_{k+1} \mathbf{d}_q\}$.
end for

7.3.2 Full Algorithm

Our subspace DFO method, which we call Derivative-Free Block Gauss-Newton (DFBGN) is given in Algorithm 7.1. Its framework is that of model-based trust-region DFO methods. In addition to the above model construction procedure, the key difference to standard model-based trust-region DFO methods is that we do not have poisedness-improving steps. Instead, we have a process to add new directions orthogonal to the current space of directions, $\mathcal{Y}_k := \text{col}(Q_k) = \text{span}\{\mathbf{y}_1 - \mathbf{x}_k, \dots, \mathbf{y}_p - \mathbf{x}_k\}$. The reason for this is that the new step $\mathbf{s}_k$ (7.10) is in $\mathcal{Y}_k$. Hence, like in DFO-LS with reduced initialisation cost (Section 3.2.1), if we simply add $\mathbf{x}_k + \mathbf{s}_k$ into the interpolation set, $\mathcal{Y}_k$ does not change across iterations, and we will never explore the whole space. In DFO-LS, this was handled by making J_k full rank, giving $\mathbf{s}_k \notin \mathcal{Y}_k$ and automatically expanding the search space. Here, we need to keep the interpolation set small to keep the linear algebra cost low, and so we do not have this option. Thus, to ensure convergence, we need a mechanism to change $\mathcal{Y}_k$ between iterations. We do this by deleting points from the interpolation set and adding new directions orthogonal to the existing directions.

Algorithm phase	DFO-LS	DFBGN	Comment
Form $\mathbf{m}_k$ (3.6), $\hat{\mathbf{m}}_k$ (7.8)	$\mathcal{O}(n^3 + mn^2)$	$\mathcal{O}(np^2 + mp^2)$	Factorisation plus linear solves
Form m_k (3.2), $\hat{m}_k$ (7.9)	$\mathcal{O}(mn^2)$	$\mathcal{O}(mp^2)$	Form $J_k^\top J_k$ or $\hat{J}_k^\top \hat{J}_k$
Trust-region subproblem	$\mathcal{O}(n^2)$–$\mathcal{O}(n^3)$	$\mathcal{O}(p^2)$–$\mathcal{O}(p^3)$	Depending on # Steihaug-Toint iters.
Project step to $\mathbb{R}^n$	—	$\mathcal{O}(np)$	Multiply by Q_k, (7.10)
Form new step $\mathbf{x}_k + \mathbf{s}_k$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	
Choose point to replace	$\mathcal{O}(n^3)$	—	Compute Lagrange polys., Alg. B.1
Model improvement	$\mathcal{O}(n^3)$	—	Section 3.3.1, recompute Lag. poly.
Choose points to remove	—	$\mathcal{O}(p^3 + np)$	See Algorithm 7.3 in Section 7.3.4
Generate new directions	—	$\mathcal{O}(np^2)$	See Algorithm 7.2 in Section 7.3.3
Total	$\mathcal{O}(mn^2 + n^3)$	$\mathcal{O}(mp^2 + np^2 + p^3)$	

Table 7.1. Comparison of per-iteration linear algebra costs of DFO-LS (with interpolation rather than regression models) and DFBGN (with block size $p \in [1, n]$).

The mechanism for generating new orthogonal directions, Algorithm 7.2, is random, and outlined in Section 7.3.3. We note that this approach is different to the approaches in DFO-LS for expanding the search space (Section 3.2.2). We still need to specify the process for removing points from the interpolation set (which is geometry-aware and requires the computation of Lagrange polynomials), and how to choose a suitable number of points to remove at each iteration, p_{drop}. We consider these in Sections 7.3.4 and 7.3.5 respectively.

A downside of our approach is that the new orthogonal directions are not chosen by minimising a model for the objective (i.e. not attempting to reduce the objective), as we have no information about how the objective varies outside $\mathcal{Y}_k$. This is the fundamental trade-off between the subspace approach and DFO-LS; we can reduce the linear algebra cost, but must spend objective evaluations to change the search space between iterations.

Linear Algebra Cost of DFBGN In Table 7.1, we expand the comparison of linear algebra costs of DFO-LS and DFBGN from Section 7.3.1. Here, we compare the full per-iteration linear algebra cost of the two methods. The overall per-iteration cost of DFO-LS is $\mathcal{O}(mn^2 + n^3)$ and the cost of DFBGN is $\mathcal{O}(mp^2 + np^2 + p^3)$, depending on the choice of $p \in [1, n]$. The key benefit is that our dependency on the underlying problem dimension n decreases from cubic in DFO-LS to linear in DFBGN (provided $p \ll n$). We also note that both methods have linear cost in the number of residuals m, but with a factor that is significantly smaller in DFBGN than in DFO-LS—$\mathcal{O}(p^2)$ compared to $\mathcal{O}(n^2)$.

Remark 7.1. At all iterations we must compute the QR factorisation (7.6). However, we note, similar to Section 3.3.2, that adding, removing and changing interpolation points all induce simple changes to $L_k^\top$ (adding or removing columns, and low-rank updates). This means that (7.6) can be computed with cost $\mathcal{O}(np)$ per iteration using the updating methods in [85, Section 12.5]. In our implementation, however, we do not do this, as we find that

these updates introduce errors[4] that accumulate at every iteration and reduce the accuracy of the resulting interpolation models. To maintain the numerical performance of our method, we need to recompute (7.6) from scratch regularly (e.g. every 10 iterations), and so would not see the $\mathcal{O}(np)$ per-iteration cost, on average.

Remark 7.2. The default parameter choices for DFBGN are the same as DFO-LS, namely: $\Delta_{\max} = 10^{10}$, $\Delta_{\text{end}} = 10^{-8}$, $\gamma_{\text{dec}} = 0.5$, $\gamma_{\text{inc}} = 2$, $\overline{\gamma}_{\text{inc}} = 4$, $\eta_1 = 0.1$, and $\eta_2 = 0.7$. DFBGN also uses the same default choice $\Delta_0 = 0.1 \max(\|\mathbf{x}_0\|_\infty, 1)$. The default choice of p_{drop} is discussed in Section 7.3.5.

Adaptive Choice of p One approach that we have considered is to allow p to vary between iterations of Algorithm 7.1, rather than being constant throughout. Instead of adding $p+1-p_{\text{drop}}$ new points at the end of each iteration (line 15), we implement a variable p by adding at least one new point to the interpolation set, continuing until some criterion is met. This criterion is designed to allow p small when such a p allows us to make reasonable progress, but to grow p up to $p \approx n$ when necessary.

We have tested several possible criteria—comparing some combination of model gradient and Hessian, trust-region radius, trust-region step length, and predicted decrease from the trust-region step—and found the most effective to be comparing the model gradient and Hessian with the trust-region radius. Specifically, we continue adding new directions until (c.f. Lemmas 4.8 and 4.23)

$$\frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)} \geq \alpha \Delta_k, \tag{7.12}$$

for some $\alpha > 0$ (we use $\alpha = 0.2(n-p)/n$ for an interpolation set with $p+1$ points). However, our numerical testing has shown that DFBGN with p fixed outperforms this approach for all budget and accuracy levels, on both medium- and large-scale problems, and so we do not consider it further here. We delegate further study of this approach to future work, to see if alternative adaptive choices for p can be beneficial.

7.3.3 Generation of New Directions

We now detail how new directions $\mathbf{d}_1, \ldots, \mathbf{d}_q$ are created in line 15 of DFBGN (Algorithm 7.1). The same approach is suitable for generating the initial directions $\mathbf{s}_1, \ldots, \mathbf{s}_p$ in line 1 of DFBGN, using $\widetilde{A} = A$ below (i.e. no Q required).

Suppose our current subspace is defined by the orthonormal columns of $Q \in \mathbb{R}^{n \times p_1}$, and we wish to generate q new orthonormal vectors that are also orthogonal to the columns of Q (with $p_1 + q \leq n$). When called in line 15 of DFBGN, we will have $p_1 = p - p_{\text{drop}}$ and

[4] Leading to $L_k^\top \neq Q_k R_k$, not relating to Q_k orthogonal or R_k upper-triangular.

Algorithm 7.2 Mechanism for generating new directions in DFBGN.

Input: Orthonormal basis for current subspace $Q \in \mathbb{R}^{n \times p_1}$ (optional), number of new directions $q \leq n - p_1$.

1: Generate $A \in \mathbb{R}^{n \times q}$ with i.i.d. standard normal entries.
2: If Q specified, calculate $\widetilde{A} = A - QQ^\top A$, otherwise set $\widetilde{A} = A$.
3: Perform the QR factorisation $\widetilde{Q}\widetilde{R} = \widetilde{A}$ and return $\mathbf{d}_1, \ldots, \mathbf{d}_q$ as the columns of $\widetilde{Q}$.

$q = p_{\text{drop}}$. We use the approach in Algorithm 7.2. From the QR factorisation, the columns of $\widetilde{Q}$ are orthonormal if $\widetilde{A}$ is full rank (which occurs with probability 1; see Lemma 7.3 below). We also have $\text{col}(\widetilde{Q}) = \text{col}(\widetilde{A})$, and so to confirm the columns of $\widetilde{Q}$ are orthogonal to Q, we only need to check that the columns of $\widetilde{A}$ are orthogonal to Q. Let $\widetilde{\mathbf{a}}_i$ be the i-th column of $\widetilde{A}$ and $\mathbf{q}_j$ be the j-th column of Q. Then, if $\mathbf{a}_i$ is the i-th column of A, we have

$$\widetilde{\mathbf{a}}_i^\top \mathbf{q}_j = \mathbf{a}_i^\top (I - QQ^\top)\mathbf{q}_j = \mathbf{a}_i^\top (\mathbf{q}_j - Q\mathbf{e}_j) = 0, \tag{7.13}$$

as required.

The cost of Algorithm 7.2 is $\mathcal{O}(nq)$ to generate A, $\mathcal{O}(np_1q)$ to form $\widetilde{A}$ and $\mathcal{O}(nq^2)$ for the QR factorisation. Since $p_1, q \leq p$ (since p_1 is the number of directions remaining in the interpolation set and q is the number of new directions to be added), the whole process has cost at most $\mathcal{O}(np^2)$. This bound is tight, up to constant factors, as we could take $p_1 = q = p/2$, for instance.

Lemma 7.3. *The matrix $\widetilde{A}$ has full column rank with probability 1.*

Proof. Let $\mathbf{a}_i$ and $\widetilde{\mathbf{a}}_i$ be the i-th columns of A and $\widetilde{A}$ respectively. From [74, Proposition 7.1], A has full column rank with probability 1, and each $\mathbf{a}_i \notin \text{col}(Q)$ with probability 1. Now suppose we have constants $c_1, \ldots, c_q$ so that $\sum_{i=1}^q c_i \widetilde{\mathbf{a}}_i = \mathbf{0}$. Then since $\widetilde{\mathbf{a}}_i = \mathbf{a}_i - QQ^\top \mathbf{a}_i$, we have

$$\sum_{i=1}^q c_i \mathbf{a}_i = \sum_{i=1}^q c_i QQ^\top \mathbf{a}_i. \tag{7.14}$$

The right-hand side is in $\text{col}(Q)$, so since $\mathbf{a}_i \notin \text{col}(Q)$, we must have $\sum_{i=1}^q c_i \mathbf{a}_i = \mathbf{0}$. Thus $c_1 = \cdots = c_q = 0$ since A has full column rank, and so $\widetilde{A}$ has full column rank. □

7.3.4 Geometry Management

In the description of Algorithm 7.1, there are no explicit mechanisms in DFBGN to ensure that the interpolation set is well-poised. The implementation of DFBGN ensures that the interpolation set has good geometry through two mechanisms:

- We use a geometry-aware mechanism for removing points, based on Section 3.3.2, which requires the computation of Lagrange polynomials. This mechanism is given in Algorithm 7.3, and is called in lines 10 and 13 of Algorithm 7.1, as well as to select a point to replace in line 12; and

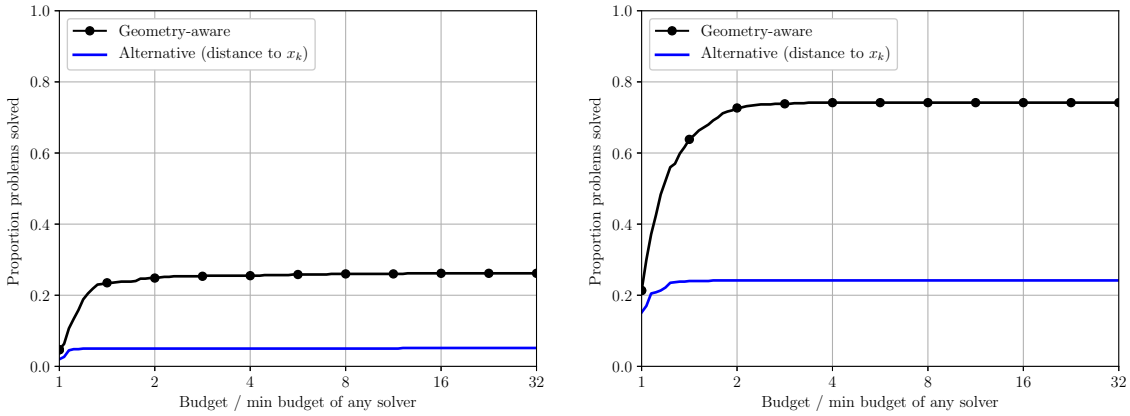
Algorithm 7.3 Mechanism for removing points from the interpolation set in DFBGN.

Input: Interpolation set $\{\mathbf{x}_{k+1}, \mathbf{y}_1, \dots, \mathbf{y}_p\}$ with current iterate $\mathbf{x}_{k+1}$, trust-region radius $\Delta_{k+1} > 0$ and number of points to remove $p_{\text{drop}} \in [1, p]$.

1: Compute the (linear) Lagrange polynomials for $\{\mathbf{x}_{k+1}, \mathbf{y}_1, \dots, \mathbf{y}_p\}$ in the same way as (7.5).
2: For $t = 1, \dots, p$ (i.e. all interpolation points except $\mathbf{x}_{k+1}$), compute

$$\theta_t := \max_{\mathbf{x} \in B(\mathbf{x}_{k+1}, \Delta_{k+1})} |\ell_t(\mathbf{x})| \cdot \max\left(\frac{\|\mathbf{y}_t - \mathbf{x}_{k+1}\|^4}{\Delta_{k+1}^4}, 1\right). \tag{7.15}$$

3: Remove the p_{drop} interpolation points with the largest values of θ_t.



(a) DFBGN with $p = n/10$ *(b) DFBGN with $p = n$*

Figure 7.2. Performance profiles (in evaluations) for DFBGN when $p = n/10$ and $p = n$, comparing removing points with the geometry-aware Algorithm 7.3 and without Lagrange polynomials (by distance to the current iterate). We use accuracy level $\tau = 10^{-5}$, and results are an average of 10 runs, each with a budget of $100(n+1)$ evaluations. The problem collection is (CR).

- Adding new directions that are orthogonal to existing directions, and of length Δ_k, means adding these new points never causes the interpolation set to have poor poisedness.

Together, these two mechanisms mean that any points causing poor poisedness are quickly removed, and replaced by high-quality interpolation points (orthogonal to existing directions, and within distance Δ_k of the current iterate).

The linear algebra cost of Algorithm 7.3 is $\mathcal{O}(p^3)$ to compute p Lagrange polynomials with cost $\mathcal{O}(p^2)$ each (since we already have a factorisation of $L_k^\top$). Then for each t we must evaluate θ_t (7.15), with cost $\mathcal{O}(p)$ to maximise $\ell_t(\mathbf{x})$ (since ℓ_t is linear and varies only in directions $\mathcal{Y}_k$), and $\mathcal{O}(n)$ to calculate $\|\mathbf{y}_t - \mathbf{x}_{k+1}\|$.[5] This gives a total cost of $\mathcal{O}(p^3 + np)$.

[5] We could instead compute $\|\mathbf{y}_t - \mathbf{x}_{k+1}\|$ by taking the norm of the t-th column of R_k, provided we have the factorisation (7.6), for cost $\mathcal{O}(p)$ for each t. This does not affect the overall conclusion of Table 7.1.

Alternative Point Removal Mechanism Instead of Algorithm 7.3, we could have used a simpler mechanism for removing points, such as removing the points furthest from the current iterate (with total cost[6] $\mathcal{O}(np)$). However, this leads to a substantial performance penalty. In Figure 7.2, we compare these two approaches for selecting points to be removed, namely Algorithm 7.3 and distance to $\mathbf{x}_{k+1}$, on the (CR) test set with $p = n/10$ and $p = n$ (using the default value of p_{drop}, as detailed in Section 7.3.5). We see that the geometry-aware criterion (7.15) gives substantially better performance than the cheaper criterion.

7.3.5 Selecting an Appropriate Value of p_{drop}

An important component of Algorithm 7.1 that we have not yet specified is how many points to remove from the interpolation set at each iteration, $p_{\text{drop}} \in [1, p]$.

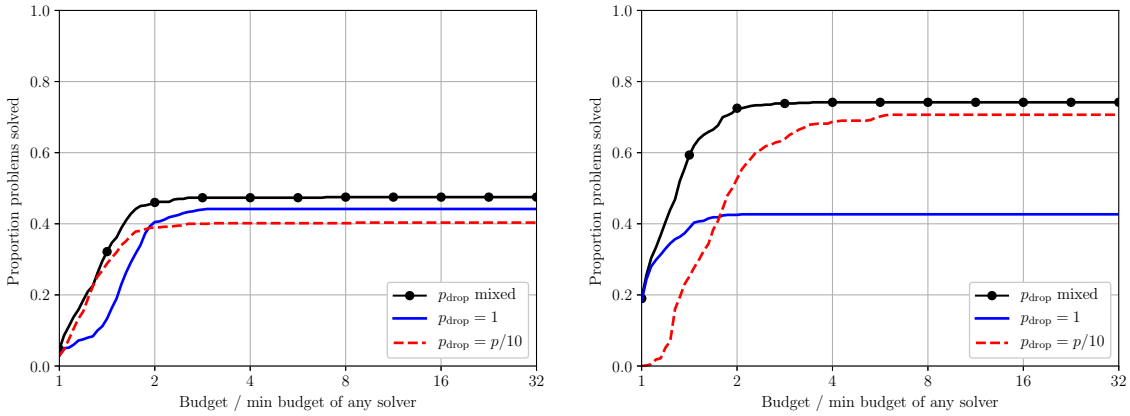
On one hand, a large p_{drop} enables us to change the subspace by a large amount between iterations, ensuring we explore the whole of $\mathbb{R}^n$ quickly, rather than searching in unproductive subspaces for many iterations. However, a small p_{drop} means we require few objective evaluations per iteration, and so are more likely to use our evaluation budget efficiently. We consider two choices of p_{drop}, aimed at each of these possible benefits: $p_{\text{drop}} = p/10$ to change subspaces quickly, and $p_{\text{drop}} = 1$ (the minimum possible value) to use few objective evaluations.

Another approach that we consider is a compromise between these two choices. We note that having $p_{\text{drop}} = 1$ is useful to make progress with few evaluations, so we use this value while we are making progress—we consider this to occur when we have a successful iteration (i.e. $R_k \geq \eta_1$). When we are not progressing (i.e. unsuccessful steps with $R_k < \eta_1$), we use the larger value $p_{\text{drop}} = p/10$.

We compare these three approaches on the (CR) problem collection with a budget of $100(n + 1)$ evaluations in Figure 7.3. Since these different choices of p_{drop} are similar when p is small, we show results for block sizes $p = n/2$ and $p = n$. We first see that, even with $p = n/2$, the three approaches all perform similarly. However, for $p = n$ the compromise choice $p_{\text{drop}} \in \{1, p/10\}$ performs better than the two constant-p approaches. In addition, $p_{\text{drop}} = 1$ outperforms $p_{\text{drop}} = p/10$ for small performance ratios, but is less robust and solves fewer problems overall.

Given these results, in DFBGN we use the compromise choice as the default mechanism: $p_{\text{drop}} = 1$ on successful iterations and $p_{\text{drop}} = p/10$ on unsuccessful iterations.

[6] As above, if we have (7.6), we could calculate all distances to $\mathbf{x}_{k+1}$ using columns of R_k, with total cost $\mathcal{O}(p^2)$.



(a) DFBGN with $p = n/2$ *(b) DFBGN with $p = n$*

Figure 7.3. Performance profiles (in evaluations) comparing different choices of p_{drop}, for DFBGN with $p = n/2$ and $p = n$, with accuracy level $\tau = 10^{-5}$. The choice 'p_{drop} mixed' uses $p_{\text{drop}} = 1$ for successful iterations and $p_{\text{drop}} = p/10$ for unsuccessful iterations. Results an average of 10 runs, each with a budget of $100(n + 1)$ evaluations. The problem collection is (CR).

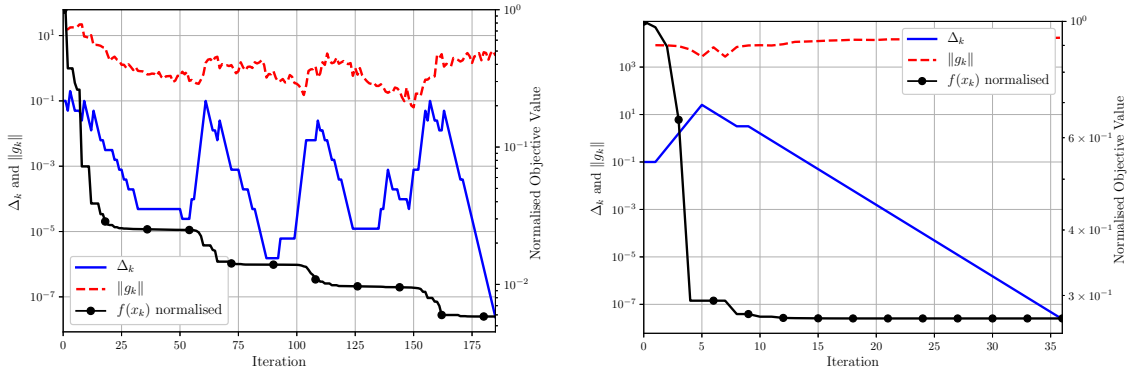
7.4 Choices of Algorithmic Features Compared to DFO-LS

DFBGN is simpler than DFO-LS (Algorithm 3.1), as there are several features in DFO-LS that we do not include in DFBGN. These are: the lower trust-region radius bound ρ_k—although the updating of Δ_k in DFBGN (7.11) is otherwise the same as in DFO-LS (3.4)—and the safety and model-improving phases.[7] We now describe the reasoning behind the removal of these features. We note that we have tested each of these features numerically, and found that they do not substantially affect the practical performance of DFBGN.

No safety phase in DFBGN The safety phase in DFO-LS helps to ensure that Δ_k is not too large compared to $\|\mathbf{g}_k\|$. This is achieved by shrinking Δ_k while it is much larger than the step length, which is connected to the model gradient via Lemma 4.6. In DFBGN, we ensure Δ_k does not get too large compared to $\|\mathbf{s}_k\|$ through (7.11). If $\|\mathbf{s}_k\|$ is much smaller than Δ_k and our step produces a poor objective decrease, then we will set $\Delta_{k+1} \leftarrow \|\mathbf{s}_k\|$ for the next iteration. Although Lemma 4.6 allows $\|\mathbf{s}_k\|$ to be large even if $\|\mathbf{g}_k\|$ is small, in practice we do not observe $\Delta_k \gg \|\mathbf{g}_k\|$ without DFBGN setting $\Delta_{k+1} \leftarrow \|\mathbf{s}_k\|$ after a small number of iterations.

No ρ_k or model-improving phase in DFBGN The roles of ρ_k and the model-improving phases are the same: to ensure that, when we have unsuccessful steps, we do not reduce Δ_k too quickly without first ensuring the quality of the interpolation model.[8]

[7] The implementations of neither DFO-LS nor DFBGN include a criticality phase; see Section 3.3.1.
[8] This is related to ensuring Δ_k does not get too small compared to $\|\mathbf{g}_k\|$ via Lemma 4.5.



(a) DRCAVTY1, $p_{\text{drop}} = 1$ *(b)* LUKSAN13, $p_{\text{drop}} = 1$

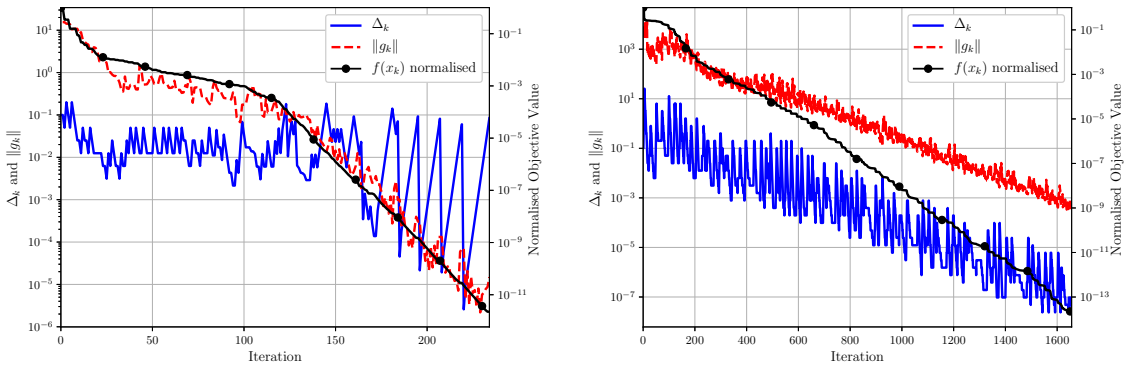
Figure 7.4. Comparison of Δ_k, $\|\mathbf{g}_k\|$ (left) and normalised objective value (right) for DFBGN with $p = n$ and $p_{\text{drop}} = 1$. The two problems are taken from (CR).

In DFBGN, as described in Section 7.3.4, we maintain the geometry of the interpolation set by replacing poorly-located points with orthogonal directions around the current iterate; in practice this ensures the quality of the interpolation set. However, the choice of p_{drop} has a large impact on causing Δ_k to shrink too quickly.

In many cases, DFBGN may reach a point where its model is not accurate and we start to have unsuccessful iterations. To fix this (and continue making progress), we need to introduce several new interpolation points to produce a high-quality model. If p_{drop} is small, this may take many unsuccessful iterations, causing Δ_k to decrease quickly.

The result of having p_{drop} small is seen in Figure 7.4. Here, we show Δ_k, $\|\mathbf{g}_k\|$ and $f(\mathbf{x}_k)$ for DFBGN with $p = n$ and $p_{\text{drop}} = 1$ for two problems from (CR). Both problems show that Δ_k can quickly shrink to be much smaller than $\|\mathbf{g}_k\|$ before reaching optimality. In the case of DRCAVTY1, we see multiple instances where, after several unsuccessful iterations, we recover a high-quality model and continue making progress (causing Δ_k to increase again); this manifests itself as large oscillations in Δ_k with comparatively little change in $\|\mathbf{g}_k\|$. Ultimately, as we terminate on $\Delta_k \leq \Delta_{\text{end}} = 10^{-8}$, DFBGN quits without solving the problem (reaching accuracy $\tau \approx 6 \times 10^{-3}$). A more extreme version of this behaviour is seen for problem LUKSAN13, where we terminate on small Δ_k in the first sequence of unsuccessful iterations—DFBGN does not allow enough time to recover a high-quality model and terminates after achieving accuracy $\tau \approx 0.3$.

Choice of p_{drop} to prevent $\Delta_k \ll \|\mathbf{g}_k\|$ This effect is mitigated by our default choice of $p_{\text{drop}} \in \{1, p/10\}$. By using a larger p_{drop} on unsuccessful iterations, when our model is performing poorly, our interpolation set is changed quickly. This results in DFBGN recovering a high-quality model after a smaller decrease in Δ_k. To demonstrate this, in



(a) DRCAVTY1, p_{drop} *mixed* *(b)* LUKSAN13, p_{drop} *mixed*

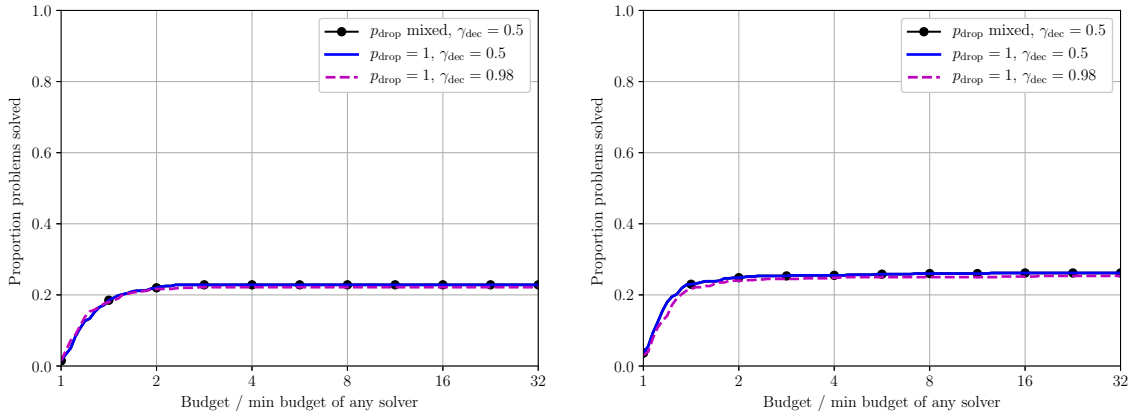
Figure 7.5. Comparison of Δ_k, $\|\mathbf{g}_k\|$ (left) and normalised objective value (right) for DFBGN with $p = n$ and the default $p_{\text{drop}} \in \{1, p/10\}$. The two problems are taken from (CR).

Figure 7.5 we show the results of DFBGN with this p_{drop} for the same problems as Figure 7.4 above. In both cases, we still see oscillations in Δ_k, but their magnitude is substantially reduced—it takes fewer iterations to get successful steps, and Δ_k stays well above Δ_{end}. This leads to both problems being solved to high accuracy.

In Figure 7.5, we also see that, as we approach the solution, $\|\mathbf{g}_k\|$ and Δ_k decrease at the same rate, as we would hope. For DRCAVTY1 after iteration 150, we also see the phenomenon described above, where Δ_k can become much larger than $\|\mathbf{g}_k\|$ due to many successful iterations, before an unsuccessful iteration with $\|\mathbf{s}_k\|$ small means that Δ_k returns to the level of $\|\mathbf{g}_k\|$ regularly.

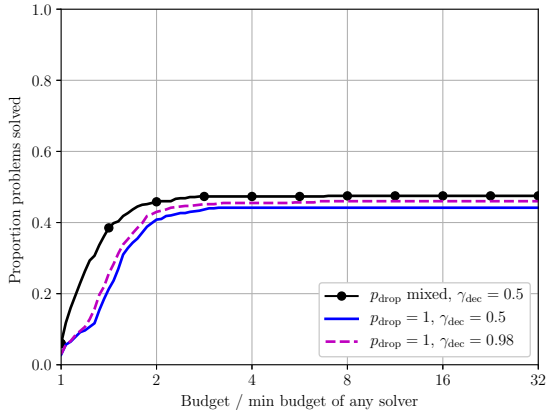
Alternative Mechanism for Recovering High-Quality Models An alternative approach for avoiding unnecessary decreases in Δ_k while the interpolation model quality is improved is to simply decrease Δ_k more slowly on unsuccessful iterations. This corresponds to setting γ_{dec} to be closer to 1, which is the default choice of DFO-LS for noisy problems (see Section 3.3.5).

In Figure 7.6, we compare the DFBGN default choices, of $p_{\text{drop}} \in \{1, p/10\}$ and $\gamma_{\text{dec}} = 0.5$, with $p_{\text{drop}} = 1$ and $\gamma_{\text{dec}} \in \{0.5, 0.98\}$ on the (CR) problem collection. For small values of p (where the different choices of p_{drop} are essentially identical), the choice of γ_{dec} has almost no impact on the performance of DFBGN. For larger values of p, using $\gamma_{\text{dec}} = 0.98$ with $p_{\text{drop}} = 1$ performs comparably well to the DFBGN default ($\gamma_{\text{dec}} = 0.5$ with $p_{\text{drop}} \in \{1, p/10\}$). However, we opt for keeping $\gamma_{\text{dec}} = 0.5$, to allow us to use the larger value for noisy problems (just as in DFO-LS), and to reduce the risk of overfitting our trust-region parameters to a particular problem collection.

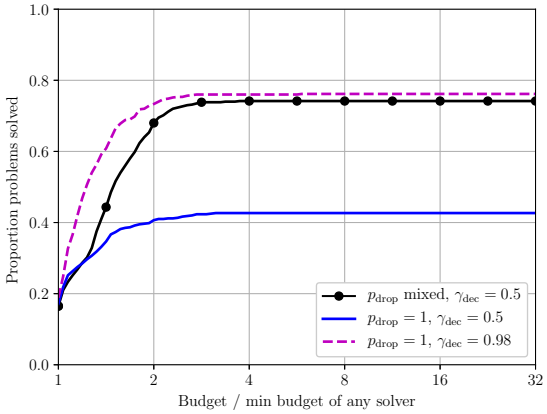


(a) DFBGN with $p = n/100$

(b) DFBGN with $p = n/10$



(c) DFBGN with $p = n/2$



(d) DFBGN with $p = n$

Figure 7.6. Performance profiles (in evaluations) comparing different choices of p_{drop} and γ_{dec} for DFBGN, at accuracy level $\tau = 10^{-5}$. The choice 'p_{drop} mixed' uses $p_{\text{drop}} = 1$ for successful iterations and $p_{\text{drop}} = p/10$ for unsuccessful iterations. Results an average of 10 runs, each with a budget of $100(n + 1)$ evaluations. The problem collection is (CR).

DFBGN for Noisy Problems In our numerical results in Section 7.5, we limit our consideration to problems without noise. This is so that we can isolate the impact on performance of exploring within a subspace at each iteration (trading off extra objective evaluations for change subspaces with reduced linear algebra cost). By not yet addressing noisy problems, we expect to be able to draw clearer conclusions about the fundamental performance of DFBGN. However, noisy objectives are an important class of problem for DFBGN to handle, and we delegate this to future work. Here, we briefly describe what we would need to consider.

To give comparable robustness to noise as DFO-LS, we would need to implement optional default trust-region parameters and a multiple restarts mechanism in DFBGN. Using default parameters has no extra cost in DFBGN than in DFO-LS, but how to implement multiple restarts is not clear. For instance, we would need to consider moving interpolation points

either to geometry-improving points in the current subspace or to new directions outside the current subspace. How best to automatically detect the need for a restart would also have to be considered.

7.5 Numerical Results

In this section we compare the performance of DFBGN to that of DFO-LS. As described in Section 7.3.2, DFBGN is based on the decision to reduce the linear algebra cost of the algorithm at the expense of more objective evaluations per iteration. Here, we will investigate this tradeoff in practice.

In our testing, we consider both the standard version of DFO-LS, and one where we use a reduced initialisation cost of $n/100$ evaluations. This will allow us to compare both the overall performance of DFBGN and its performance with small budgets (since DFBGN also has a reduced initialisation cost of $p + 1$ evaluations). We compare these against DFBGN with the choices $p \in \{n/100, n/10, n/2, n\}$. In all cases we use the default settings of both solvers on 10 instances of each problem with a budget of $100(n + 1)$ evaluations, and allow each solver a maximum runtime of 12 hours for each instance of each problem.[9] We consider the problem collections (CR) and (CR-large), but the runtime limit only impacts the results for (CR-large).

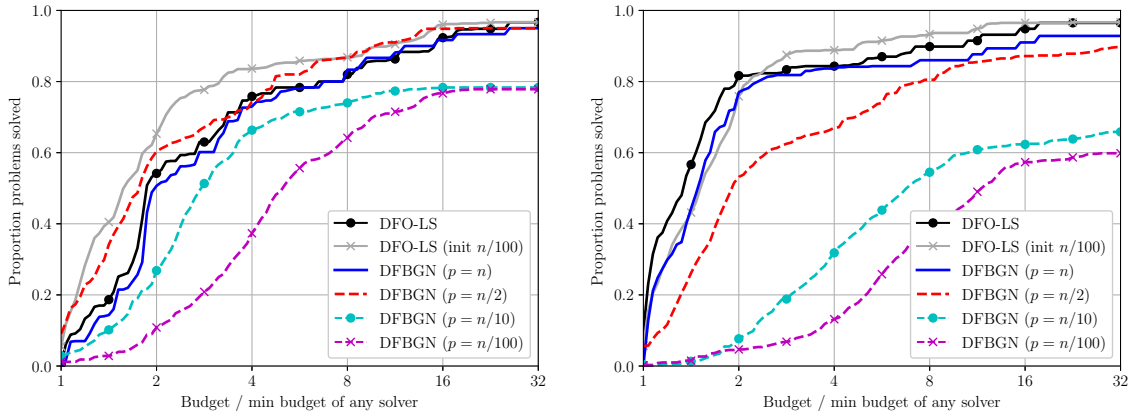
7.5.1 Results Based on Evaluations

We begin our comparisons by considering the performance of DFO-LS and DFBGN when measured in terms of evaluations.

Medium-Scale Problems (CR) First, in Figure 7.7, we show the results for different accuracy levels for the (CR) problem collection (with $n \approx 100$). For the lowest accuracy level $\tau = 0.5$, DFO-LS with reduced initialisation cost is the best-performing solver, followed by DFBGN with $p = n/2$. These correspond to methods with lower initialisation costs than DFO-LS and DFBGN with $p = n$, so this is likely a large driver behind their performance. DFBGN with full block size $p = n$ performs similarly to DFO-LS, and DFBGN with $p = n/10$ and $p = n/100$ perform worst (as they are optimising in a very small subspace at each iteration).

However, as we look at higher accuracy levels, we see that DFO-LS (with and without reduced initialisation cost) performs best, and the DFBGN methods perform worse. The performance gap is more noticeable for small values of p. As expected, this means that

[9] Since all problems are implemented in Fortran via CUTEst, the cost of objective evaluations for this testing is minimal.



(a) $\tau = 0.5$ (b) $\tau = 10^{-1}$

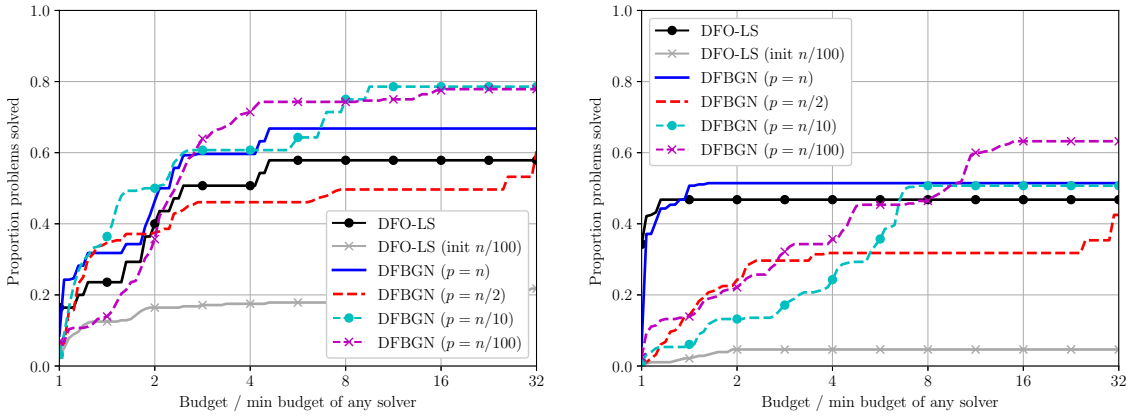
(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-5}$

Figure 7.7. Performance profiles (in evaluations) comparing DFO-LS (with and without reduced initialisation cost) with DFBGN (various p choices) for different accuracy levels. Results are an average of 10 runs for each problem, with a budget of $100(n+1)$ evaluations and a 12 hour runtime limit per instance. The problem collection is (CR).

DFBGN requires more evaluations to achieve these levels of accuracy, and benefits from being allowed to use a larger p. Notably, DFBGN with $p = n$ has only a slight performance loss compared to DFO-LS, even though it uses $p/10$ evaluations on unsuccessful iterations (rather than 1–2 for DFO-LS); this indicates that our choice of p_{drop} provides a suitable compromise between solver robustness and evaluation efficiency.

Large-Scale Problems (CR-large) Next, in Figure 7.8, we show the same plots but for the (CR-large) problem collection, with $n \approx 1000$. Compared to Figure 7.7, the situation is quite different.

At the lowest accuracy level, $\tau = 0.5$, DFBGN with small blocks ($p = n/10$ and $p = n/100$) gives the best-performing solvers, followed by the full-block solvers (DFO-LS and DFBGN with $p = n$). For higher accuracy levels, the performance of DFBGN



(a) $\tau = 0.5$ (b) $\tau = 10^{-1}$

(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-5}$

Figure 7.8. Performance profiles (in evaluations) comparing DFO-LS (with and without reduced initialisation cost) with DFBGN (various p choices) for different accuracy levels. Results are an average of 10 runs for each problem, with a budget of $100(n+1)$ evaluations and a 12 hour runtime limit per instance. The problem collection is (CR-large).

with small p deteriorates compared with the full-block methods. DFBGN with $p = n/2$ is the worst-performing DFBGN variant at low accuracy levels, and performs similar to DFBGN with small p at high accuracy levels. DFO-LS with reduced initialisation cost is the worst-performing solver for this dataset.

Unlike the medium-scale results above, we no longer have a clear trend in the performance of DFBGN as we vary p. Instead, we have a combination of two factors coming into play, which have opposite impacts on the performance of DFBGN as we vary p. On one hand, we have the number of evaluations required for DFBGN (with a given p) to reach the desired accuracy level. On the other hand, we have the number of iterations that DFBGN can perform before reaching the 12 hour runtime limit.

DFBGN with small p requires more evaluations to reach a given level of accuracy (as seen with the medium-scale results), but can perform many evaluations before timing out

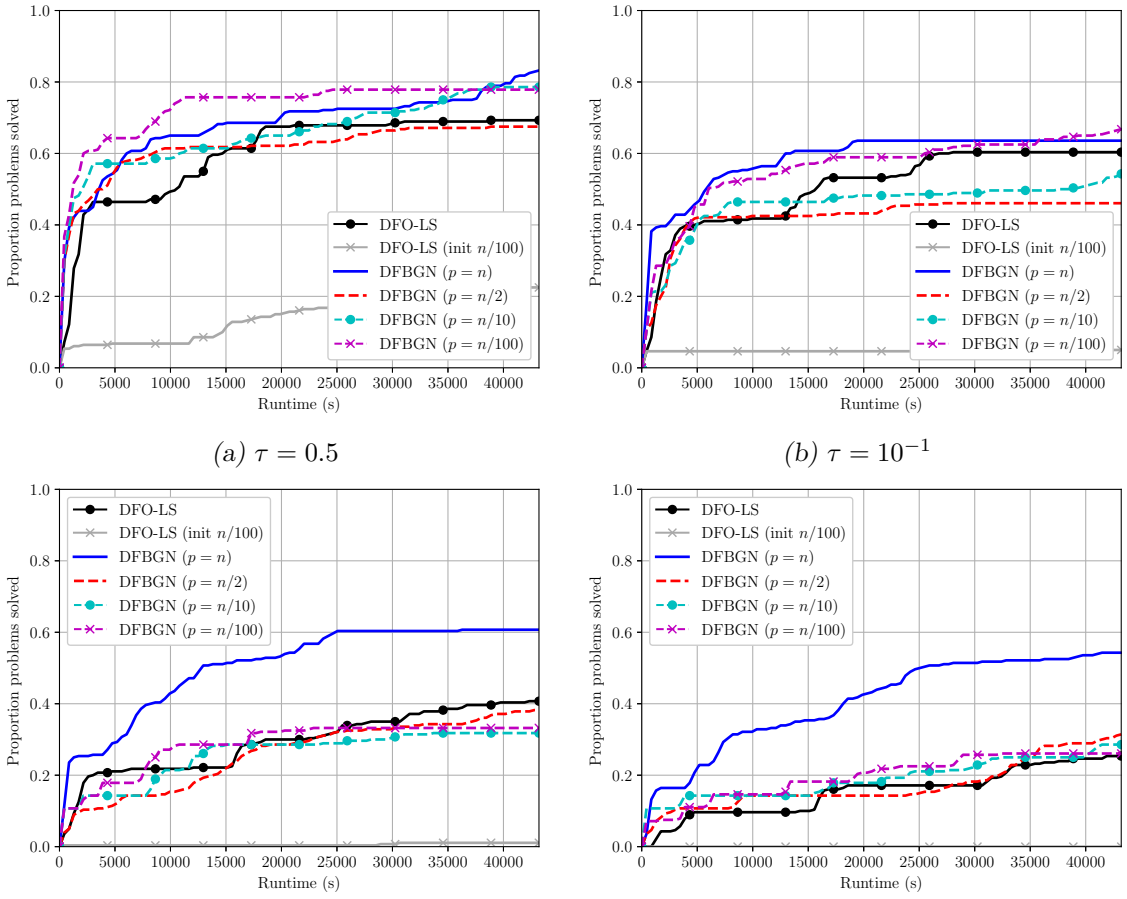
Solver	% timeout
DFO-LS	92.5%
DFO-LS (init $n/100$)	97.9%
DFBGN ($p = n/100$)	34.6%
DFBGN ($p = n/10$)	73.9%
DFBGN ($p = n/2$)	81.8%
DFBGN ($p = n$)	66.4%

Table 7.2. Proportion of problem instances from (CR-large) for which each solver terminated on the maximum 12 hour runtime.

due to its low per-iteration linear algebra cost. This is reflected in it solving many problems to low accuracy, but few problems to high accuracy. By contrast, DFBGN with $p = n$ is allowed to perform fewer iterations before timing out (and hence see fewer evaluations), but requires many fewer evaluations to solve problems, particularly for high accuracy. This manifests in its good performance for low and high accuracy levels. The middle ground, DFBGN with $p = n/2$, has its performance negatively impacted by both issues: it requires many fewer evaluations to solve problems than $p = n$ (especially for high accuracy), but also has a relatively high per-iteration linear algebra cost and times out compared to small p.

Both variants of DFO-LS show worse performance here than for the medium-scale problems. This is because, as suggested by the results in Section 7.2, they are both affected by the runtime limit. DFO-LS with reduced initialisation cost is particularly affected, because of the high cost of the SVD (of the full $m \times n$ Jacobian) at each iteration for these problems. We note that this cost is only noticeable for these large-scale problems, and this variant of DFO-LS is still useful for small- and medium-scale problems, as per Section 5.2.2.

We can verify the impact of the timeout on DFO-LS and DFBGN by considering the proportion of problem instances for (CR-large) for which the solver terminated because of the timeout. These results are presented in Table 7.2. DFO-LS reaches the 12 hour maximum much more frequently than DFBGN: over 90% rather than 35% for DFBGN with $p = n/100$ or 66% for DFBGN with $p = n$ (see Remark 7.4 below). For DFBGN with different values of p, we see the same behaviour as in Figure 7.8. That is, DFBGN with small p times out the least frequently, as its low per-iteration runtime means it performs enough iterations to terminate naturally. For DFBGN with $p = n$, we time out more frequently (due to the high per-iteration runtime), but not as often as with $p = n/2$, as the its superior budget performance for high accuracy levels means it fully solves more problems, even with comparatively fewer iterations. We note that Table 7.2 does not measure what accuracy level was achieved before the timeout, which is better captured in the performance profiles Figure 7.8.



(a) $\tau = 0.5$ (b) $\tau = 10^{-1}$

(c) $\tau = 10^{-3}$ (d) $\tau = 10^{-5}$

Figure 7.9. Data profiles comparing the runtime of DFO-LS (with and without reduced initialisation cost) with DFBGN (various p choices) for different accuracy levels. Results are an average of 10 runs for each problem, with a budget of $100(n+1)$ evaluations and a 12 hour runtime limit per instance. The problem collection is (CR-large).

Remark 7.4. DFBGN with $p = n$ has a similar per-iteration linear algebra cost to DFO-LS. Hence it can perform a similar number of iterations before reaching the runtime limit. However, DFBGN performs more objective evaluations per iteration, because of the choice of p_{drop}. Since DFBGN with $p = n$ has a similar performance to DFO-LS when measured on budget (as seen in Figure 7.7), this means that it has a superior performance when measured by runtime. Additionally, if multiple objective evaluations can be run in parallel, then DFBGN would also be able to benefit from this, unlike DFO-LS.

Remark 7.5. For completeness, in Appendix D we compare DFBGN with DFO-LS on the (MW) problem collection. We do not include this discussion here as these problems are low-dimensional, which is not the main use case for DFBGN.

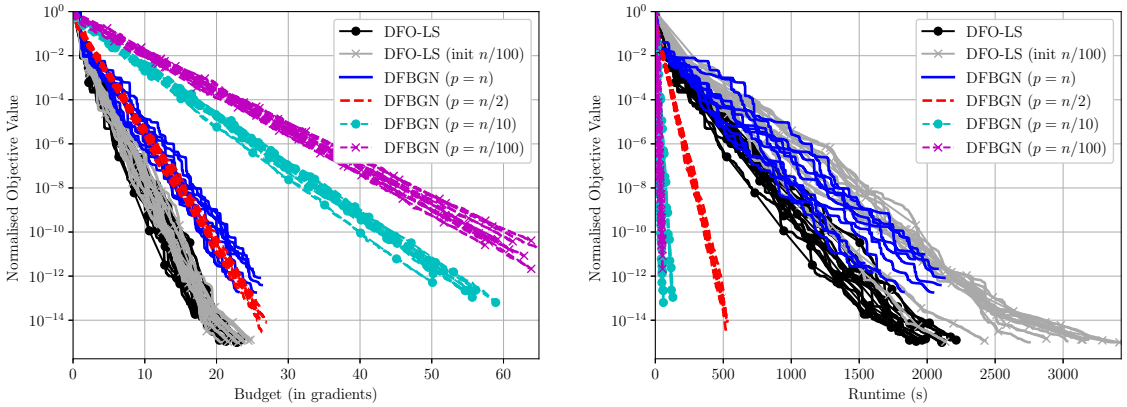
7.5.2 Results Based on Runtime

We have seen above that DFBGN performs well compared to DFO-LS on the (CR-large) problem collection, as the 12 hour timeout causes DFO-LS to terminate after relatively few objective evaluations. In Figure 7.9, we show the same comparison for (CR-large) as in Figure 7.8, but showing data profiles of problems solved versus runtime (rather than evaluations). Here, all DFBGN variants perform similar to or better than DFO-LS for low accuracy levels, since DFBGN has a lower per-iteration runtime than DFO-LS, and this is the regime where DFBGN performs best (on budget). For high accuracy levels, DFBGN with $p = n$ is the best-performing solver, as it uses large enough blocks to solve many problems to high accuracy. By contrast, both DFBGN with small p and DFO-LS perform similarly at high accuracy levels—the impact of the timeout on DFO-LS roughly matches the reduced robustness of DFBGN with small p at these accuracy levels. Again, as we observed above, DFO-LS with reduced initialisation cost is the worst-performing solver, due to the high cost of the SVD at each iteration.

To further see the impact of this issue, we now consider how the solvers perform for a variable-dimension test problem, as we increase the underlying dimension. We run each solver, with the same settings as above, on the CUTEst problem ARWHDNE for different choices of problem dimension n.[10] In Figure 7.10 we plot the objective reduction for each solver against budget and runtime for DFO-LS and DFBGN, showing $n = 100$, $n = 1000$ and $n = 2000$.

We see that, when measured on evaluations, both DFO-LS variants achieve the fastest objective reduction, and that DFBGN with small p achieves the slowest objective reduction. This is in line with our results from Section 7.5.1. However, when we instead consider objective decrease against runtime, we see that DFBGN with small p gives the fastest decrease—the larger number of iterations needed by these variants (as seen by the larger number of evaluations) is offset by the substantially reduced per-iteration linear algebra cost. When viewed against runtime, both DFO-LS variants can only achieve a small objective decrease in the allowed 12 hours, even though they are showing fast decrease against budget, and would achieve higher accuracy than DFBGN if the linear algebra cost were negligible.

[10] This problem appears in the collections (CR) and (CR-large), with $n = 100$ and $n = 1000$ respectively.



(a) $n = 100$, *objective vs. budget*

(b) $n = 100$, *objective vs. runtime*

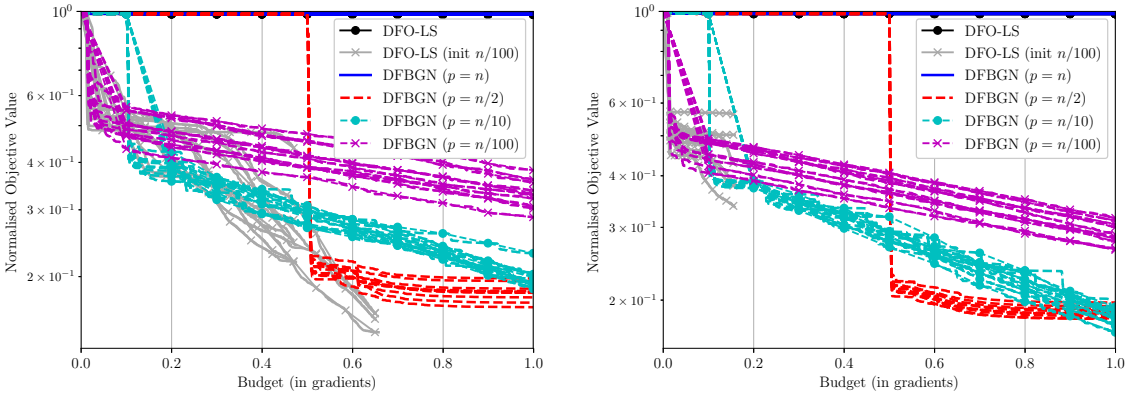
(c) $n = 1000$, *objective vs. budget*

(d) $n = 1000$, *objective vs. runtime*

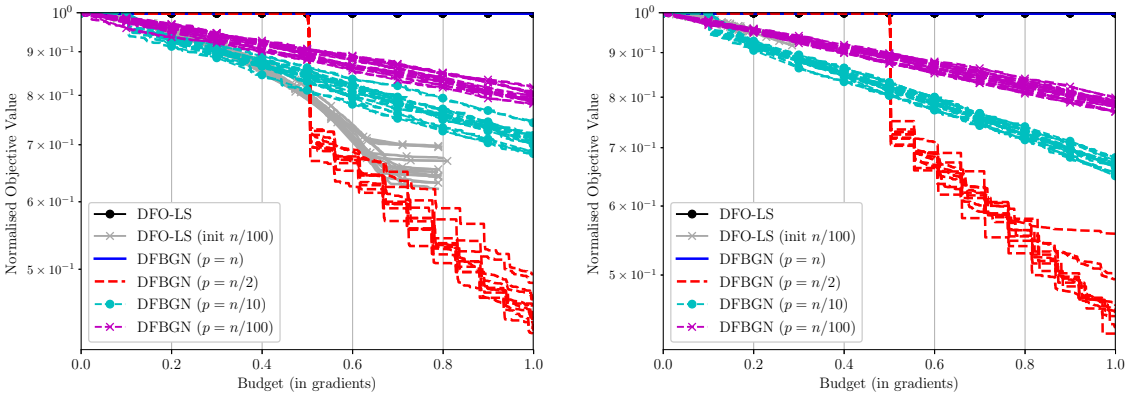
(e) $n = 2000$, *objective vs. budget*

(f) $n = 2000$, *objective vs. runtime*

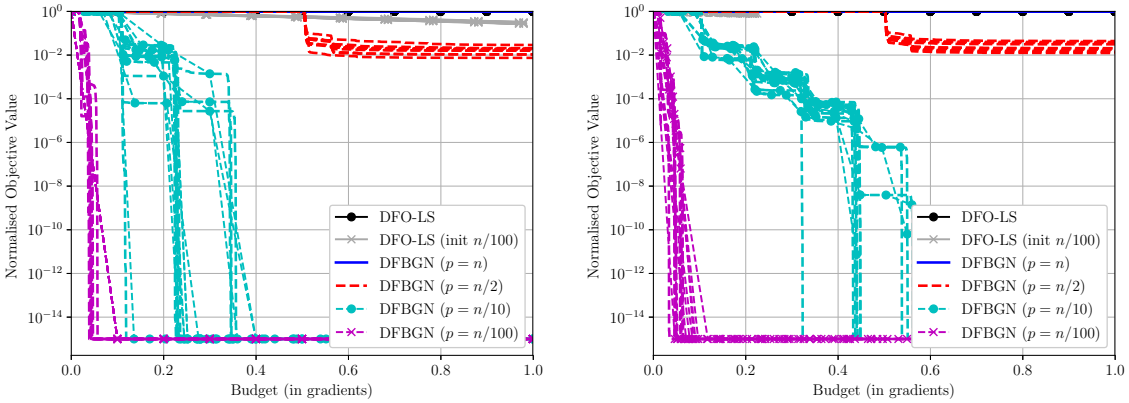
Figure 7.10. Normalised objective value (versus evaluations and runtime) for 10 runs of DFO-LS and DFBGN on CUTEst problem ARWHDNE. *These results use a budget of* $100(n + 1)$ *evaluations and a 12 hour runtime limit per instance.*



(a) ARWHDNE, $n = 1000$ (b) ARWHDNE, $n = 2000$

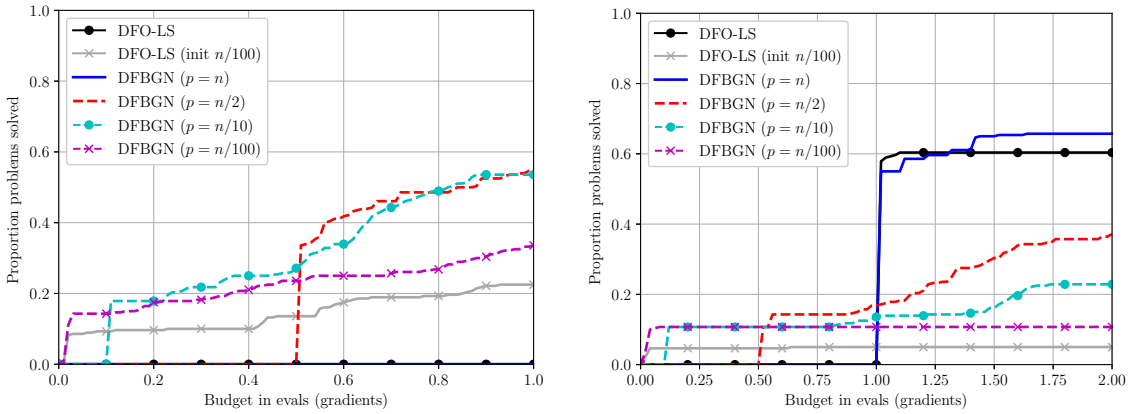


(c) CHANDHEQ, $n = 1000$ (d) CHANDHEQ, $n = 2000$



(e) VARDIMNE, $n = 1000$ (f) VARDIMNE, $n = 2000$

Figure 7.11. Normalised objective value (versus evaluations) for 10 runs of DFO-LS and DFBGN on different CUTEst problems (all with $n = 1000$ and $n = 2000$). These results use a budget of $n + 1$ evaluations and a 12 hour runtime limit per instance.



(a) $\tau = 0.5$*, budget* $n + 1$ *evaluations*

(b) $\tau = 0.1$*, budget* $2(n + 1)$ *evaluations*

Figure 7.12. Data profiles (in evaluations) comparing DFO-LS (with and without reduced initialisation cost) with DFBGN (various p *choices) for different accuracy levels and budgets. Results are an average of 10 runs for each problem, with a budget of* $n + 1$ *or* $2(n + 1)$ *evaluations and a 12 hour runtime limit per instance. The problem collection is (CR-large).*

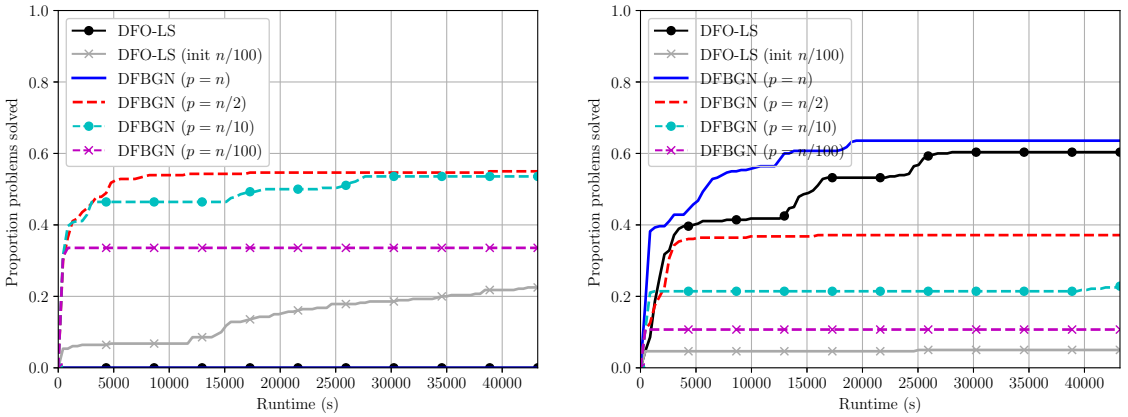
7.5.3 Results for Small Budgets

Another benefit of DFBGN is that it has a small initialisation cost of $p + 1$ evaluations. When n is large, it is more likely for a user to be limited by a budget of fewer than n evaluations. Here, we examine how DFBGN compares for small budgets to DFO-LS with reduced initialisation cost.

We recall from Section 7.3 that DFO-LS with reduced initialisation cost progressively increases the dimension of the subspace of its interpolation model, until it reaches the whole space $\mathbb{R}^n$ (after approximately $n+1$ evaluations), while in DFBGN we restrict the dimension at all iterations.

In Figure 7.11 we consider 3 variable-dimensional CUTEst problems from (CR) and (CR-large), all using $n = 1000$ and $n = 2000$. We show the objective decrease against budget for 10 runs of each solver, restricted to a maximum of $n + 1$ evaluations. Similar to Section 5.2.2 for DFO-LS, we see that the smaller p used in DFBGN, the faster DFBGN is able to make progress (due to the lower number of initial evaluations). However, this is offset by the faster objective decrease achieved by larger p values (after the higher initialisation cost)—if the user can afford a larger p, both in terms of linear algebra and initial evaluations, then this is usually a better option. An exception to this is the problem VARDIMNE, where its simple structure means DFBGN with small p solves the problem to very high accuracy with very few evaluations, substantially outperforming both DFBGN with larger p, and DFO-LS with reduced initialisation cost.

In Figure 7.11 we also show the decrease for DFO-LS with full initialisation cost and DFBGN with $p = n$, but they use the full budget on initialisation, and so make no progress.



(a) $\tau = 0.5$, $n + 1$ *evaluations (vs. runtime)*

(b) $\tau = 0.1$, $2(n + 1)$ *evaluations (vs. runtime)*

Figure 7.13. Data profiles (in runtime) comparing DFO-LS (with and without reduced initialisation cost) with DFBGN (various p choices) for different accuracy levels and budgets. Results are an average of 10 runs for each problem, with a budget of $n + 1$ *or* $2(n + 1)$ *evaluations and a 12 hour runtime limit per instance. The problem collection is (CR-large).*

However, in addition, we show DFO-LS with a reduced initialisation cost of $n/100$ evaluations. This variant performs well, in most cases matching the decrease of DFBGN with $p = n/100$ initially, but achieving a faster objective reduction against budget—this matches with our previous observations. However, the extra cost of the linear algebra means that DFO-LS with reduced initialisation does not end up using the full budget, instead hitting the 12 hour timeout. This is most clear when comparing the results for $n = 1000$ with $n = 2000$, where DFO-LS with reduced initialisation cost begins by achieving a similar decrease in both cases, but hits the timeout more quickly with $n = 2000$, and so terminates after fewer objective evaluations (with a corresponding smaller objective decrease).

We analyse this more systematically in Figure 7.12, where we show data profiles (measured on budget) of DFBGN and DFO-LS on the (CR-large) problem collection, for low accuracy and small budgets. These results verify our conclusions: DFBGN with small p can make progress on many problems with a very short budget (fewer than $n + 1$ evaluations), and outperform DFO-LS with reduced initialisation cost due to its slow runtime. However, once we reach a budget of more than $n + 1$ evaluations, then DFO-LS and DFBGN with $p = n$ become the best-performing solvers (when measuring on evaluations only). They are also able to achieve a higher level of accuracy compared to DFBGN with small p.

Lastly, in Figure 7.13 we show the same results as Figure 7.12, but showing profiles measured on runtime. We note that we are only measuring the linear algebra costs, as the cost of objective evaluation for our problems is negligible. Here, the benefits of DFBGN with small p are not seen. This is because the problems that can be solved by DFBGN with small p using very few evaluations are likely easier, and so can likely be solved by DFBGN

with large p in few iterations. Thus, the runtime requirements for DFBGN with large p to solve the problem are not large—even though they have a higher per-iteration cost, the number of iterations is small. In this setting, therefore, the benefit of DFBGN with small p is not lower linear algebra costs, but fewer evaluations—which is likely to be the more relevant issue in this small-budget regime.

Chapter 8

Conclusions and Future Work

Derivative-free optimisation algorithms have been growing in popularity in recent years due to their applicability to nonlinear optimisation problems where the objective is black-box, expensive to evaluate and/or contains noise. In particular, model-based DFO methods are known to perform well in practice, and are well-suited to capturing structure in the objective. In this thesis, we have developed model-based DFO algorithms for nonlinear least-squares and general minimisation problems that exhibit strong numerical performance and greater flexibility, robustness and scalability compared to state-of-the-art methods.

It is well-known that, for nonlinear least-squares problems, using linear local models for each residual is sufficient to approximate the objective well, especially for zero-residual problems. This forms the basis of the derivative-based Gauss-Newton and Levenberg-Marquardt methods, and has motivated the development of our model-based trust-region method DFO-LS (Chapter 3). This method is based on the DFLS framework from [240], which in principle allows linear, or very close to linear, residual models, but numerically it only assesses the performance of partially or fully quadratic residual models.

Our use of linear residual models gives DFO-LS increased flexibility. As well as having a lower initialisation cost in terms of objective evaluations than quadratic residual models, DFO-LS can allow a further reduced initialisation cost of as few as 2 objective evaluations, after which the main iteration can begin, rather than at least $n + 1$ as in many model-based DFO codes (for an n-dimensional problem). DFO-LS also has numerous techniques for ensuring its robust performance on noisy problems. This is due to its ability to select different, more appropriate, algorithm parameters for noisy problems, and its use of multiple restarts. However, DFO-LS also allows the user to employ a wide family of sample averaging strategies, and linear regression models.

We prove global convergence and provide a worst-case complexity analysis of DFO-LS (Chapter 4). Our analysis extends the results from [240] to include a weaker requirement on the trust-region subproblem solution, and showing that the whole sequence, not just

a subsequence, of the gradients at the iterates converges to zero, as well as a worst-case complexity result. Our worst-case complexity result demonstrates that DFO-LS has the same complexity as model-based DFO methods for general objectives, but with a dependency on dimension that is between first- and second-order methods. This is because DFO-LS is adapted to the least-squares problem structure, and in particular captures partial second-order information.

In Chapter 5, we present the first numerical tests of linear residual models in a DFO context, and we show that DFO-LS reduces both the computational cost of solving the interpolation problem (leading to a runtime reduction of at least a factor of 6) and the memory cost of storing the models (from $\mathcal{O}(mn^2)$ to $\mathcal{O}(mn)$ for a problem with m residuals). These savings result in a substantially faster runtime and improved scalability of DFO-LS compared to DFBOLS, the implementation from [240], which uses (possibly significantly) underdetermined quadratic residual models. Furthermore, the simpler local models do not adversely affect the performance of DFO-LS numerically, in terms of evaluation counts: DFO-LS performs as well as DFBOLS and POUNDERS [225], on smooth test problems from the Moré & Wild and CUTEst collections. When the objective has noise, DFO-LS—without its new features for noisy problems, such as multiple restarts—suffers a small performance penalty compared to DFBOLS and POUNDERS, which is larger for additive than multiplicative noise, but not when considering only nonzero residual problems (compared to all problems).

We also provide extensive testing of the key novel features in DFO-LS. Using a reduced initialisation cost enables DFO-LS to make reasonable progress on several problems even with fewer than n objective evaluations (i.e. less than the cost of evaluating the gradient of the objective at a single point). This improvement has a tradeoff in performance for medium-sized budgets, but achieves the same long-term performance as having a full initialisation cost. For noisy problems, DFO-LS with selected default parameters and using multiple restarts yields a substantial improvement in its robustness for noisy problems. Multiple restarts are cheap to implement and do not cause a deterioration in performance in the early phase of the algorithm, when compared to other techniques for improving robustness to noise, such as sample averaging and regression models. When enhanced with these novel features to improve robustness, DFO-LS outperforms DFBOLS and POUNDERS on noisy problems.

We then consider the problem of general minimisation (Chapter 6), and introduce Py-BOBYQA, our Python implementation of BOBYQA [186]. We take the successful algorithmic framework of BOBYQA (with simplified linear algebra) and enhance it with features from DFO-LS, such as suitable default trust-region parameters and multiple restarts.

As a result, it matches the performance of BOBYQA, as well as NOWPAC [16] on smooth problems. However, these additional features for noisy problems mean it outperforms the original BOBYQA in this setting. We also compare Py-BOBYQA with STORM [50] and SNOWPAC [17] for noisy problems, and we see that multiple restarts benefit (compared to sample averaging, regression models, and surrogate models) both from being cheap to implement and from having strong performance in the early phases of the runs. We note that for stochastic optimisation problems, where the objective is the expectation of a random function [129, Section 1], that STORM and other approaches have convergence guarantees, as they use an increasing number of samples of the objective as they progress. By contrast, the numerical results in Chapter 6 indicate that Py-BOBYQA is able to reach a neighbourhood of the solution with fewer objective evaluations than such methods. However, we do not currently have theoretical guarantees for the convergence of Py-BOBYQA with multiple restarts for noisy problems, and we delegate this to future work. We note that there is work proving the convergence of (derivative-based) methods to a neighbourhood of a solution for noisy problems [232], but we anticipate that it may be possible to prove stronger results for Py-BOBYQA provided sufficiently many restarts are allowed.

We then demonstrate that multiple restarts provide a useful heuristic for Py-BOBYQA to escape local minima. We outline a variant of Py-BOBYQA that further enhances this feature, and find encouraging performance of the improved variant when compared to Bayesian and surrogate algorithms, and DIRECT, SNOBFIT and CMA-ES, for both smooth and noisy problems solved to low or high accuracy requirements. Our experiments illustrate that local derivative free methods may be effective for global optimisation, in the regime typical for global solvers, namely when the objective is black-box, possibly noisy and computationally expensive to evaluate. This finding is useful as these local methods have greater flexibility (such as allowing the use of a good initial guess of the solution), guarantee improvement in current best guess of the solution, and have better scalability.

Finally, in Chapter 7 we introduce DFBGN, a model-based trust-region DFO method for nonlinear least-squares problems aimed at large-scale problems. Like DFO-LS, DFBGN constructs linear residual models via interpolation and therefore simplified quadratic models for the full objective that capture some curvature information. Unlike DFO-LS, however, DFBGN allows for an interpolation set of reduced size, of between 2 and $n + 1$ points, and as such, builds low-dimensional models at each iteration. The choice of the size of the interpolation set allows the user to increase or decrease the linear algebra cost of each iteration based on their computational resources, which we show is the key component affecting the runtime of DFO-LS on large-scale problems. Thus, as a subspace optimisation method, at each iteration, DFBGN selects a new iterate within the subspace of the ambient

space defined by the current interpolation set. However, this requires that DFBGN has a mechanism for changing the subspace between iterations, which we do by removing interpolation points and selecting new points based on random directions orthogonal to the current subspace. We select the number of points changed at each iteration based on whether the iteration is successful or not, and incorporate a geometry-aware mechanism for selecting the points to be removed.

Because of these features, DFBGN can omit several key algorithmic phases from DFO-LS while still achieving strong practical performance. In particular, DFBGN with a full interpolation set achieves similar performance to DFO-LS at all accuracy levels, when measured on budget. DFBGN with a full interpolation set has a similar per-iteration linear algebra cost to DFO-LS, but uses more evaluations per iteration. As a result, when tested on medium- and large-scale problems, DFBGN with a full interpolation set is able to achieve a greater objective reduction than DFO-LS when measured by runtime. DFBGN with a small interpolation set has a much lower linear algebra cost per iteration, and can outperform DFO-LS for low accuracy levels. The reduction in linear algebra cost is more profound when considering DFO-LS with reduced initialisation cost, due to the extra expense of making the Jacobian full rank. However, the benefit of DFBGN (for small interpolation sets) is offset by lower performance for high accuracy levels, when measured on evaluations.

Another advantage of DFBGN is that it can make progress with very small budgets—which is more important for large-scale problems. We find that DFBGN with small interpolation sets can make good progress with small interpolation sets in settings where the given budget does not allow DFO-LS to construct its first model. This decrease is similar to DFO-LS with reduced initialisation cost, but with substantially reduced runtime requirements.

Future Directions There are numerous potential directions for future work in this area, both in terms of algorithm design and theoretical analysis.

Firstly, there are several aspects of the DFO-LS theory (Chapter 4) where there may be scope for further development. For instance, the safety and criticality phases serve largely the same goal—namely, to ensure the trust-region radius is not too large compared to the model gradient. Therefore, it is likely that convergence of DFO-LS could be established without the criticality phase, which would align the theory more closely with the implementation. Also, proving local convergence rates of DFO-LS for both zero- and nonzero residual problems (similar to [239] for DFLS) would be helpful to better understand the benefits of linear residual models. There would also be a benefit to understanding the theoretical properties

of the multiple restarts framework, to better grasp their usefulness on noisy problems, and to provide full theoretical guarantees for DFO-LS.

For Py-BOBYQA, theoretical results for multiple restarts, or based on excluding the criticality phase, would also be relevant. The ability of Py-BOBYQA to escape local minima could also be extended with more functionality for exploring the feasible region to improve its ability to find global minima, rather than just escaping local minima (similar to [220] for direct search, [4] for finding multiple solutions to nonlinear systems, and [131] for finding multiple local minima).

There are still several areas where we plan to extend DFBGN. Firstly, the use of adaptive interpolation set sizes should be further considered, to see if any alternative approaches can show a practical benefit over fixed sizes (see Section 7.3.2). In addition, the performance of DFBGN should be examined in the case of noisy objective evaluations, to see if the subspace approach maintains its usefulness in this setting. This would lead to an examination of how to best implement a multiple restarts mechanism (as discussed in Section 7.4). It would also be useful to extend the subspace approach to general objective problems. Here, the difficulty lies in switching from linear to quadratic interpolation, and so we would no longer have the same number of interpolation directions as subspace dimensions. In both cases, a theoretical analysis of the resulting algorithm is an important step. Beyond this, for nonlinear least-squares problems, one may further reduce the problem size by sketching (i.e. by reducing the number of residuals for which models are constructed at each iteration). This is another route through which the scalability of least-squares DFO methods could be improved. Another approach would be to have a method able to exploit a known Jacobian sparsity structure, similar to the approach considered in [52].

There are also several areas where the algorithmic framework of DFO-LS, Py-BOBYQA and DFBGN could be extended. For instance, it could be of practical interest to allow for regularisation terms—typically with known structure—in the objective, or to handle more general constraints than simple bounds. For constrained problems, this leads to several possible regimes, depending on whether derivatives are available for either the objective or the constraints, or if a subset of the constraints have derivatives. Other regimes of potential interest that have received relatively little attention for model-based DFO include multiobjective optimisation, and the setting where parallel objective evaluations may allow for more information to be included at every iteration (but perhaps not as many as required for implicit filtering). That DFBGN with a full interpolation set performs well compared to DFO-LS is evidence that there is room for progress in this latter regime.

We expect that the development of efficient model-based DFO methods will grow ever-more-important, corresponding to the ubiquitous need to optimise expensive and noisy

functions. Ongoing work to improve the scalability of these methods will help to extend their use into the increasingly-relevant regime of large-scale nonlinear optimisation. This, coupled with the development of model-based DFO methods for more classes of problems and practical scenarios (such as multiobjective or parallelisable problems), will help to further ensure the applicability of DFO and its position in the stable of essential optimisation techniques.

Bibliography

[1] M. A. Abramson, L. Frimannslund, and T. Steihaug, *A subclass of generating set search with convergence to second-order stationary points*, Optimization Methods and Software, 29 (2014), pp. 900–918.

[2] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, *A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems*, Journal of Global Optimization, 31 (2005), pp. 635–672.

[3] M. Alzantot, Y. Sharma, S. Chakraborty, and M. B. Srivastava, *GenAttack: Practical black-box attacks with gradient-free optimization*, arXiv preprint arXiv:1805.11090, (2018).

[4] Y. Aoki, K. Hayami, H. De Sterck, and A. Konagaya, *Cluster Newton method for sampling multiple solutions of underdetermined inverse problems: Application to a parameter identification problem in pharmacokinetics*, SIAM Journal on Scientific Computing, 36 (2014), pp. B14–B44.

[5] M. B. Arouxét, N. Echebest, and E. A. Pilotta, *Active-set strategy in Powell's method for optimization without derivatives*, Computational and Applied Mathematics, 30 (2011), pp. 171–196.

[6] W. Arter, A. Osojnik, C. Cartis, G. Madho, C. Jones, and S. Tobias, *Data assimilation approach to analysing systems of ordinary differential equations*, in 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018, pp. 1–5.

[7] C. Audet, *A survey on direct search methods for blackbox optimization and their applications*, in Mathematics Without Boundaries, P. M. Pardalos and T. M. Rassias, eds., Springer-Verlag, New York, 2014, pp. 31–56.

[8] C. Audet, V. Béchard, and S. Le Digabel, *Nonsmooth optimization through Mesh Adaptive Direct search and Variable Neighborhood Search*, Journal of Global Optimization, 41 (2008), pp. 299–318.

[9] C. Audet and J. E. Dennis Jr., *Analysis of generalized pattern searches*, SIAM Journal on Optimization, 13 (2003), pp. 889–903.

[10] ——, *Mesh adaptive direct search algorithms for constrained optimization*, SIAM Journal on Optimization, 17 (2006), pp. 188–217.

[11] C. Audet, S. L. Digabel, C. Tribes, and V. R. Montplaisir, *The NOMAD project*. `https://www.gerad.ca/nomad/`, 2011.

[12] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, Cham, Switzerland, 2017.

[13] C. Audet, S. Le Digabel, and M. Peyrega, *A derivative-free trust-region augmented Lagrangian algorithm*, Tech. Rep. G-2016-53, HEC Montréal, 2016.

[14] C. Audet and D. Orban, *Finding optimal algorithmic parameters using derivative-free optimization*, SIAM Journal on Optimization, 17 (2006), pp. 642–664.

[15] C. Audet and C. Tribes, *Mesh-based Nelder–Mead algorithm for inequality constrained optimization*, Computational Optimization and Applications, 71 (2018), pp. 331–352.

[16] F. Augustin and Y. M. Marzouk, *NOWPAC: A provably convergent derivative-free nonlinear optimizer with path-augmented constraints*, arXiv preprint arXiv:1403.1931, (2014).

[17] ——, *A trust-region method for derivative-free nonlinear constrained stochastic optimization*, arXiv preprint arXiv:1703.04156, (2017).

[18] F. Augustin and F. Menhorn, *(S)NOWPAC*. `https://bitbucket.org/fmaugust/nowpac`, 2018.

[19] AutoML, *pySMAC*. `https://github.com/automl/pysmac`, 2018.

[20] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Web page*. `http://www.mcs.anl.gov/petsc`, 2018.

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

[22] ——, *Convergence of trust-region methods based on probabilistic models*, SIAM Journal on Optimization, 24 (2014), pp. 1238–1264.

[23] R. G. Begiato, A. L. Custódio, and M. A. Gomes-Ruggiero, *A global hybrid derivative-free method for large-scale systems of nonlinear equations*, tech. rep., available on Optimization Online, 2018.

[24] S. Bellavia, S. Gratton, and E. Riccietti, *A Levenberg-Marquardt method for large nonlinear least squares problems with noisy functions and gradients*, Numerische Mathematik, to appear (2019).

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

[26] E. Bergou, Y. Diouane, V. Kungurtsev, and C. W. Royer, *A stochastic Levenberg-Marquardt method using random models with application to data assimilation*, arXiv preprint arXiv:1807.02176, (2018).

[27] E. Bergou, S. Gratton, and L. N. Vicente, *Levenberg–Marquardt methods based on probabilistic gradient models and inexact subproblem solution, with application to data assimilation*, SIAM/ASA Journal on Uncertainty Quantification, 4 (2016), pp. 924–951.

[28] J. Bergstra, D. Yamins, and D. Cox, *Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures*, in Proceedings of the 30th International Conference on Machine Learning, S. Dasgupta and D. McAllester, eds., vol. 28 of Proceedings of Machine Learning Research, Atlanta, 2013, PMLR, pp. 115–123.

[29] J. Bergstra, D. L. K. Yamins, and D. D. Cox, *Hyperopt: Distributed asynchronous hyper-parameter optimization*. `https://github.com/hyperopt/hyperopt`, 2018.

[30] S. C. Billups, J. W. Larson, and P. Graf, *Derivative-free optimization of expensive functions with computational error using weighted regression*, SIAM Journal on Optimization, 23 (2013), pp. 27–53.

[31] J. Blanchet, C. Cartis, M. Menickelly, and K. Scheinberg, *Convergence rate analysis of a stochastic trust region method for nonconvex optimization*, INFORMS Journal on Optimization, 1 (2019), pp. 92–119.

[32] A.-L. Both, H. Hisken, J.-J. Rückmann, and T. Steihaug, *Surrogate-based model parameter optimization based on gas explosion experimental data*, Engineering Optimization, 51 (2019), pp. 301–316.

[33] R. Brekelmans, L. Driessen, H. Hamers, and D. Den Hertog, *Constrained optimization involving expensive function evaluations: A sequential approach*, European Journal of Operational Research, 160 (2005), pp. 121–138.

[34] R. P. Brent, *Algorithms for minimization without derivatives*, Dover Publications, Mineola, New York, 2002.

[35] E. Brochu, V. M. Cora, and N. D. Freitas, *A tutorial on Bayesian optimization of expensive cost functions , with application to active user modeling and hierarchical reinforcement learning*, arXiv preprint arXiv:1012.2599, (2010).

[36] K. M. Brown and J. E. Dennis Jr., *Derivative free analogues of the Levenberg-Marquardt and Gauss algorithms for nonlinear least squares approximation*, Numerische Mathematik, 18 (1971), pp. 289–297.

[37] C. G. Broyden, *A class of methods for solving nonlinear simultaneous equations*, Mathematics of Computation, 19 (1965), pp. 577–593.

[38] R. L. Burden and J. D. Faires, *Numerical Analysis*, Brooks/Cole, Cengage Learning, Boston, 9th ed., 2011.

[39] C. Cartis, J. Fiala, B. Marteau, and L. Roberts, *Improving the flexibility and robustness of model-based derivative-free optimization solvers*, ACM Transactions on Mathematical Software, 45 (2019), pp. 32:1–32:41.

[40] C. Cartis and J. Fowkes, *A block-coordinate Gauss-Newton method for nonlinear least squares*, in preparation, (2019).

[41] C. Cartis, N. I. M. Gould, and P. L. Toint, *On the complexity of steepest descent, Newton's and regularized Newton's methods for nonconvex unconstrained optimization problems*, SIAM Journal on Optimization, 20 (2010), pp. 2833–2852.

[42] ——, *Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results*, Mathematical Programming, 127 (2011), pp. 245–295.

[43] ——, *Adaptive cubic regularisation methods for unconstrained optimization. Part II: worst-case function- and derivative-evaluation complexity*, Mathematical Programming, 130 (2011), pp. 295–319.

[44] ——, *On the evaluation complexity of composite function minimization with applications to nonconvex nonlinear programming*, SIAM Journal on Optimization, 21 (2011), pp. 1721–1739.

[45] ——, *Complexity bounds for second-order optimality in unconstrained optimization*, Journal of Complexity, 28 (2012), pp. 93–108.

[46] ——, *On the oracle complexity of first-order and derivative-free algorithms for smooth nonconvex minimization*, SIAM Journal on Optimization, 22 (2012), pp. 66–86.

[47] C. Cartis and L. Roberts, *A derivative-free Gauss-Newton method*, Mathematical Programming Computation, (2019).

[48] C. Cartis and K. Scheinberg, *Global convergence rate analysis of unconstrained optimization methods based on probabilistic models*, Mathematical Programming, 169 (2018), pp. 337–375.

[49] C. Cartis, O. Sheridan-Methven, and L. Roberts, *Escaping local minima with derivative-free methods: a numerical investigation*, arXiv preprint arXiv:1812.11343, (2019).

[50] R. Chen, M. Menickelly, and K. Scheinberg, *Stochastic optimization using a trust-region method and random models*, Mathematical Programming, 169 (2018), pp. 447–487.

[51] A. Cohen, M. A. Davenport, and D. Leviatan, *On the stability and accuracy of least squares approximations*, Foundations of Computational Mathematics, 13 (2013), pp. 819–834.

[52] B. Colson and P. L. Toint, *Optimizing partially separable functions without derivatives*, Optimization Methods and Software, 20 (2005), pp. 493–508.

[53] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, vol. 1 of MPS-SIAM Series on Optimization, MPS/SIAM, Philadelphia, 2000.

[54] A. R. Conn and S. Le Digabel, *Use of quadratic models with mesh-adaptive direct search for constrained black box optimization*, Optimization Methods and Software, 28 (2013), pp. 139–158.

[55] A. R. Conn, K. Scheinberg, and P. L. Toint, *On the convergence of derivative-free methods for unconstrained optimization*, in Approximation Theory and Optimization: Tributes to M. J. D. Powell, M. D. Buhmann and A. Iserles, eds., Cambridge University Press, Cambridge, 1997, pp. 83–108.

[56] ——, *Recent progress in unconstrained nonlinear optimization without derivatives*, Mathematical Programming, 79 (1997), pp. 397–414.

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

[58] ——, *Geometry of sample sets in derivative-free optimization: Polynomial regression and underdetermined interpolation*, IMA Journal of Numerical Analysis, 28 (2008), pp. 721–748.

[59] ——, *Global convergence of general derivative-free trust-region algorithms to first- and second-order critical points*, SIAM Journal on Optimization, 20 (2009), pp. 387–415.

[60] ——, *Introduction to Derivative-Free Optimization*, vol. 8 of MPS-SIAM Series on Optimization, MPS/SIAM, Philadelphia, 2009.

[61] A. R. Conn and P. L. Toint, *An algorithm using quadratic interpolation for unconstrained derivative free optimization*, in Nonlinear Optimization and Applications, G. Di Pillo and F. Gianessi, eds., Plenum Publishing, New York, 1996, pp. 27–47.

[62] A. R. Conn and L. N. Vicente, *Bilevel derivative-free optimization and its application to robust optimization*, Optimization Methods and Software, 27 (2012), pp. 561–577.

[63] I. Coope and R. Tappenden, *Efficient calculation of regular simplex gradients*, Computational Optimization and Applications, 72 (2019), pp. 561–588.

[64] A. L. Custódio, J. F. A. Madeira, A. I. F. Vaz, and L. N. Vicente, *Direct multisearch for multiobjective optimization*, SIAM Journal on Optimization, 21 (2011), pp. 1109–1140.

[65] A. L. Custódio, H. Rocha, and L. N. Vicente, *Incorporating minimum Frobenius norm models in direct search*, Computational Optimization and Applications, 46 (2010), pp. 265–278.

[66] A. L. Custódio, K. Scheinberg, and L. N. Vicente, *Methodologies and software for derivative-free optimization*, in Advances and Trends in Optimization with Engineering Applications, T. Terlaky, M. F. Anjos, and S. Ahmed, eds., vol. 24 of MOS-SIAM Book Series on Optimization, MOS/SIAM, Philadelphia, 2017, pp. 495–506.

[67] A. L. Custódio and L. N. Vicente, *Using sampling and simplex derivatives in pattern search methods*, SIAM Journal on Optimization, 18 (2007), pp. 537–555.

[68] ——, *SID-PSM*. `http://www.mat.uc.pt/sid-psm/`, 2014.

[69] J. W. Demmel, *Applied Numerical Linear Algebra*, Other Titles in Applied Mathematics, SIAM, Philadelphia, 1997.

[70] G. Deng and M. C. Ferris, *Adaptation of the UOBYQA algorithm for noisy functions*, in Proceedings of the 2006 Winter Simulation Conference, L. F. Peronne, F. P. Weiland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds., 2006, pp. 312–319.

[71] ——, *Variable-number sample-path optimization*, Mathematical Programming, 117 (2009), pp. 81–109.

[72] J. E. Dennis Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Classics in Applied Mathematics, SIAM, Philadelphia, 1996.

[73] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, 91 (2002), pp. 201–213.

[74] M. L. Eaton, *Multivariate Statistics: A Vector Space Approach*, vol. 53 of Lecture Notes–Monograph Series, Institute of Mathematical Statistics, Beachwood, Ohio, 2007.

[75] C. Elster and A. Neumaier, *A method of trust region type for minimizing noisy functions*, Computing, 58 (1997), pp. 31–46.

[76] D. Eriksson, D. Bindel, and C. Shoemaker, *Surrogate optimization toolbox (pySOT)*. `https://github.com/dme65/pySOT`, 2015.

[77] G. Fasano, G. Liuzzi, S. Lucidi, and F. Rinaldi, *A linesearch-based derivative-free approach for nonsmooth constrained optimization*, SIAM Journal on Optimization, 24 (2014), pp. 959–992.

[78] G. Fasano, J. L. Morales, and J. Nocedal, *On the geometry phase in model-based algorithms for derivative-free optimization*, Optimization Methods and Software, 24 (2009), pp. 145–154.

[79] P. S. Ferreira, E. W. Karas, M. Sachine, and F. N. C. Sobral, *Global convergence of a derivative-free inexact restoration filter algorithm for nonlinear programming*, Optimization, 66 (2017), pp. 271–292.

[80] R. Flamary, A. Rakotomamonjy, and G. Gasso, *Importance sampling strategy for non-convex randomized block-coordinate descent*, in IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, Cancun, Mexico, 2015.

[81] R. Fletcher, *Practical Methods of Optimization*, Wiley, Chichester, UK, 2nd ed., 2008.

[82] J. Fowkes, *Bayesian numerical analysis: global optimization and other applications*, PhD thesis, University of Oxford, 2011.

[83] R. Garmanjani, D. Júdice, and L. N. Vicente, *Trust-region methods without using derivatives: Worst case complexity and the nonsmooth case*, SIAM Journal on Optimization, 26 (2016), pp. 1987–2011.

[84] H. Ghanbari and K. Scheinberg, *Black-box optimization in machine learning with trust region based derivative free algorithm*, arXiv preprint arXiv:1703.06925, (2017).

[85] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 3rd ed., 1996.

[86] N. I. M. Gould, D. Orban, and P. L. Toint, *CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization*, Computational Optimization and Applications, 60 (2015), pp. 545–557.

[87] N. I. M. Gould, M. Porcelli, and P. L. Toint, *Updating the regularization parameter in the adaptive cubic regularization algorithm*, Computational Optimization and Applications, 53 (2012), pp. 1–22.

[88] R. Gower, D. Goldfarb, and P. Richtarik, *Stochastic block BFGS: Squeezing more curvature out of data*, in Proceedings of The 33rd International Conference on Machine Learning, M. F. Balcan and K. Q. Weinberger, eds., vol. 48 of Proceedings of Machine Learning Research, New York, 2016, PMLR, pp. 1869–1878.

[89] R. M. Gower and P. Richtárik, *Randomized iterative methods for linear systems*, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 1660–1690.

[90] G. N. Grapiglia, J. Yuan, and Y.-x. Yuan, *A derivative-free trust-region algorithm for composite nonsmooth optimization*, Computational and Applied Mathematics, 35 (2016), pp. 475–499.

[91] S. Gratton, C. W. Royer, and L. N. Vicente, *A second-order globally convergent direct-search method and its worst-case complexity*, Optimization, 65 (2016), pp. 1105–1128.

[92] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang, *Direct search based on probabilistic descent*, SIAM Journal on Optimization, 25 (2015), pp. 1515–1541.

[93] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang, *Complexity and global rates of trust-region methods based on probabilistic models*, IMA Journal of Numerical Analysis, 38 (2017), pp. 1579–1597.

[94] ——, *Direct search based on probabilistic feasible descent for bound and linearly constrained problems*, Computational Optimization and Applications, 72 (2019), pp. 525–559.

[95] S. Gratton, P. L. Toint, and A. Tröltzsch, *An active-set trust-region method for derivative-free nonlinear bound-constrained optimization*, Optimization Methods and Software, 26 (2011), pp. 873–894.

[96] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia, 2nd ed., 2008.

[97] M. Grippo and M. Sciandrone, *Nonmonotone derivative-free methods for nonlinear equations*, Computational Optimization and Applications, 37 (2007), pp. 297–328.

[98] L. Han and G. Liu, *On the convergence of the UOBYQA method*, Journal of Applied Mathematics and Computing, 16 (2004), pp. 125–142.

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

[100] ——, *pycma*. `https://github.com/CMA-ES/pycma`, 2018.

[101] N. Hansen and A. Ostermeier, *Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation*, in Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, 1996, pp. 312–317.

[102] W. Hare and G. Jarry-Bolduc, *Calculus identities for generalized simplex gradients: Rules and applications*, tech. rep., University of British Columbia, 2018.

[103] W. Hare, J. Loeppky, and S. Xie, *Methods to compare expensive stochastic optimization algorithms with random restarts*, Journal of Global Optimization, 72 (2018), pp. 781–801.

[104] W. Hare, C. Planiden, and C. Sagastizábal, *A derivative-free $\mathcal{VU}$-algorithm for convex finite-max problems*, arXiv preprint arXiv:1903.11184, (2019).

[105] W. L. Hare and Y. Lucet, *Derivative-free optimization via proximal point methods*, Journal of Optimization Theory and Applications, 160 (2014), pp. 204–220.

[106] X. Huang and D. Zhu, *An interior affine scaling cubic regularization algorithm for derivative-free optimization subject to bound constraints*, Journal of Computational and Applied Mathematics, 321 (2017), pp. 108–127.

[107] F. Hutter, H. H. Hoos, and K. Leyton-Brown, *Sequential model-based optimization for general algorithm configuration*, in International Conference on Learning and Intelligent Optimization, Rome, 2011, pp. 507–223.

[108] W. Huyer and A. Neumaier, *Global optimization by multilevel coordinate search*, Journal of Global Optimization, 14 (1999), pp. 331–355.

[109] ——, *SNOBFIT – stable noisy optimization by branch and fit*, ACM Transactions on Mathematical Software, 35 (2008), pp. 1–25.

[110] J. M. Gablonsky, *pydirect*. `https://code.google.com/archive/p/pydirect`, 2012.

[111] K. G. Jamieson, R. D. Nowak, and B. Recht, *Query complexity of derivative-free optimization*, in Advances in Neural Information Processing Systems, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012, pp. 2672–2680.

[112] D. R. Jones, *A taxonomy of global optimization methods based on response surfaces*, Journal of Global Optimization, 21 (2001), pp. 345–383.

[113] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, *Lipschitzian optimization without the Lipschitz constant*, Journal of Optimization Theory and Applications, 79 (1993), pp. 157–181.

[114] D. Júdice, *Trust-Region Methods without using Derivatives: Worst-Case Complexity and the Non-Smooth Case*, PhD thesis, University of Coimbra, 2015.

[115] A. Kannan and S. M. Wild, *Obtaining quadratic models of noisy functions*, Tech. Rep. ANL/MCS-P1975-1111, Argonne National Laboratory, 2012.

[116] C. T. Kelley, *Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition*, SIAM Journal on Optimization, 10 (1999), pp. 43–55.

[117] ——, *Iterative Methods for Optimization*, vol. 18 of Frontiers in Applied Mathematics, SIAM, Philadelphia, 1999.

[118] ——, *Implicit filtering and nonlinear least squares problems*, in CSMO 2001: System Modeling and Optimization XX, E. W. Sachs and R. Tichatschke, eds., Springer US, 2003, pp. 71–90.

[119] ——, *Implicit Filtering*, Software, Environments and Tools, SIAM, Philadelphia, 2011.

[120] K. A. Khan, J. Larson, and S. M. Wild, *Manifold sampling for optimization of nonconvex functions that are piecewise linear compositions of smooth components*, SIAM Journal on Optimization, 28 (2018), pp. 3001–3024.

[121] T. G. Kolda, R. M. Lewis, and V. Torczon, *Optimization by direct search: New perspectives on some classical and modern methods*, SIAM Review, 45 (2003), pp. 385–482.

[122] W. La Cruz, *A projected derivative-free algorithm for nonlinear equations with convex constraints*, Optimization Methods and Software, 29 (2014), pp. 24–41.

[123] W. La Cruz, J. M. Martínez, and M. Raydan, *Spectral residual method without gradient information for solving large-scale nonlinear systems of equations*, Mathematics of Computation, 75 (2006), pp. 1429–1448.

[124] W. La Cruz and M. Raydan, *Nonmonotone spectral methods for large-scale nonlinear systems*, Optimization Methods and Software, 18 (2003), pp. 583–599.

[125] J. C. Lagarias, B. Poonen, and M. H. Wright, *Convergence of the restricted Nelder-Mead algorithm in two dimensions*, SIAM Journal on Optimization, 22 (2012), pp. 501–532.

[126] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, *Convergence properties of the Nelder-Mead simplex method in low dimensions*, SIAM Journal on Optimization, 9 (1998), pp. 112–147.

[127] J. W. Larson and S. C. Billups, *Stochastic derivative-free optimization using a trust region framework*, Computational Optimization and Applications, 64 (2016), pp. 619–645.

[128] J. W. Larson, M. Menickelly, and S. M. Wild, *Manifold sampling for ℓ_1 nonconvex optimization*, SIAM Journal on Optimization, 26 (2016), pp. 2540–2563.

[129] ——, *Derivative-free optimization methods*, Acta Numerica, 28 (2019), pp. 287–404.

[130] J. W. Larson and S. M. Wild, *Non-intrusive termination of noisy optimization*, Optimization Methods and Software, 28 (2013), pp. 993–1011.

[131] ——, *Asynchronously parallel optimization solver for finding multiple minima*, Mathematical Programming Computation, 10 (2018), pp. 303–332.

[132] S. Le Digabel, *Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm*, ACM Transactions on Mathematical Software, 37 (2011), pp. 1–15.

[133] H. A. Le Thi, A. I. F. Vaz, and L. N. Vicente, *Optimizing radial basis functions by d.c. programming and its use in direct search for global derivative-free optimization*, TOP, 20 (2012), pp. 190–214.

[134] D.-h. Li and M. Fukushima, *A derivative-free line search and global convergence of Broyden-like method for nonlinear equations*, Optimization Methods and Software, 13 (2000), pp. 181–201.

[135] M. Li, *A derivative-free PRP method for solving large- scale nonlinear systems of equations and its global convergence*, Optimization Methods and Software, 29 (2014), pp. 503–514.

[136] Q. Li and D.-H. Li, *A class of derivative-free methods for large-scale nonlinear monotone equations*, IMA Journal of Numerical Analysis, 31 (2011), pp. 1625–1635.

[137] G. Liuzzi, *DFL – a derivative-free library.* `http://www.iasi.cnr.it/~liuzzi/DFL/index.php`, 2019.

[138] G. Liuzzi, S. Lucidi, and F. Rinaldi, *Derivative-free methods for mixed-integer constrained optimization problems*, Journal of Optimization Theory and Applications, 164 (2015), pp. 933–965.

[139] G. Liuzzi, S. Lucidi, F. Rinaldi, and L. N. Vicente, *Trust-region methods for the derivative-free optimization of nonsmooth black-box functions*, tech. rep., available on Optimization Online, 2019.

[140] M. Locatelli and F. Schoen, *Global Optimization: Theory, Algorithms, and Applications*, vol. 15 of MOS-SIAM Series on Optimization, MOS/SIAM, Philadelphia, 2013.

[141] Z. Lu and L. Xiao, *A randomized nonmonotone block proximal gradient method for a class of structured nonlinear programming*, SIAM Journal on Numerical Analysis, 55 (2017), pp. 2930–2955.

[142] S. Lucidi and M. Sciandrone, *On the global convergence of derivative-free methods for unconstrained optimization*, SIAM Journal on Optimization, 13 (2002), pp. 97–116.

[143] L. Lukšan, *Hybrid methods for large sparse nonlinear least squares*, Journal of Optimization Theory and Applications, 89 (1996), pp. 575–595.

[144] L. Lukšan, C. Matonoha, and J. Vlček, *Modified CUTE problems for sparse unconstrained optimization*, Tech. Rep. 1081, Institute of Computer Science, Academy of Sciences of the Czech Republic, 2010.

[145] L. Lukšan, C. Matonoha, and J. Vlček, *Hybrid methods for nonlinear least squares problems*, Tech. Rep. 1246, Institute of Computer Science, Academy of Sciences of the Czech Republic, 2019.

[146] L. Lukšan and J. Vlček, *New quasi-Newton method for solving systems of nonlinear equations*, Applications of Mathematics, 62 (2017), pp. 121–134.

[147] M. Marazzi and J. Nocedal, *Wedge trust region methods for derivative free optimization*, Mathematical Programming, 91 (2002), pp. 289–305.

[148] L. Marini, B. Morini, and M. Porcelli, *Quasi-Newton methods for constrained nonlinear systems: complexity analysis and applications*, Computational Optimization and Applications, 71 (2018), pp. 147–170.

[149] K. I. M. McKinnon, *Convergence of the Nelder-Mead simplex method to a nonstationary point*, SIAM Journal on Optimization, 9 (1998), pp. 148–158.

[150] M. Menickelly and S. M. Wild, *Derivative-free robust optimization by outer approximations*, Mathematical Programming, (2019).

[151] C. D. Meyer, Jr., *Generalized inversion of modified matrices*, SIAM Journal on Applied Mathematics, 24 (1973), pp. 315–323.

[152] H. Mohammed and S. A. Santos, *A structured diagonal Hessian approximation method with evaluation complexity analysis for nonlinear least squares*, Computational and Applied Mathematics, 37 (2018), pp. 6619–6653.

[153] K. Mombaur, A. Truong, and J.-P. Laumond, *From human to humanoid locomotion—an inverse optimal control approach*, Autonomous Robots, 28 (2010), pp. 369–383.

[154] J. J. Moré, *The Levenberg-Marquardt algorithm: Implementation and theory*, in Numerical analysis: Proceedings of the Biennial Conference Held at Dundee, June 28–July 1, 1977, G. A. Watson, ed., Springer-Verlag, Berlin, 1977, pp. 105–116.

[155] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing unconstrained optimization software*, ACM Transactions on Mathematical Software, 7 (1981), pp. 17–41.

[156] J. J. Moré and J. A. Trangenstein, *On the global convergence of Broyden's method*, Mathematics of Computation, 30 (1976), pp. 523–540.

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

[158] ——, *Estimating computational noise*, SIAM Journal on Scientific Computing, 33 (2011), pp. 1292–1314.

[159] ——, *Estimating derivatives of noisy simulations*, ACM Transactions on Mathematical Software, 38 (2012), pp. 1–21.

[160] B. Morini, M. Porcelli, and P. L. Toint, *Approximate norm descent methods for constrained nonlinear systems*, Mathematics of Computation, 87 (2018), pp. 1327–1351.

[161] R. J. Muirhead, *Aspects of Multivariate Statistical Theory*, Wiley, New York, 1982.

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

[163] E. Newby and M. M. Ali, *A trust-region-based derivative free algorithm for mixed integer programming*, Computational Optimization and Applications, 60 (2015), pp. 199–229.

[164] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, 2nd ed., 2006.

[165] R. Oeuvray, *Trust-region methods based on radial basis functions with application to biomedical imaging*, PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2005.

[166] R. Oeuvray and M. Bierlaire, *A new derivative-free algorithm for the medical image registration problem*, International Journal of Modelling and Simulation, 72 (2007), pp. 115–124.

[167] ——, *BOOSTERS: A derivative-free algorithm based on radial basis functions*, International Journal of Modelling and Simulation, 29 (2009), pp. 26–36.

[168] C. Paquette and K. Scheinberg, *A stochastic line search method with convergence rate analysis*, arXiv preprint arXiv:1807.07994, (2018).

[169] G. Peckham, *A new method for minimising a sum of squares without calculating gradients*, The Computer Journal, 13 (1970), pp. 418–420.

[170] M. Pilanci and M. J. Wainwright, *Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence*, SIAM Journal on Optimization, 27 (2017), pp. 205–245.

[171] M. Porcelli and P. L. Toint, *BFO, a trainable derivative-free brute force optimizer for nonlinear bound-constrained optimization and equilibrium computations*, ACM Transactions on Mathematical Software, 44 (2017), pp. 6:1–6:25.

[172] M. J. D. Powell, *An efficient method for finding the minimum of a function of several variables without calculating derivatives*, The Computer Journal, 7 (1964), pp. 155–162.

[173] ——, *A method for minimizing a sum of squares of non-linear functions without calculating derivatives*, The Computer Journal, 7 (1965), pp. 303–307.

[174] ——, *A FORTRAN subroutine for solving systems of nonlinear algebraic equations*, in Numerical Algorithms for Nonlinear Algebraic Equations, P. Rabinowitz, ed., Gordon and Breach, London, 1970, pp. 115–161.

[175] ——, *A hybrid method for nonlinear equations*, in Numerical Algorithms for Nonlinear Algebraic Equations, P. Raboniwitz, ed., Gordon and Breach, London, 1970, pp. 87–114.

[176] ——, *On search directions for minimization algorithms*, Mathematical Programming, 4 (1973), pp. 193–201.

[177] ——, *A direct search optimization method that models the objective and constraint functions by linear interpolation*, in Advances in Optimization and Numerical Analysis, S. Gomez and J.-P. Hennart, eds., Kluwer Academic Publishers, Dordrecht, 1994, pp. 51–67.

[178] ——, *Direct search algorithms for optimization calculations*, Acta Numerica, 7 (1998), pp. 287–336.

[179] ——, *On the Lagrange functions of quadratic models that are defined by interpolation*, Optimization Methods and Software, 16 (2001), pp. 289–309.

[180] ——, *UOBYQA: Unconstrained optimization by quadratic approximation*, Mathematical Programming, 92 (2002), pp. 555–582.

[181] ——, *On trust region methods for unconstrained minimization without derivatives*, Mathematical Programming, 97 (2003), pp. 605–623.

[182] ——, *Least Frobenius norm updating of quadratic models that satisfy interpolation conditions*, Mathematical Programming, 100 (2004), pp. 183–215.

[183] ——, *On the use of quadratic models in unconstrained minimization without derivatives*, Optimization Methods and Software, 19 (2004), pp. 399–411.

[184] ——, *The NEWUOA software for unconstrained optimization without derivatives*, in Large-Scale Nonlinear Optimization, G. Di Pillo and M. Roma, eds., vol. 83 of Nonconvex Optimization and Its Applications, Springer, Boston, 2006, pp. 255–297.

[185] ——, *Developments of NEWUOA for minimization without derivatives*, IMA Journal of Numerical Analysis, 28 (2008), pp. 649–664.

[186] ——, *The BOBYQA algorithm for bound constrained optimization without derivatives*, Tech. Rep. DAMTP 2009/NA06, University of Cambridge, 2009.

[187] ——, *Beyond symmetric Broyden for updating quadratic models in minimization without derivatives*, Mathematical Programming, 138 (2013), pp. 475–500.

[188] ——, *On fast trust region methods for quadratic models with linear constraints*, Mathematical Programming Computation, 7 (2015), pp. 237–267.

[189] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1986.

[190] H. Qian, Y.-q. Hu, and Y. Yu, *Derivative-free optimization of high-dimensional non-convex functions by sequential random embeddings*, in Proceedings of the 25th International Joint Conference on Artificial Intelligence, S. Kambhampati, ed., New York, 2016, AAAI Press, pp. 1946–1952.

[191] M. L. Ralston and R. I. Jennrich, *Dud, a derivative-free algorithm for nonlinear least squares*, Technometrics, 20 (1978), pp. 7–14.

[192] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, Adaptive Computation and Machine Learning, The MIT Press, Cambridge, 2006.

[193] R. G. Regis, *The calculus of simplex gradients*, Optimization Letters, 9 (2015), pp. 845–865.

[194] R. G. Regis and C. A. Shoemaker, *A stochastic radial basis function method for the global optimization of expensive functions*, INFORMS Journal on Computing, 19 (2007), pp. 497–509.

[195] ——, *Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization*, Engineering Optimization, 45 (2013), pp. 529–555.

[196] R. G. Regis and S. M. Wild, *CONORBIT: constrained optimization by radial basis function interpolation in trust regions*, Optimization Methods and Software, 32 (2017), pp. 552–580.

[197] P. Richtárik and M. Takáč, *Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function*, Mathematical Programming, 144 (2014), pp. 1–38.

[198] E. S. Riis, M. J. Ehrhardt, G. R. W. Quispel, and C.-B. Schönlieb, *A geometric integration approach to nonsmooth, nonconvex optimisation*, arXiv preprint arXiv:1807.07554, (2018).

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

[200] L. Roberts, *Derivative-free optimisation for data fitting*, tech. rep., University of Oxford, 2016.

[201] F. Roosta-Khorasani and M. W. Mahoney, *Sub-sampled Newton methods*, Mathematical Programming, 174 (2019), pp. 293–326.

[202] K. Scheinberg, *DFO*. `https://projects.coin-or.org/Dfo`, 2006.

[203] K. Scheinberg and P. L. Toint, *Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization*, SIAM Journal on Optimization, 20 (2010), pp. 3512–3532.

[204] B. Scherrer, A. Schwartzman, M. Taquet, M. Sahin, S. P. Prabhu, and S. K. Warfield, *Characterizing brain tissue by assessment of the distribution of anisotropic microstructural environments in diffusion-compartment imaging (DIAMOND)*, Magnetic Resonance in Medicine, 76 (2016), pp. 963–977.

[205] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. D. Freitas, *Taking the human out of the loop: A review of Bayesian optimization*, Proceedings of the IEEE, 104 (2016), pp. 148–175.

[206] S. Shashaani, F. S. Hashemi, and R. Pasupathy, *ASTRO-DF: A class of adaptive sampling trust-region algorithms for derivative-free stochastic optimization*, SIAM Journal on Optimization, 28 (2018), pp. 3145–3176.

[207] J. Snoek, H. Larochelle, and R. P. Adams, *Practical Bayesian optimization of machine learning algorithms*, in Advances in Neural Information Processing Systems, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012, pp. 2951–2959.

[208] W. Spendley, *Nonlinear least squares fitting using a modified simplex minimization method*, in Optimization: Symposium of the Institute of Mathematics and its Applications, R. Fletcher, ed., Academic Press, 1969, pp. 259–270.

[209] W. H. Swann, *A survey of non-linear optimization techniques*, FEBS Letters, 2 (1969), pp. S39–S55.

[210] J. Tanner and G. Ughi, *Model based optimisation applied to black-box attacks in deep learning*, in preparation, (2019).

[211] S. F. B. Tett, M. J. Mineter, C. Cartis, D. J. Rowlands, and P. Liu, *Can top-of-atmosphere radiation measurements constrain climate predictions? Part I: Tuning*, Journal of Climate, 26 (2013), pp. 9348–9366.

[212] S. F. B. Tett, K. Yamazaki, M. J. Mineter, C. Cartis, and N. Eizenberg, *Calibrating climate models using inverse methods: Case studies with HadAM3, HadAM3P and HadCM3*, Geoscientific Model Development, 10 (2017), pp. 3567–3589.

[213] The GPyOpt authors, *GPyOpt: A Bayesian optimization framework in Python*. `http://github.com/SheffieldML/GPyOpt`, 2016.

[214] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2013.

[215] A. Tröltzsch, *An active-set trust-region method for derivative-free nonlinear bound-constrained optimization applied to noisy aerodynamic design problems*, PhD thesis, Institut National Polytechnique de Toulouse, 2011.

[216] ——, *A sequential quadratic programming algorithm for equality-constrained optimization without derivatives*, Optimization Letters, 10 (2016), pp. 383–399.

[217] P. Tseng, *Fortified-descent simplicial search method: A general approach*, SIAM Journal on Optimization, 10 (1999), pp. 269–288.

[218] ——, *Convergence of a block coordinate descent method for nondifferentiable minimization*, Journal of Optimization Theory and Applications, 109 (2001), pp. 475–494.

[219] F. Vanden Berghen and H. Bersini, *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 (2005), pp. 157–175.

[220] A. I. F. Vaz and L. N. Vicente, *A particle swarm pattern search method for bound constrained global optimization*, Journal of Global Optimization, 39 (2007), pp. 197–219.

[221] L. N. Vicente, *Worst case complexity of direct search*, EURO Journal on Computational Optimization, 1 (2013), pp. 143–153.

[222] Y. Wang, S. S. Du, S. Balakrishnan, and A. Singh, *Stochastic zeroth-order optimization in high dimensions*, in Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, A. Storkey and F. Perez-Cruz, eds., vol. 84 of Proceedings of Machine Learning Research, PMLR, 2018, pp. 1356–1365.

[223] S. M. Wild, *MNH: A derivative-free optimization algorithm using minimal norm hessians*, in Tenth Copper Mountain Conference on Iterative Methods, 2008.

[224] ——, *Derivative-Free Optimization Algorithms for Computationally Expensive Functions*, PhD thesis, Cornell University, 2009.

[225] ——, *POUNDERS in TAO: Solving derivative-free nonlinear least-squares problems with POUNDERS*, in Advances and Trends in Optimization with Engineering Applications, T. Terlaky, M. F. Anjos, and S. Ahmed, eds., vol. 24 of MOS-SIAM Book Series on Optimization, MOS/SIAM, Philadelphia, 2017, pp. 529–539.

[226] S. M. Wild, R. G. Regis, and C. A. Shoemaker, *ORBIT: Optimization by radial basis function interpolation in trust-regions*, SIAM Journal on Scientific Computing, 30 (2008), pp. 3197–3219.

[227] S. M. Wild and C. A. Shoemaker, *Global convergence of radial basis function trust-region algorithms for derivative-free optimization*, SIAM Review, 55 (2013), pp. 349–371.

[228] D. Winfield, *Function minimization by interpolation in a data table*, IMA Journal of Applied Mathematics, 12 (1973), pp. 339–347.

[229] D. J. Woods, *An Interactive Approach for Solving Multi-Objective Optimization Problems*, PhD thesis, Rice University, 1985.

[230] M. Wright, *Direct search methods: Once scorned, now respectable*, in Numerical analysis: Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis, D. F. Griffiths and G. A. Watson, eds., Addison-Wesley, Harlow, UK, 1995, pp. 191–208.

[231] S. J. Wright, *Coordinate descent algorithms*, Mathematical Programming, 151 (2015), pp. 3–34.

[232] Y. Xie, R. Byrd, and J. Nocedal, *Analysis of the BFGS method with errors*, arXiv preprint arXiv:1901.09063, (2019).

[233] Y. Xu and W. Yin, *A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion*, SIAM Journal on Imaging Sciences, 6 (2013), pp. 1758–1789.

[234] ——, *Block stochastic gradient iteration for convex and nonconvex optimization*, SIAM Journal on Optimization, 25 (2015), pp. 1686–1716.

[235] ——, *A globally convergent algorithm for nonconvex optimization based on block coordinate update*, Journal of Scientific Computing, 72 (2017), pp. 700–734.

[236] D. Xue and W. Sun, *On convergence analysis of a derivative-free trust region algorithm for constrained optimization with separable structure*, Science China Mathematics, 57 (2013), pp. 1287–1302.

[237] Y. Yang, M. Pesavento, Z.-q. Luo, and B. Ottersten, *Inexact block coordinate descent algorithms for nonsmooth nonconvex optimization*, arXiv preprint arXiv:1905.04211, (2019).

[238] Z. Fu, *Package snobfit*. `http://reflectometry.org/danse/docs/snobfit`, 2009.

[239] H. Zhang and A. R. Conn, *On the local convergence of a derivative-free algorithm for least-squares minimization*, Computational Optimization and Applications, 51 (2012), pp. 481–507.

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

[241] Z. Zhang, *Sobolev seminorm of quadratic functions with applications to derivative-free optimization*, Mathematical Programming, 146 (2014), pp. 77–96.

[242] ——, *Software by Professor M. J. D. Powell*. `http://mat.uc.pt/~zhang/software.html`, 2017.

Appendix A

Proof of Lemma 4.2

This result extends the proof[1] of [60, Theorem 2.13] to vector-valued functions, and their composition in a least-squares objective.

Using $\mathbf{y}_0 = \mathbf{x}_k$ (as per Section 2.2.2), we may write

$$W_k := \begin{bmatrix} 1 & (\mathbf{y}_0 - \mathbf{x}_k)^\top \\ \vdots & \vdots \\ 1 & (\mathbf{y}_p - \mathbf{x}_k)^\top \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{e} & L_k \end{bmatrix}, \tag{A.1}$$

where $\mathbf{e} \in \mathbb{R}^p$ is the vector of ones and $L_k \in \mathbb{R}^{p \times n}$ has rows $(\mathbf{y}_t - \mathbf{x}_k)^\top$ for $t = 1, \ldots, p$.

We will also use scaled versions of these matrices:

$$\hat{W}_k := \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{e} & \hat{L}_k \end{bmatrix}, \quad \text{where} \quad \hat{L}_k := \begin{bmatrix} (\mathbf{y}_1 - \mathbf{x}_k)^\top / \Delta_k \\ \vdots \\ (\mathbf{y}_p - \mathbf{x}_k)^\top / \Delta_k \end{bmatrix} = L / \Delta_k. \tag{A.2}$$

Equivalently, we have

$$\hat{W}_k = W_k D_k, \quad \text{where} \quad D_k := \begin{bmatrix} 1 & \mathbf{0}^\top \\ \mathbf{0} & \Delta_k^{-1} I_{n \times n} \end{bmatrix}. \tag{A.3}$$

Our overdetermined interpolation system (3.6) can be rewritten in matrix form as

$$W_k \begin{bmatrix} \mathbf{r}_k^\top \\ J_k^\top \end{bmatrix} = \begin{bmatrix} \mathbf{r}(\mathbf{x}_k)^\top \\ \mathbf{r}(\mathbf{y}_1)^\top \\ \vdots \\ \mathbf{r}(\mathbf{y}_p)^\top \end{bmatrix}, \quad \text{and so} \quad \begin{bmatrix} \mathbf{r}_k^\top \\ J_k^\top \end{bmatrix} = W_k^\dagger \begin{bmatrix} \mathbf{r}(\mathbf{x}_k)^\top \\ \mathbf{r}(\mathbf{y}_1)^\top \\ \vdots \\ \mathbf{r}(\mathbf{y}_p)^\top \end{bmatrix}. \tag{A.4}$$

Separately, we compute

$$W_k \begin{bmatrix} \mathbf{r}(\mathbf{x}_k)^\top \\ J(\mathbf{x}_k)^\top \end{bmatrix} - \begin{bmatrix} \mathbf{r}(\mathbf{x}_k)^\top \\ \mathbf{r}(\mathbf{y}_1)^\top \\ \vdots \\ \mathbf{r}(\mathbf{y}_p)^\top \end{bmatrix} = \begin{bmatrix} \mathbf{r}(\mathbf{x}_k)^\top \\ [\mathbf{r}(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{y}_1 - \mathbf{x}_k)]^\top \\ \vdots \\ [\mathbf{r}(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{y}_p - \mathbf{x}_k)]^\top \end{bmatrix} - \begin{bmatrix} \mathbf{r}(\mathbf{x}_k)^\top \\ \mathbf{r}(\mathbf{y}_1)^\top \\ \vdots \\ \mathbf{r}(\mathbf{y}_p)^\top \end{bmatrix} =: E. \tag{A.5}$$

[1] This argument is not in the original text, but given in the errata (`http://www.mat.uc.pt/~lnv/idfo/`).

Combining (A.4) and (A.5), we get

$$\begin{bmatrix} [\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k]^\top \\ [J(\mathbf{x}_k) - J_k]^\top \end{bmatrix} = W_k^\dagger E, \tag{A.6}$$

and so from (A.3), since D_k is invertible, we have $\hat{W}_k^\dagger = D_k^{-1} W_k^\dagger$ and hence conclude

$$\begin{bmatrix} [\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k]^\top \\ \Delta_k[J(\mathbf{x}_k) - J_k]^\top \end{bmatrix} = D_k^{-1} \begin{bmatrix} [\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k]^\top \\ [J(\mathbf{x}_k) - J_k]^\top \end{bmatrix} = \hat{W}_k^\dagger E. \tag{A.7}$$

Since the first row of $E \in \mathbb{R}^{(p+1)\times m}$ is zero, and the norms of all other rows are bounded by (2.10), we have

$$\|E\| \leq \|E\|_F = \left(0 + \sum_{t=1}^{p} \|\mathbf{r}(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{y}_t - \mathbf{x}_k) - \mathbf{r}(\mathbf{y}_t)\|^2\right)^{1/2} \leq \frac{1}{2} L_J \sqrt{p} \Delta_k^2. \tag{A.8}$$

This gives us the error bounds

$$\|\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k\| \leq \left\| \begin{bmatrix} [\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k]^\top \\ \Delta_k[J(\mathbf{x}_k) - J_k]^\top \end{bmatrix} \right\| \leq \frac{1}{2} L_J \sqrt{p} \|\hat{W}_k^\dagger\| \Delta_k^2, \tag{A.9}$$

$$\|J(\mathbf{x}_k) - J_k\| \leq \Delta_k^{-1} \left\| \begin{bmatrix} [\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k]^\top \\ \Delta_k[J(\mathbf{x}_k) - J_k]^\top \end{bmatrix} \right\| \leq \frac{1}{2} L_J \sqrt{p} \|\hat{W}_k^\dagger\| \Delta_k. \tag{A.10}$$

Thus we conclude that for any $\mathbf{y} \in B(\mathbf{x}_k, \Delta_k)$

$$\|J_k - J(\mathbf{y})\| \leq \|J_k - J(\mathbf{x}_k)\| + \|J(\mathbf{y}) - J(\mathbf{x}_k)\| \leq L_J \left(1 + \frac{1}{2}\sqrt{p}\|\hat{W}_k^\dagger\|\right) \Delta_k. \tag{A.11}$$

For convenience, define $\kappa_{\text{eg}}^{\text{r}} := L_J \left(1 + \frac{1}{2}\sqrt{p}\|\hat{W}_k^\dagger\|\right)$. Next, we compute

$$\begin{aligned} \|\mathbf{m}_k(\mathbf{y} - \mathbf{x}_k) - \mathbf{r}(\mathbf{y})\| &= \|\mathbf{r}(\mathbf{y}) - \mathbf{r}_k - J_k(\mathbf{y} - \mathbf{x}_k)\|, & \text{(A.12)} \\ &\leq \|\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k\| + \|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x}_k) - J(\mathbf{x}_k)(\mathbf{y} - \mathbf{x}_k)\| \\ &\qquad + \|J(\mathbf{x}_k) - J_k\| \cdot \|\mathbf{y} - \mathbf{x}_k\|, & \text{(A.13)} \\ &\leq \left(\frac{1}{2} L_J \sqrt{p} \|\hat{W}_k^\dagger\| + \frac{L_J}{2} + \kappa_{\text{eg}}^{\text{r}}\right) \Delta_k^2, & \text{(A.14)} \end{aligned}$$

where we use (A.11) and (2.10). Since $\|\hat{W}_k^\dagger\| \leq \sqrt{p+1}\,\Lambda$ from (2.28) in the proof of Theorem 2.17, we conclude that $\mathbf{m}_k$ is a fully linear model for $\mathbf{r}$ with constants $\kappa_{\text{eg}}^{\text{r}}$ defined above and $\kappa_{\text{ef}}^{\text{r}} = 2\kappa_{\text{eg}}^{\text{r}}$.

Since $\mathbf{m}_k$ is fully linear, (A.11) gives us $\|J_k\| \leq \|J(\mathbf{x}_k) - J_k\| + \|J(\mathbf{x}_k)\| \leq \kappa_{\text{eg}}^{\text{r}} \Delta_{\max} + J_{\max}$, so $\|J_k\|$ is uniformly bounded for all k. Since $H_k = J_k^\top J_k$, we get that $\|H_k\| = \|J_k\|^2$ is uniformly bounded for all k. Similarly, using (A.9), we have the bounds $\|\mathbf{r}(\mathbf{x}_k) - \mathbf{r}_k\| \leq \kappa_{\text{eg}}^{\text{r}} \Delta_k^2$ and $\|\mathbf{r}_k\| \leq r_{\max} + \kappa_{\text{eg}}^{\text{r}} \Delta_{\max}^2$.

To prove full linearity of m_k, we begin by computing

$$\begin{aligned}
\|\nabla m_k(\mathbf{y}-\mathbf{x}_k)-\nabla f(\mathbf{y})\| &= \|\nabla f(\mathbf{y})-J_k^\top \mathbf{r}_k - J_k^\top J_k(\mathbf{y}-\mathbf{x}_k)\|, && \text{(A.15)}\\
&\leq \|\nabla f(\mathbf{y})-\nabla f(\mathbf{x}_k)\| + \|J(\mathbf{x}_k)^\top(\mathbf{r}(\mathbf{x}_k)-\mathbf{r}_k)\| \\
&\qquad + \|(J(\mathbf{x}_k)-J_k)^\top \mathbf{r}_k\| + \|J_k^\top J_k\|\cdot\|\mathbf{y}-\mathbf{x}_k\|, && \text{(A.16)}\\
&\leq L_{\nabla f}\Delta_k + J_{\max}\kappa_{\text{eg}}^{\text{r}}\Delta_k^2 \\
&\qquad + \kappa_{\text{eg}}^{\text{r}} r_{\max}\Delta_k + \left(\kappa_{\text{eg}}^{\text{r}}\Delta_{\max}+J_{\max}\right)^2\Delta_k, && \text{(A.17)}\\
&\leq \kappa_{\text{eg}}\Delta_k, && \text{(A.18)}
\end{aligned}$$

where $\kappa_{\text{eg}} := L_{\nabla f} + J_{\max}\kappa_{\text{eg}}^{\text{r}}\Delta_{\max} + \kappa_{\text{eg}}^{\text{r}} r_{\max} + (\kappa_{\text{eg}}^{\text{r}}\Delta_{\max}+J_{\max})^2$, as required. Then, we use (2.10) and the above to get

$$\begin{aligned}
|m_k(\mathbf{y}-\mathbf{x}_k)-f(\mathbf{y})| &= \left|f(\mathbf{y}) - \frac{1}{2}\|\mathbf{r}_k\|^2 - \mathbf{g}_k^\top(\mathbf{y}-\mathbf{x}_k) - \frac{1}{2}(\mathbf{y}-\mathbf{x}_k)^\top H_k(\mathbf{y}-\mathbf{x}_k)\right|, && \text{(A.19)}\\
&\leq \left|f(\mathbf{y})-f(\mathbf{x}_k)-\nabla f(\mathbf{x}_k)^\top(\mathbf{y}-\mathbf{x}_k)\right| \\
&\qquad + \frac{1}{2}\|\mathbf{r}(\mathbf{x}_k)-\mathbf{r}_k\|\left(\|\mathbf{r}(\mathbf{x}_k)\|+\|\mathbf{r}_k\|\right) \\
&\qquad + \left\|\nabla f(\mathbf{x}_k) - \mathbf{g}_k - \frac{1}{2}H_k(\mathbf{y}-\mathbf{x}_k)\right\|\cdot\|\mathbf{y}-\mathbf{x}_k\|, && \text{(A.20)}\\
&\leq \frac{1}{2}L_{\nabla f}\Delta_k^2 + \frac{1}{2}\kappa_{\text{eg}}^{\text{r}}\Delta_k^2\left(2r_{\max}+\kappa_{\text{eg}}^{\text{r}}\Delta_{\max}\right) \\
&\qquad + \left[\|\nabla f(\mathbf{x}_k)-\nabla m_k(\mathbf{x}_k)\| + \frac{1}{2}\|H_k\|\cdot\|\mathbf{y}-\mathbf{x}_k\|\right]\cdot\Delta_k, && \text{(A.21)}\\
&\leq \frac{1}{2}L_{\nabla f}\Delta_k^2 + \frac{1}{2}\kappa_{\text{eg}}^{\text{r}}\left(2r_{\max}+\kappa_{\text{eg}}^{\text{r}}\Delta_{\max}\right)\Delta_k^2 \\
&\qquad + \left[\kappa_{\text{eg}}\Delta_k + (\kappa_{\text{eg}}^{\text{r}}\Delta_{\max}+J_{\max})^2\Delta_k\right]\Delta_k, && \text{(A.22)}\\
&\leq \kappa_{\text{ef}}\Delta_k^2, && \text{(A.23)}
\end{aligned}$$

where $\kappa_{\text{ef}} := \kappa_{\text{eg}} + L_{\nabla f}/2 + \kappa_{\text{eg}}^{\text{r}}\left(r_{\max}+\kappa_{\text{eg}}^{\text{r}}\Delta_{\max}/2\right) + (\kappa_{\text{eg}}^{\text{r}}\Delta_{\max}+J_{\max})^2$. □

Appendix B

Calculating Geometry-Improving Steps

Here, we provide Algorithm B.1, a method for solving calculating a geometry-improving step (2.52) with linear models subject to bound constraints. That is, if our current trust-region is $B(\mathbf{x}, \Delta)$, our geometry-improving step is $\mathbf{x} + \mathbf{s}$, where $\mathbf{s}$ solves

$$\arg\max_{\mathbf{s}\in\mathbb{R}^n} \mathbf{g}^\top \mathbf{s} \quad \text{subject to} \quad \mathbf{a} \leq \mathbf{s} \leq \mathbf{b} \quad \text{and} \quad \|\mathbf{s}\| \leq \Delta, \tag{B.1}$$

where the bounds on the variables have been shifted by $-\mathbf{x}$, and $\pm\mathbf{g}$ is the gradient of the relevant Lagrange polynomial. Note that, to solve (2.52), we need to solve (B.1) with $\pm\mathbf{g}$, to find the point with maximal absolute value. We assume, in (B.1), that $a_i \leq 0 \leq b_i$ for all i—i.e. the base point $\mathbf{x}$ is feasible—and $\Delta > 0$ (otherwise $\mathbf{s} = \mathbf{0}$ is the solution).

Algorithm B.1 Solve geometry-improving subproblem (B.1).

Initialise $\mathbf{s}^{(0)} := \mathbf{0}$ and step direction $\mathbf{d}^{(0)} := \mathbf{g}$.
Define fixed variables $\mathcal{A}_0 := \{i : g_i = 0\}$ and free variables $\mathcal{I}_0 := \{1, \ldots, n\} \setminus \mathcal{A}_0$.
for $j = 0, 1, 2, \ldots, n-1$ **do**
    if $\|\mathbf{d}^{(j)}\| = 0$ **then**
        return $\mathbf{s}^{(j)}$.
    end if
    Find tentative step length α_j by finding the largest solution to $\|\mathbf{s}^{(j)} + \alpha_j \mathbf{d}^{(j)}\|^2 = \Delta^2$.
    if $\mathbf{a} \leq \mathbf{s}^{(j)} + \alpha_j \mathbf{d}^{(j)} \leq \mathbf{b}$ **then**
        return $\mathbf{s}^{(j+1)} = \mathbf{s}^{(j)} + \alpha_j \mathbf{d}^{(j)}$.
    else
        Let $i \in \mathcal{I}_j$ be any index such that $s_i^{(j)} + \alpha_j d_i^{(j)} \notin [a_i, b_i]$.
        Let $\beta_j < \alpha_j$ be the largest value such that $s_i^{(j)} + \beta_j d_i^{(j)} \in [a_i, b_i]$.
        Set $\mathbf{s}^{(j+1)} = \mathbf{s}^{(j)} + \beta_j \mathbf{d}^{(j)}$.
        Set $\mathcal{I}_{j+1} = \mathcal{I}_j \setminus \{i\}$, $\mathcal{A}_{j+1} = \mathcal{A}_j \cup \{i\}$, and $\mathbf{d}^{(j+1)} = \mathbf{d}^{(j)}$, except for $d_i^{(j+1)} = 0$.
    end if
end for
return $\mathbf{s}^{(n)}$.

We observe that, at the start of iteration j of Algorithm B.1 (provided we have not already terminated), the index sets $\mathcal{A}_j$ and $\mathcal{I}_j$ partition $\{1,\dots,n\}$. In addition, our current search direction $\mathbf{d}^{(j)}$ has $d_i^{(j)} = 0$ for all $i \in \mathcal{A}_j$, and $d_i^{(j)} = g_i \neq 0$ for all $i \in \mathcal{I}_j$. Lastly, our current iterate $\mathbf{s}^{(j)}$ has $s_i^{(j)} = \gamma_{j-1} g_i$ for all $i \in \mathcal{I}_j$, where $\gamma_j := \sum_{l=-1}^{j} \beta_l$ (with the convention $\beta_{-1} := 0$).

The next two lemmas prove some key preliminary results that we will use to show that Algorithm B.1 solves (B.1). We note that Algorithm B.1 always terminates, as it only has finitely-many iterations.

Lemma B.1. *If $\mathbf{a} \leq \mathbf{0} \leq \mathbf{b}$, $\Delta > 0$ and we have not terminated by the start of iteration j, then $\|\mathbf{s}^{(j)}\| < \Delta$, $\gamma_{j-1} \geq 0$, and*

$$s_i^{(j)} = \begin{cases} a_i, & g_i < 0, \\ b_i & g_i > 0, \\ 0 & g_i = 0, \end{cases} \qquad \forall i \in \mathcal{A}_j. \tag{B.2}$$

Also, the equation for α_j has a real solution, with $\alpha_j > -\gamma_{j-1}$.

Proof. In the first iteration $j = 0$, all statements are true (since $\Delta > 0$, $\mathbf{s}^{(j)} = \mathbf{0}$, $\gamma_{-1} = 0$ and $\mathcal{A}_j = \{i : g_i = 0\}$) except possibly that α_0 has a real solution. In any iteration, we terminate in line 5 if and only if $\mathcal{I}_j = \emptyset$ (or equivalently $\mathbf{d}^{(j)} = \mathbf{0}$). So in iteration 0, we search for α_0 if $\mathbf{d}^{(0)} \neq \mathbf{0}$ and want $\alpha_0^2 \|\mathbf{d}^{(0)}\|^2 = \Delta^2$; hence we get $\alpha_0 = \Delta / \|\mathbf{d}^{(0)}\| > \gamma_{-1}$ is well-defined.

We now proceed by induction and suppose $j > 0$ and all statements are true for iterations $0, \dots, j-1$. Therefore, we start after line 7 in iteration $j-1$ (i.e. just after finding the well-defined $\alpha_j > -\gamma_{j-1}$). We wish to prove statements about the start of iteration j, so we cannot terminate in line 9; let $i \in \mathcal{I}_{j-1}$ be the chosen index in line 11. Since

$$s_i^{(j-1)} + \alpha_{j-1} d_i^{(j-1)} = (\gamma_{j-2} + \alpha_{j-1}) g_i \notin [a_i, b_i], \tag{B.3}$$

and $\alpha_{j-1} \geq -\gamma_{j-2}$, we must have $g_i \neq 0$ and $\gamma_{j-2} + \alpha_{j-1} > 0$ (since $0 \in [a_i, b_i]$), and we get

$$\beta_{j-1} = -\gamma_{j-2} + \begin{cases} b_i / g_i, & g_i > 0, \\ a_i / g_i, & g_i < 0. \end{cases} \tag{B.4}$$

Thus since $a_i \leq 0$ and $b_i \geq 0$ we have $\gamma_{j-1} = \gamma_{j-2} + \beta_{j-1} \geq 0$. From (B.4), we also conclude that

$$s_i^{(j)} = s_i^{(j-1)} + \beta_{j-1} d_i^{(j-1)} = \begin{cases} b_i, & g_i > 0, \\ a_i, & g_i < 0, \end{cases} \tag{B.5}$$

giving us (B.2). Iteration $j-1$ finishes by defining $\mathbf{s}^{(j)} = \mathbf{s}^{(j-1)} + \beta_{j-1}\mathbf{d}^{(j)}$, and so

$$\|\mathbf{s}^{(j)}\|^2 = \sum_{i\in\mathcal{A}_{j-1}} (s_i^{(j-1)})^2 + (\gamma_{j-2} + \beta_{j-1})^2 \sum_{i\in\mathcal{I}_{j-1}} g_i^2, \tag{B.6}$$

$$< \sum_{i\in\mathcal{A}_{j-1}} (s_i^{(j-1)})^2 + (\gamma_{j-2} + \alpha_{j-1})^2 \sum_{i\in\mathcal{I}_{j-1}} g_i^2, \tag{B.7}$$

$$= \Delta^2, \tag{B.8}$$

since $-\gamma_{j-2} \leq \beta_{j-1} \leq \alpha_{j-1}$ and using the definition of α_{j-1}. Here, we have finished iteration $j-1$ and it only remains to show that α_j is well-defined and $\alpha_j \geq -\gamma_{j-1}$.

We only search for α_j if $\mathcal{I}_j \neq \emptyset$, and in this case we wish to solve

$$\sum_{i\in\mathcal{A}_j} (s_i^{(j)})^2 + \sum_{i\in\mathcal{I}_j} (\gamma_{j-1} + \alpha_j)^2 g_i^2 = \Delta^2, \tag{B.9}$$

which gives

$$\alpha_j = -\gamma_{j-1} + \sqrt{\frac{\Delta^2 - \sum_{i\in\mathcal{A}_j} (s_i^{(j)})^2}{\sum_{i\in\mathcal{I}_j} g_i^2}}. \tag{B.10}$$

The denominator of the last term is strictly positive since $\mathcal{I}_j \neq \emptyset$ and $g_i \neq 0$ for $i \in \mathcal{I}_j$. The numerator is positive since $\sum_{i\in\mathcal{A}_j} (s_i^{(j)})^2 \leq \|\mathbf{s}^{(j)}\|^2 < \Delta^2$. Thus α_j is well-defined, and $\alpha_j > -\gamma_{j-1}$. □

Lemma B.2. *If $\mathbf{a} \leq \mathbf{0} \leq \mathbf{b}$ and $\Delta > 0$, then the sequence $(\gamma_{j-1} + \alpha_j)$ is increasing in j (until termination is reached).*

Proof. Let i_j be the index fixed in iteration $j-1$. Then by (B.3) we have $(\gamma_{j-2} + \alpha_{j-1})g_{i_j} > b_{i_j} \geq 0$ if $g_{i_j} > 0$ or $(\gamma_{j-2} + \alpha_{j-1})g_{i_j} < a_{i_j} \leq 0$ otherwise. Either way, defining $c_{i_j} := b_{i_j}$ if $g_{i_j} > 0$ or $c_{i_j} := a_{i_j}$ otherwise, we have $(\gamma_{j-2} + \alpha_{j-1})^2 > c_{i_j}^2 / g_{i_j}^2$. Then we compute, using $s_i^{(j-1)} = s_i^{(j)}$ for all $i \in \mathcal{A}_{j-1} \subset \mathcal{A}_j$,

$$(\gamma_{j-1} + \alpha_j)^2 = \frac{\Delta^2 - \sum_{i\in\mathcal{A}_j} (s_i^{(j)})^2}{\sum_{i\in\mathcal{I}_j} g_i^2}, \tag{B.11}$$

$$= \frac{\Delta^2 - \sum_{i\in\mathcal{A}_{j-1}} (s_i^{(j-1)})^2 - c_{i_j}^2}{\sum_{i\in\mathcal{I}_{j-1}} g_i^2 - g_{i_j}^2}, \tag{B.12}$$

$$> \frac{\Delta^2 - \sum_{i\in\mathcal{A}_{j-1}} (s_i^{(j-1)})^2 - (\gamma_{j-2} + \alpha_{j-1})^2 g_{i_j}^2}{\sum_{i\in\mathcal{I}_{j-1}} g_i^2 - g_{i_j}^2}, \tag{B.13}$$

$$= \frac{\left[\Delta^2 - \sum_{i\in\mathcal{A}_{j-1}} (s_i^{(j-1)})^2\right] \left[1 - g_{i_j}^2 / \left(\sum_{i\in\mathcal{I}_{j-1}} g_i^2\right)\right]}{\sum_{i\in\mathcal{I}_{j-1}} g_i^2 - g_{i_j}^2}, \tag{B.14}$$

$$= \frac{\left[\Delta^2 - \sum_{i\in\mathcal{A}_{j-1}} (s_i^{(j-1)})^2\right] \left[\left(\sum_{i\in\mathcal{I}_{j-1}} g_i^2\right) - g_{i_j}^2\right]}{\left[\left(\sum_{i\in\mathcal{I}_{j-1}} g_i^2\right)\right] \left[\sum_{i\in\mathcal{I}_{j-1}} g_i^2 - g_{i_j}^2\right]}, \tag{B.15}$$

$$= (\gamma_{j-2} + \alpha_{j-1})^2, \tag{B.16}$$

and the result follows since $\gamma_{j-1} + \alpha_j > 0$ for all j (by Lemma B.1). □

Theorem B.3. *Suppose* $\mathbf{a} \le \mathbf{0} \le \mathbf{b}$ *and* $\Delta > 0$. *Then Algorithm B.1 returns a minimiser of* (B.1).

Proof. The problem (B.1) is convex, and so it suffices to find a KKT point. That is, we need to find $\mathbf{s}^*$ and Lagrange multipliers $\underline{\lambda}_i$, $\overline{\lambda}_i$ (for $i = 1, \dots, n$) and μ such that

$$-\mathbf{g} = -\mu\mathbf{s}^* + \sum_{i=1}^{n}(\underline{\lambda}_i\mathbf{e}_i - \overline{\lambda}_i\mathbf{e}_i), \tag{B.17a}$$

$$s_i^* - a_i \ge 0, \qquad \underline{\lambda}_i \ge 0, \qquad \underline{\lambda}_i(s_i^* - a_i) = 0, \qquad \forall i = 1, \dots, n, \tag{B.17b}$$

$$b_i - s_i^* \ge 0, \qquad \overline{\lambda}_i \ge 0, \qquad \overline{\lambda}_i(b_i - s_i^*) = 0, \qquad \forall i = 1, \dots, n, \tag{B.17c}$$

$$\frac{1}{2}(\Delta^2 - \|\mathbf{s}^*\|^2) \ge 0, \qquad \mu \ge 0, \qquad \frac{1}{2}\mu(\Delta^2 - \|\mathbf{s}^*\|^2) = 0. \tag{B.17d}$$

First, suppose we terminate in line 5 or 17 with result $\mathbf{s}$. Then we have (B.2) with $\mathcal{A}_j = \{1, \dots, n\}$ and $\|\mathbf{s}\| \le \Delta$. In this case, we have a KKT point with $\mu = 0$, $\overline{\lambda}_i = \max(g_i, 0)$ and $\underline{\lambda}_i = \max(-g_i, 0)$.

In the other case (termination on line 9 of iteration j), we have $\mathcal{I}_j \ne \emptyset$, $\mathbf{a} \le \mathbf{s} \le \mathbf{b}$ and $\|\mathbf{s}\| = \Delta$, with

$$s_i = \begin{cases} (\gamma_{j-1} + \alpha_j)g_i, & i \in \mathcal{I}_j, \\ a_i, & i \in \mathcal{A}_j \text{ and } g_i < 0, \\ b_i & i \in \mathcal{A}_j \text{ and } g_i > 0, \\ 0 & g_i = 0. \end{cases} \tag{B.18}$$

Using (B.17a) we get Lagrange multipliers $\mu = 1/(\gamma_{j-1} + \alpha_j) > 0$ (from Lemma B.1) and

$$\overline{\lambda}_i = \begin{cases} 0, & i \in \mathcal{I}_j, \\ g_i - \mu b_i, & i \in \mathcal{A}_j \text{ and } g_i > 0, \\ 0, & i \in \mathcal{A}_j \text{ and } g_i \le 0, \end{cases} \qquad \text{and} \qquad \underline{\lambda}_i = \begin{cases} 0, & i \in \mathcal{I}_j, \\ \mu a_i - g_i, & i \in \mathcal{A}_j \text{ and } g_i < 0, \\ 0, & i \in \mathcal{A}_j \text{ and } g_i \ge 0. \end{cases} \tag{B.19}$$

For $\mathbf{s}$ to be a KKT point, it remains to show that $\overline{\lambda}_i, \underline{\lambda}_i \ge 0$; that is,

$$(\gamma_{j-1} + \alpha_j)g_i \begin{cases} \ge b_i, & i \in \mathcal{A}_j \text{ and } g_i > 0, \\ \le a_i, & i \in \mathcal{A}_j \text{ and } g_i < 0. \end{cases} \tag{B.20}$$

This follows from Lemma B.2, as for any $i \in \mathcal{A}_j$ with $g_i > 0$, there was an earlier iteration $j_i < j$ with $(\gamma_{j_i-1} + \alpha_{j_i})g_i > b_i$ (from (B.3)), and so $(\gamma_{j-1} + \alpha_j)g_i \ge (\gamma_{j_i-1} + \alpha_{j_i})g_i > b_i$ and (B.20) holds; a similar reasoning holds for $i \in \mathcal{A}_j$ with $g_i < 0$. □

Appendix C

Comparison of Sample Averaging and Regression

There are two places in Algorithm 3.1 where noise in objective evaluations can have an impact: the construction of the model (3.1), and the measurement of objective decrease (3.3). We show below that the errors in model construction due to noise are likely comparable when using either sample averaging or regression. However, for a fixed level of noise, sample averaging will produce a better estimate of objective decrease (compare [164, Lemma 9.1], for instance). Thus, overall, we would expect sample averaging to perform somewhat better than regression, when considering overall robustness. Since sample averaging may use more objective evaluations per iteration, this may not be the case when the computational budget is limited.

Error Bounds on Model Estimation Here, we give a short argument that sample averaging and regression models produce comparably good models. For simplicity, suppose we wish to construct a model for a linear function $f(\mathbf{x}) = c + \mathbf{g}^\top \mathbf{x}$ from noisy evaluations $\widetilde{f}(\mathbf{x}) = f(\mathbf{x}) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$ is i.i.d. stochastic noise.

If we perform sample averaging using N samples at a given interpolation point $\mathbf{y}_t$, we get an unbiased estimate for f with smaller variance:

$$\widetilde{f}_N(\mathbf{y}_t) = \frac{1}{N}\sum_{i=1}^{N}(f(\mathbf{y}_t) + \epsilon_i) \sim N\left(f(\mathbf{y}_t), \frac{\sigma^2}{N}\right). \tag{C.1}$$

Now suppose we construct a regression model as per (3.5) using points $Y := \{\mathbf{y}_0, \ldots, \mathbf{y}_p\}$ for some $p \geq n$. We assume that Y is *strongly* Λ-poised [60, Definition 4.10] in $B(\mathbf{y}_0, \Delta)$, which is a stronger condition[1] than Definition 2.16, but better suited to comparing the geometry of sets with different sizes p.

[1] It can be achieved if, for instance, Y is formed by concatenating several sets of size $n+1$ that are all Λ-poised for linear *interpolation* (Definition 2.11).

Under these conditions, the Gauss-Markov Theorem (e.g. [161, Chapter 10.1]) implies that the regression model from (3.5) gives an optimal unbiased estimator $(\widetilde{c}, \widetilde{\mathbf{g}})$ for $(c, \mathbf{g})$ with error (co)variance

$$\mathbb{E}\left[\left(\begin{bmatrix}\widetilde{c}\\ \widetilde{\mathbf{g}}\end{bmatrix} - \begin{bmatrix}c\\ \mathbf{g}\end{bmatrix}\right)\left(\begin{bmatrix}\widetilde{c}\\ \widetilde{\mathbf{g}}\end{bmatrix} - \begin{bmatrix}c\\ \mathbf{g}\end{bmatrix}\right)^{\top}\right] = \frac{\sigma^2}{N}(W_k^{\top}W_k)^{-1}. \tag{C.2}$$

By shifting Y to $\hat{Y} := \{(\mathbf{y}_t - \mathbf{y}_0)/\Delta : t = 0, \dots, p\}$ as in the proof of Lemma 4.2, we have (A.3), and so the variance satisfies

$$\left\|(W_k^{\top}W_k)^{-1}\right\| = \left\|D_k\left(\hat{W}_k^{\top}\hat{W}_k\right)^{-1}D_k^{\top}\right\| \leq \max(1, \Delta^{-2})\left\|(\hat{W}_k^{\top}\hat{W}_k)^{-1}\right\| = \frac{\max(1, \Delta^{-2})}{\sigma_{\min}(\hat{W}_k)^2}, \tag{C.3}$$

where $\sigma_{\min}(\hat{W}_k)$ is the smallest singular value of $\hat{W}_k$, since $\|D_k\| = \max(1, \Delta^{-1})$. However, from [60, Theorem 4.12], the strong Λ-poisedness of Y gives

$$\frac{1}{\sigma_{\min}(\hat{W}_k)} \leq \frac{\theta(n+1)}{\sqrt{p+1}}\Lambda, \tag{C.4}$$

for some constant $\theta > 0$. All together, we get

$$\mathbb{E}\left[\left(\begin{bmatrix}\widetilde{c}\\ \widetilde{\mathbf{g}}\end{bmatrix} - \begin{bmatrix}c\\ \mathbf{g}\end{bmatrix}\right)\left(\begin{bmatrix}\widetilde{c}\\ \widetilde{\mathbf{g}}\end{bmatrix} - \begin{bmatrix}c\\ \mathbf{g}\end{bmatrix}\right)^{\top}\right] \leq \frac{\sigma^2 \max(1, \Delta^{-2})\theta^2(n+1)^2\Lambda^2}{N(p+1)}, \tag{C.5}$$

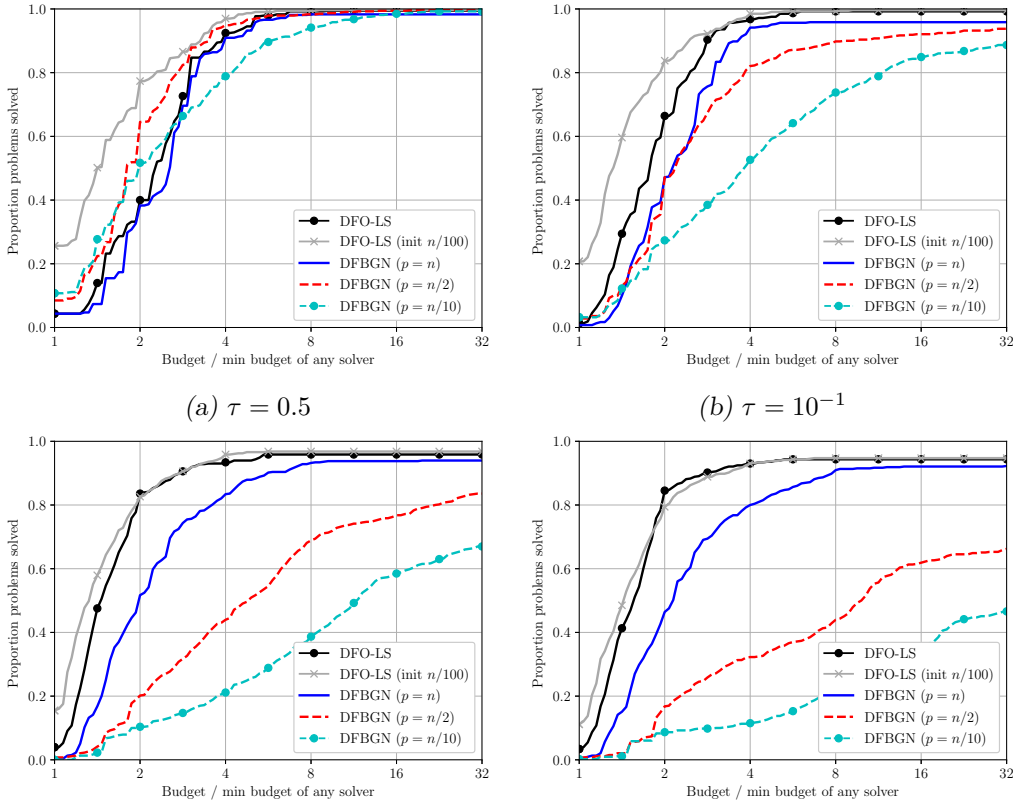
so the square error in the regression model $(\widetilde{c}, \widetilde{\mathbf{g}})$ is inversely proportional to $N(p+1)$, the total number of evaluations of $\widetilde{f}$ used in building the right-hand side of (3.6). We conclude that, all else being equal (including the strong Λ-poisedness of Y), we would get the same model error from using $p + 1 = c(n + 1)$ points with no sample averaging, or $p + 1 = n + 1$ points and using c samples per point. This provides further support for the similar results for sample averaging and regression models observed in Section 5.2.4.

Appendix D

Extended Results for DFBGN

In Figure D.1, we show performance profiles comparing DFBGN with DFO-LS on the (MW) problem collection. Since these problems are low-dimensional ($n \leq 12$), they do not represent the setting for which DFBGN is designed, however we include them here for completeness.

Similar to Figure 7.7, we see that DFBGN performs better (in terms of evaluations) the larger the block size p, with the performance with $p = n$ similar to DFO-LS. For very low accuracy $\tau = 0.5$, DFBGN with $p < n$ sometimes outperforms DFO-LS (with the full initialisation cost), but not DFO-LS with reduced initialisation cost.



(a) $\tau = 0.5$

(b) $\tau = 10^{-1}$

(c) $\tau = 10^{-3}$

(d) $\tau = 10^{-5}$

Figure D.1. Performance profiles (in evaluations) comparing DFO-LS (with and without reduced initialisation cost) with DFBGN (various p choices) for different accuracy levels. Results are an average of 10 runs for each problem, with a budget of $100(n + 1)$ evaluations and a 12 hour runtime limit per instance. The problem collection is (MW).

Appendix E

Test Problems

Here, we give details of our collections of test problems; see Section 2.4.1 for details.

E.1 Moré and Wild Collection

#	Objective Function	n	m	$2f(\mathbf{x}_0)$	$2f^*$
1	Linear (full rank)	9	45	72	36
2	Linear (full rank)	9	45	1125	36
3	Linear (rank 1)	7	35	1.165420×10^{7}	8.380282
4	Linear (rank 1)	7	35	1.168591×10^{9}	8.380282
5	Linear (rank 1 w. zero row/col)	7	35	4.989195×10^{6}	9.880597
6	Linear (rank 1 w. zero row/col)	7	35	5.009356×10^{8}	9.880597
7	Rosenbrock	2	2	24.2	0
8	Rosenbrock	2	2	1.795769×10^{6}	0
9	Helical Valley	3	3	2500	0
10	Helical Valley	3	3	10600	0
11	Powell Singular	4	4	215	0
12	Powell Singular	4	4	1.615400×10^{6}	0
13	Freudenstein & Roth	2	2	400.5	48.98425
14	Freudenstein & Roth	2	2	1.545754×10^{8}	48.98425
15	Bard	3	15	41.68170	8.214877×10^{-3}
16	Bard	3	15	1306.234	8.214877×10^{-3}
17	Kowalik & Osborne	4	11	5.313172×10^{-3}	3.075056×10^{-4}
18	Meyer	3	16	1.693608×10^{9}	87.94586
19	Watson	6	31	16.43083	2.287670×10^{-3}
20	Watson	6	31	2.323367×10^{6}	2.287670×10^{-3}
21	Watson	9	31	26.90417	1.399760×10^{-6}
22	Watson	9	31	8.158877×10^{6}	1.399760×10^{-6}
23	Watson	12	31	73.67821	4.722381×10^{-10}
24	Watson	12	31	2.059384×10^{7}	4.722381×10^{-10}
25	Box 3d	3	10	1031.154	0
26	Jennrich & Sampson	2	10	4171.306	124.3622
27	Brown & Dennis	4	20	7.926693×10^{6}	8.582220×10^{4}
28	Brown & Dennis	4	20	3.081064×10^{11}	8.582220×10^{4}
29	Chebyquad	6	6	4.642817×10^{-2}	0
30	Chebyquad	7	7	3.377064×10^{-2}	0
31	Chebyquad	8	8	3.861770×10^{-2}	3.516874×10^{-3}
32	Chebyquad	9	9	2.888298×10^{-2}	0
33	Chebyquad	10	10	3.376327×10^{-2}	4.772714×10^{-3}
34	Chebyquad	11	11	2.674060×10^{-2}	2.799762×10^{-3}
35	Brown almost-linear	10	10	273.2480	0
36	Osborne 1	5	33	16.17411	5.464895×10^{-5}
37	Osborne 2	11	65	2.093420	4.013774×10^{-2}
38	Osborne 2	11	65	199.6847	4.013774×10^{-2}
39	bdqrtic	8	8	904	10.23897
40	bdqrtic	10	12	1356	18.28116
41	bdqrtic	11	14	1582	22.26059
42	bdqrtic	12	16	1808	26.27277
43	Cube	5	5	56.5	0
44	Cube	6	6	70.5625	0
45	Cube	8	8	98.6875	0
46	Mancino	5	5	2.539084×10^{9}	0
47	Mancino	5	5	6.873795×10^{12}	0
48	Mancino	8	8	3.367961×10^{9}	0
49	Mancino	10	10	3.735127×10^{9}	0
50	Mancino	12	12	3.991072×10^{9}	0
51	Mancino	12	12	1.130015×10^{13}	0
52	Heart8ls	8	8	9.385672	0
53	Heart8ls	8	8	3.365815×10^{10}	0

Table E.1. Details of test problems from [157], including the value of f^* used in (2.55) for each problem. Note that, in line with the implementation of our least-squares solvers, we show $2f(\mathbf{x}_0)$ and $2f^*$; i.e. excluding the $1/2$ factor in (2.6).

E.2 CUTEst Nonlinear Least-Squares Collection

#	Problem	n	m	$2f(\mathbf{x}_0)$	$2f^*$	Parameters
1	ARGLALE	100	400	700	300	$N = 100$
2	ARGLBLE	100	400	5.460944×10^{14}	99.62547	$N = 100$
3	ARGTRIG	100	100	32.99641	0	$N = 100$
4	ARTIF	100	100	36.59115	0	$N = 100$
5	ARWHDNE	100	198	495	27.66203	$N = 100$
6	BDVALUES	100	100	1.943417×10^{7}	0	$NDP = 102$
7	BRATU2D	64	64	0.1560738	0	$P = 10$
8	BRATU2DT	64	64	0.4521311	1.853474×10^{-5}	$P = 10$
9	BRATU3D	27	27	4.888529	0	$P = 5$
10	BROWNALE	100	100	2.524757×10^{5}	0	$N = 100$
11	BROYDN3D	100	100	111	0	$N = 100$
12	BROYDNBD	100	100	2404	0	$N = 100$
13	CBRATU2D	50	50	0.4822531	0	$P = 7$
14	CHANDHEQ*	100	100	6.923365	0	$N = 100$
15	CHEMRCTA*	100	100	3.0935	0	$N = 50$
16	CHEMRCTB*	100	100	1.446513	1.404424×10^{-3}	$N = 100$
17	CHNRSBNE	50	98	7635.84	0	$N = 50$
18	DRCAVTY1	100	100	0.4513889	0	$M = 10$
19	DRCAVTY2	100	100	0.4513889	5.449602×10^{-3}	$M = 10$
20	DRCAVTY3	100	100	0.4513889	0	$M = 10$
21	EIGENA*	110	110	285	0	$N = 10$
22	EIGENB	110	110	19	0	$N = 10$
23	FLOSP2HH	59	59	519	0.3333333	$M = 2$
24	FLOSP2HL	59	59	519	0.3333333	$M = 2$
25	FLOSP2HM	59	59	519	0.3333333	$M = 2$
26	FLOSP2TH	59	59	516	0	$M = 2$
27	FLOSP2TL	59	59	516	0	$M = 2$
28	FLOSP2TM	59	59	516	0	$M = 2$
29	FREURONE	100	198	9.95565×10^{4}	1.196458×10^{4}	$N = 100$
30	HATFLDG	25	25	27	0	—
31	HYDCAR20	99	99	1341.663	0	—
32	HYDCAR6	29	29	704.1073	0	—
33	INTEGREQ	100	100	0.5730503	0	$N = 100$
34	METHANB8	31	31	1.043105	0	—
35	METHANL8	31	31	4345.100	0	—
36	MOREBVNE	100	100	3.633100×10^{-4}	0	$N = 100$
37	MSQRTA	100	100	212.7162	0	$P = 10$
38	MSQRTB	100	100	205.0846	0	$P = 10$
39	OSCIGRNE	100	100	6.120720×10^{8}	0	$N = 100$
40	PENLT1NE	100	101	1.144806×10^{11}	9.025000×10^{-9}	$N = 100$
41	PENLT2NE	100	200	1.591383×10^{6}	0.9809377	$N = 100$
42	POWELLSE	100	100	41875	0	$N = 100$
43	QR3D*	40	40	1.2	0	$M = 5$
44	QR3DBD*	37	40	1.2	0	$M = 5$
45	SEMICN2U	100	100	2.025037×10^{4}	0	$(N, LN) = (100, 90)$
46	SEMICON2*	100	100	2.025037×10^{4}	0	$(N, LN) = (100, 90)$
47	SPMSQRT	100	164	74.33542	0	$M = 34$
48	VARDIMNE	100	102	1.310584×10^{14}	0	$N = 100$
49	WATSONNE	31	31	30	0	$N = 31$
50	YATP1SQ	120	120	2.073643×10^{6}	0	$N = 10$
51	YATP2SQ	120	120	1.831687×10^{5}	0	$N = 10$
52	LUKSAN11	100	198	626.0640	0	—
53	LUKSAN12	98	192	3.2160×10^{4}	4292.197	—
54	LUKSAN13	98	224	6.4352×10^{4}	2.518886×10^{4}	—
55	LUKSAN14	98	224	2.6880×10^{4}	123.9235	—
56	LUKSAN15	100	196	2.701585×10^{4}	3.569697	—
57	LUKSAN16	100	196	1.306848×10^{4}	3.569697	—
58	LUKSAN17	100	196	1.687370×10^{6}	0.4931613	—
59	LUKSAN21	100	100	99.98751	0	—
60	LUKSAN22	100	198	2.487686×10^{4}	872.9230	—

*Table E.2. Details of medium-scale test problems from the CUTEst test set (showing $2f(\mathbf{x}_0)$ and $2f^*$, as per Table E.1). The set of problems are taken primarily from [87, 155, 143]. Some problems are variable-dimensional; the relevant parameters yielding the given (n, m) are provided. Problems marked * have box constraints. The value of n shown excludes fixed variables.*

E.3 CUTEst General Objective Collection

#	Problem	n	$f(\mathbf{x}_0)$	$f(\mathbf{x}^*)$	Parameters
1	ARWHEAD	100	297	0	$N = 100$
2	BDEXP*	100	26.52572	0	$N = 100$
3	BOX	100	0	-11.24044	$N = 100$
4	BOXPOWER	100	866.2462	0	$N = 100$
5	BROYDN7D	100	350.9842	40.12284	$N/2 = 50$
6	CHARDIS1	98	830.9353	0	$NP1 = 50$
7	COSINE	100	86.88067	-99	$N = 100$
8	CURLY10	100	-6.237221×10^{-3}	-1.003163×10^{4}	$N = 100$
9	CURLY20	100	-1.296535×10^{-2}	-1.003163×10^{4}	$N = 100$
10	DIXMAANA	90	856	1	$M = 30$
11	DIXMAANF	90	1.225292×10^{3}	1	$M = 30$
12	DIXMAANP	90	2.128648×10^{3}	1	$M = 30$
13	ENGVAL1	100	5841	109.0881	$N = 100$
14	FMINSRF2	64	23.461408	1	$P = 8$
15	FMINSURF	64	32.84031	1	$P = 8$
16	NCB20	110	202.002	179.7358	$N = 100$
17	NCB20B	100	200	196.6801	$N = 100$
18	NONCVXU2	100	2.639748×10^{6}	231.8274	$N = 100$
19	NONCVXUN	100	2.727010×10^{6}	231.6808	$N = 100$
20	NONDQUAR	100	106	0	$N = 100$
21	ODC	100	0	-1.098018×10^{-2}	$(NX, NY) = (10, 10)$
22	PENALTY3	100	9.801798×10^{7}	0.001	$N/2 = 50$
23	POWER	100	2.550250×10^{7}	0	$N = 100$
24	RAYBENDL	62	98.03445	96.25168	$NKNOTS = 32$
25	SCHMVETT	100	-280.2864	-294	$N = 100$
26	SINEALI*	100	-0.8414710	-9.900962×10^{3}	$N = 100$
27	SINQUAD	100	0.6561	-4.005585×10^{3}	$N = 100$
28	TOINTGOR	50	5.073786×10^{3}	1.373905×10^{3}	—
29	TOINTGSS	100	892	10.10204	$N = 100$
30	TOINTPSP	50	1.827709×10^{3}	225.5604	—

*Table E.3. Details of medium-scale general objective test problems from the CUTEst test set, including the value of $f(\mathbf{x}^*)$ used in (2.55) for each problem. Some problems are variable-dimensional; the relevant parameters yielding the given n are provided. Problems marked * have bound constraints. The value of n shown excludes fixed variables. Some of the problems were taken from [144].*

E.4 Global Optimisation Collection

#	Problem	n	Bounds	Global minimum $f(\mathbf{x}^*)$
1	Ackley	10	$-30 \leq x_i \leq 30$	0
2	Aluffi-Pentini	2	$-10 \leq x_i \leq 10$	-0.3523861
3	Becker and Lago	2	$-10 \leq x_i \leq 10$	0
4	Bohachevsky 1	2	$-50 \leq x_i \leq 50$	0
5	Bohachevsky 2	2	$-50 \leq x_i \leq 50$	0
6	Branin	2	$[-5, 0] \leq \mathbf{x} \leq [10, 15]$	3.926991
7	Camel 3	2	$-5 \leq x_i \leq 5$	0
8	Camel 6	2	$-5 \leq x_i \leq 5$	-1.031628
9	Cosine Mixture	4	$-1 \leq x_i \leq 1$	-0.4
10	Dekkers and Aarts	2	$-20 \leq x_i \leq 20$	-2.477289×10^4
11	Easom	2	$-10 \leq x_i \leq 10$	-1
12	Epistatic Michalewicz	5	$0 \leq x_i \leq \pi$	-4.687658
13	Exponential*	40	$-1 \leq x_i \leq 1$	-1
14	Goldstein and Price	2	$-2 \leq x_i \leq 2$	3
15	Griewank*	25	$-600 \leq x_i \leq 600$	0
16	Gulf Research	3	$[0.1, 0, 0] \leq \mathbf{x} \leq [100, 25.6, 5]$	0
17	Hartman 3	3	$0 \leq x_i \leq 1$	-3.862782
18	Hartman 6	6	$0 \leq x_i \leq 1$	-3.322368
19	Helical Valley	3	$-10 \leq x_i \leq 10$	0
20	Hosaki	2	$[0, 0] \leq \mathbf{x} \leq [5, 6]$	-2.345812
21	Kowalik	4	$0 \leq x_i \leq 0.42$	3.074871×10^{-4}
22	Levy and Montalvo 1	3	$-10 \leq x_i \leq 10$	0
23	Levy and Montalvo 2	10	$-5 \leq x_i \leq 5$	0
24	McCormick	2	$[-1.5, -3] \leq \mathbf{x} \leq [4, 3]$	-1.913223
25	Meyer and Roth	3	$-20 \leq x_i \leq 20$	4.355269×10^{-5}
26	Miele and Cantrell	4	$-1 \leq x_i \leq 1$	0
27	Modified Langerman	10	$0 \leq x_i \leq 10$	-0.965
28	Modified Rosenbrock	2	$-5 \leq x_i \leq 5$	0
29	Multi-Gaussian	2	$-2 \leq x_i \leq 2$	-1.296954
30	Neumaier 2	4	$0 \leq x_i \leq 4$	0
31	Neumaier 3*	30	$-900 \leq x_i \leq 900$	-4930
32	Odd Square	20	$-15 \leq x_i \leq 15$	-1
33	Paviani	10	$2.001 \leq x_i \leq 9.999$	-45.77845
34	Periodic	2	$-10 \leq x_i \leq 10$	0.9
35	Powell Quadratic	4	$-10 \leq x_i \leq 10$	0
36	Price Transistor Monitoring	9	$-10 \leq x_i \leq 10$	0
37	Rastrigin*	30	$-5.12 \leq x_i \leq 5.12$	0
38	Rosenbrock*	50	$-30 \leq x_i \leq 30$	0
39	Salomon*	50	$-100 \leq x_i \leq 100$	0
40	Schaffer 1	2	$-100 \leq x_i \leq 100$	0
41	Schaffer 2	2	$-100 \leq x_i \leq 100$	0
42	Schwefel*	40	$-500 \leq x_i \leq 500$	-1.675932×10^4
43	Shekel 10	4	$0 \leq x_i \leq 10$	-10.53628
44	Shekel 5	4	$0 \leq x_i \leq 10$	-10.15320
45	Shekel 7	4	$0 \leq x_i \leq 10$	-10.40282
46	Shekel Foxholes	10	$0 \leq x_i \leq 10$	-10.20879
47	Shubert	2	$-10 \leq x_i \leq 10$	-186.7309
48	Sinusoidal	20	$0 \leq x_i \leq 180$	-3.5
49	Storn Tchebyshev	9	$-256 \leq x_i \leq 256$	0
50	Wood	4	$-10 \leq x_i \leq 10$	0

*Table E.4. Details of global optimisation test problems, taken from [2]. Problems marked with * had $n = 10$ originally, but had their dimension increased here.*

E.5 CUTEst Large-Scale Nonlinear Least-Squares Collection

#	Problem	n	m	$2f(\mathbf{x}_0)$	$2f^*$	Parameters
1	ARGLALE	2000	4000	10000	2000	$(N, M) = (2000, 4000)$
2	ARGLBLE	2000	4000	8.545072×10^{22}	999.6250	$(N, M) = (2000, 4000)$
3	ARGTRIG	1000	1000	333.0006	0	$N = 1000$
4	ARTIF	5000	5000	1827.355	0	$N = 5000$
5	ARWHDNE	5000	9998	24995	1396.793	$N = 5000$
6	BDVALUES	1000	1000	1.996774×10^4	0	$NDP = 1002$
7	BRATU2D	4900	4900	3.085195×10^{-3}	0	$P = 72$
8	BRATU2DT	4900	4900	8.937521×10^{-3}	7.078014×10^{-11}	$P = 72$
9	BRATU3D	3375	3375	2.386977	0	$P = 17$
10	BROWNALE	1000	1000	2.502498×10^8	0	$N = 1000$
11	BROYDN3D	1000	1000	1011	0	$N = 1000$
12	BROYDNBD	5000	5000	124904	0	$N = 5000$
13	CBRATU2D	2888	2888	1.560446×10^{-2}	0	$P = 40$
14	CHANDHEQ	1000	1000	69.41682	0	$N = 1000$
15	EIGENB	2550	2550	99	0	$N = 50$
16	FREURONE	5000	9998	5.0485565×10^6	6.081592×10^5	$N = 5000$
17	INTEGREQ	1000	1000	5.678349	0	$N = 1000$
18	MOREBVNE	1000	1000	3.961509×10^{-6}	0	$N = 1000$
19	MSQRTA	4900	4900	7.975592×10^4	0	$P = 70$
20	MSQRTB	1024	1024	7926.444	0	$P = 32$
21	OSCIGRNE	1000	1000	6.120720×10^8	0	$N = 1000$
22	PENLT1NE	1000	1001	1.114448×10^{17}	9.686272×10^{-8}	$N = 1000$
23	POWELLSE	1000	1000	418750	0	$N = 1000$
24	SEMICN2U	1000	1000	1.960620×10^4	0	$(N, LN) = (1000, 900)$
25	SPMSQRT	1000	1664	797.0033	0	$M = 334$
26	VARDIMNE	1000	1002	1.241994×10^{22}	0	$N = 1000$
27	YATP1SQ	2600	2600	5.184111×10^7	0	$N = 50$
28	YATP2SQ	2600	2600	2.246192×10^7	0	$N = 50$

Table E.5. Details of large-scale test problems from the CUTEst test set (showing $2f(\mathbf{x}_0)$ and $2f^*$, as per Table E.1). The set of problems are taken from those in Table E.2; the relevant parameters yielding the given (n, m) are provided. The value of n shown excludes fixed variables.