A Derivative-Free Gauss-Newton Method

Coralia Cartis * Lindon Roberts †

29th October 2017

Abstract

We present DFO-GN, a derivative-free version of the Gauss-Newton method for solving nonlinear least-squares problems. As is common in derivative-free optimization, DFO-GN uses interpolation of function values to build a model of the objective, which is then used within a trust-region framework to give a globally-convergent algorithm requiring $\mathcal{O}(\epsilon^{-2})$ iterations to reach approximate first-order criticality within tolerance ϵ. This algorithm is a simplification of the method from [H. Zhang, A. R. Conn, and K. Scheinberg, *A Derivative-Free Algorithm for Least-Squares Minimization*, SIAM J. Optim., 20 (2010), pp. 3555–3576], where we replace quadratic models for each residual with linear models. We demonstrate that DFO-GN performs comparably to the method of Zhang et al. in terms of objective evaluations, as well as having a substantially faster runtime and improved scalability.

Keywords: derivative-free optimization, least-squares, Gauss-Newton method, trust region methods, global convergence, worst-case complexity.

Mathematics Subject Classification: 65K05, 90C30, 90C56

1 Introduction

Over the last 15–20 years, there has been a resurgence and increased effort devoted to developing efficient methods for derivative-free optimization (DFO) — that is, optimizing an objective using only function values. These methods are useful to many applications [7], for instance, when the objective function is a black-box function or legacy code (meaning manual computation of derivatives or algorithmic differentiation is impractical), has stochastic noise (so finite differencing is inaccurate) or expensive to compute (so the evaluation of a full n-dimensional gradient is intractable). There are several popular classes of DFO methods, such as direct and pattern search, model-based and evolutionary algorithms [22, 15, 26, 9]. Here, we consider model-based methods, which capture curvature in the objective well [9] and have been shown to have good practical performance [18].

Model-based methods typically use a trust-region framework for selecting new iterates, which ensures global convergence, provided we can build a sufficiently accurate model for the objective [3]. The model-building process most commonly uses interpolation of quadratic functions, as originally proposed by Winfield [33] and later developed by Conn, Scheinberg and Toint [8, 4] and Powell [21, 23]. Another common choice for model-building is to use radial basis functions [31, 20]. Global convergence results exist in both cases [6, 7, 32]. Several codes for model-based DFO are available, including those by Powell [35] and others (see e.g. [7, 9] and references therein).

Summary of contributions In this work, we consider nonlinear least-squares minimization, without constraints in the theoretical developments but allowing bounds in the implementation.

*Mathematical Institute, University of Oxford, Radcliffe Observatory Quarter, Woodstock Road, Oxford, OX2 6GG, United Kingdom (cartis@maths.ox.ac.uk).

†Mathematical Institute, University of Oxford, Radcliffe Observatory Quarter, Woodstock Road, Oxford, OX2 6GG, United Kingdom (robertsl@maths.ox.ac.uk). This work was supported by the EPSRC Centre For Doctoral Training in Industrially Focused Mathematical Modelling (EP/L015803/1) in collaboration with the Numerical Algorithms Group Ltd.

Model-based DFO is naturally suited to exploiting problem structure, and in this work we propose a method inspired by the classical Gauss-Newton method for derivative-based optimization (e.g. [19, Chapter 10]). This method, which we call DFO-GN (Derivative-Free Optimization using Gauss-Newton), is a simplification of the method by Zhang, Conn and Scheinberg [34]. It constructs linear interpolants for each residual, requiring exactly $n + 1$ points on each iteration, which is less than common proposals that generally insist on (partial or full) quadratic local models for each residual. In addition to proving theoretical guarantees for DFO-GN in terms of global convergence and worst-case complexity, we provide an implementation that is a modification of Powell's BOBYQA and that we extensively test and compare with existing state of the art DFO solvers. We show that little to nothing is lost by our simplified approach in terms of algorithm performance on a given evaluation budget, when applied to smooth and noisy, zero- and non-zero residual problems. Furthermore, significant gains are made in terms of reduced computational cost of the interpolation problem (leading to a runtime reduction of at least a factor of 7) and memory costs of storing the models. These savings result in a substantially faster runtime and improved scalability of DFO-GN compared to the implementation DFBOLS in [34].

Relevant existing literature. In [34], each residual function is approximated by a quadratic interpolating model, using function values from $p \in [n+1, (n+1)(n+2)/2]$ points. A quadratic (or higher-order) model for the overall least-squares objective is built from the models for each residual function, that takes into account full quadratic terms in the models asymptotically but allows the use of simpler models early on in the run of the algorithm. The DFBOLS implementation in [34] is shown to perform better than Powell's BOBYQA on a standard least-squares test set. A similar derivative-free framework for nonlinear least-squares problems is POUNDERS by Wild [30]: it also constructs quadratic interpolation models for each residual, but takes them all into account in the objective model construction on each iteration. In its implementation, it allows parallel computation of each residual component, and accepts previously-computed evaluations as an input providing extra information for the solver. We also note the connection to [2], which considers a Levenberg-Marquardt method for nonlinear least-squares when gradient evaluations are noisy; the framework is that of probabilistic local models, and it uses a regularization parameter rather than trust region to ensure global convergence. The algorithm is applied and further developed for data assimilation problems, with careful quantification of noise and algorithm parameters. Using linear vector models for objectives which are a composition of a (possibly nonconvex) vector function with a (possibly nonsmooth) convex function, such as a sum of squares, was also considered in [10]. There, worst-case complexity bounds for a general model-based trust-region DFO method applied to such objectives are established. Our approach differs in that it is designed specifically for nonlinear least-squares, and uses an algorithmic framework that is much closer to the software of Powell [27]. Finally, we note a mild connection to the approach in [1], where multiple solutions to nonlinear inverse problems are sought by means of a two-phase method, where in the first phase, low accuracy solutions are obtained by building a linear regression model from a (large) cloud of points and moving each point to its corresponding, slightly perturbed, Gauss-Newton step.

Further details of contributions. In terms of theoretical guarantees, we extend the global convergence results in [34] to allow inexact solutions to the trust-region subproblem (given by the usual Cauchy decrease condition), a simplification of the so-called 'criticality phase' and 'safety step', and prove first-order convergence of the whole sequence of iterates $\mathbf{x}_k$ rather than a subsequence. We also provide a worst-case complexity analysis and show an iteration count which matches that of Garmanjani, Júdice and Vicente [10], but with problem constants that correspond to second-order methods. This reflects the fact that we capture some of the curvature in the objective (since linear models for residuals still give an approximate quadratic model for the least-squares objective), and so the complexity of DFO-GN sits between first- and second-order methods.

In the DFO-GN implementation, which is very much the focus of this work, the simplification from quadratic to linear models leads to a confluence of two approaches for analysing and improving the geometry of the interpolation set. We compare DFO-GN to Powell's general DFO solver BOBYQA and to least-squares DFO solvers DFBOLS [34] (Fortran), POUNDERS [30] and our Python DFBOLS re-implementation Py-DFBOLS. The primary test set is Moré & Wild

[18] where additionally, we also consider noisy variants for each problem, perturbing the test set appropriately with unbiased Gaussian (multiplicative and additive), and with additive χ^2 noise; we solve to low as well as high accuracy requirements for a given evaluation budget. We find — and show by means of performance and data profiles — that DFO-GN performs comparably well in terms of objective evaluations to the best of solvers, albeit with a small penalty for objectives with additive stochastic noise and an even less penalty for nonzero-residuals. We then do a runtime comparison between DFO-GN and Py-DFBOLS on the same test set and settings, comparing like for like, and find that DFO-GN is at least 7 times faster; see Table 1 for details. We further investigate scalability features of DFO-GN. We compare memory requirements and runtime for DFO-GN and DFBOLS on a particular nonlinear equation problem from CUTEst with growing problem dimension n; we find that both of these increase much more rapidly for the latter than the former (for example, for $n = 2500$ DFO-GN's runtime is 2.5 times faster than the Fortran DFBOLS' for $n = 1400$). To further illustrate that the improved scalability of DFO-GN does not come at the cost of performance, we compare evaluation performance of DFO-GN and DFBOLS on 60 medium-size least-squares problems from CUTEst and find similarly good behaviour of DFO-GN as on the Moré & Wild set.

Implementation. Our Python implementation of DFO-GN is available on GitHub[1], and is released under the open-source GNU General Public License.

Structure of paper. In Section 2 we state the DFO-GN algorithm. We prove its global convergence to first-order critical points and worst case complexity in Section 3. Then we discuss the differences between this algorithm and its software implementation in Section 4. Lastly, in Section 5, we compare DFO-GN to other model-based derivative-free least-squares solvers on a selection of test problems, including noisy and higher-dimensional problems. We draw our conclusions in Section 6.

2 DFO-GN Algorithm

Here, our focus is unconstrained nonlinear least-squares minimization

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) := \frac{1}{2}\|\mathbf{r}(\mathbf{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\mathbf{x})^2, \tag{2.1}$$

where $\mathbf{r}(\mathbf{x}) := \begin{bmatrix} r_1(\mathbf{x}) & \cdots & r_m(\mathbf{x}) \end{bmatrix}^\top$ maps $\mathbb{R}^n \to \mathbb{R}^m$ and is continuously differentiable with $m \times n$ Jacobian matrix $[J(\mathbf{x})]_{i,j} = \frac{\partial r_i(\mathbf{x})}{\partial x_j}$, although these derivatives are not available. Typically $m \geq n$ in practice, but we do not require this for our method. Throughout, $\|\cdot\|$ refers to the Euclidean norm for vectors or largest singular value for matrices, unless otherwise stated, and we define $B(\mathbf{x}, \Delta) := \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{y} - \mathbf{x}\| \leq \Delta\}$ to be the closed ball of radius $\Delta > 0$ about $\mathbf{x} \in \mathbb{R}^n$.

In this section, we introduce the DFO-GN algorithm for solving (2.1) using linear interpolating models for $\mathbf{r}$.

2.1 Linear Residual Models

In the classical Gauss-Newton method, we approximate $\mathbf{r}$ in the neighbourhood of an iterate $\mathbf{x}_k$ by its linearization: $\mathbf{r}(\mathbf{y}) \approx \mathbf{r}(\mathbf{x}_k) + J(\mathbf{x}_k)(\mathbf{y} - \mathbf{x}_k)$, where $J(\mathbf{x}) \in \mathbb{R}^{m\times n}$ is the Jacobian matrix of first derivatives of $\mathbf{r}$. For DFO-GN, we use a similar approximation, but replace the Jacobian with an approximation to it calculated by interpolation.

Assume at iteration k we have a set of $n + 1$ interpolation points $Y_k := \{\mathbf{y}_0, \ldots, \mathbf{y}_n\}$ in $\mathbb{R}^n$ at which we have evaluated $\mathbf{r}$. This set always includes the current iterate; for simplicity of notation, we assume $\mathbf{y}_0 = \mathbf{x}_k$. We then build the model

$$\mathbf{r}(\mathbf{x}_k + \mathbf{s}) \approx \mathbf{m}_k(\mathbf{s}) := \mathbf{r}(\mathbf{x}_k) + J_k\mathbf{s}, \tag{2.2}$$

by finding the unique $J_k \in \mathbb{R}^{m\times n}$ satisfying the interpolation conditions

$$\mathbf{m}_k(\mathbf{y}_t - \mathbf{x}_k) = \mathbf{r}(\mathbf{y}_t), \qquad \text{for } t = 1, \ldots, n, \tag{2.3}$$

[1] `https://github.com/numericalalgorithmsgroup/dfogn`

noting that the other interpolation condition $\mathbf{m}_k(\mathbf{0}) = \mathbf{r}(\mathbf{x}_k)$ is automatically satisfied by (2.2)[2]. We can find J_k by solving the $n \times n$ system

$$\begin{bmatrix} (\mathbf{y}_1 - \mathbf{x}_k)^\top \\ \vdots \\ (\mathbf{y}_n - \mathbf{x}_k)^\top \end{bmatrix} \mathbf{j}_{k,i} = \begin{bmatrix} r_i(\mathbf{y}_1) - r_i(\mathbf{x}_k) \\ \vdots \\ r_i(\mathbf{y}_n) - r_i(\mathbf{x}_k) \end{bmatrix}, \tag{2.4}$$

for each $i = 1, \ldots, m$, where the rows of J_k are $\mathbf{j}_{k,i}^\top$. This system is invertible whenever the set of vectors $\{\mathbf{y}_1 - \mathbf{x}_k, \ldots, \mathbf{y}_n - \mathbf{x}_k\}$ is linearly independent. We ensure this in the algorithm by routines which improve the geometry of Y_k (in a specific sense to be discussed in Section 2.3).

Having constructed the linear models for each residual (2.1), we need to construct a model for the full objective f. To do this we simply take the sum of squares of the residual models, namely,

$$f(\mathbf{x}_k + \mathbf{s}) \approx m_k(\mathbf{s}) := \frac{1}{2}\|\mathbf{m}_k(\mathbf{s})\|^2 = f(\mathbf{x}_k) + \mathbf{g}_k^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top H_k \mathbf{s}, \tag{2.5}$$

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

2.2 Trust Region Framework

The DFO-GN algorithm is based on a trust-region framework [3]. In such a framework, we use our model for the objective (2.5), and maintain a parameter $\Delta_k > 0$ which characterizes the region in which we 'trust' our model to be a good approximation to the objective; the resulting 'trust region' is $B(\mathbf{x}_k, \Delta_k)$. At each iteration, we use our model to find a new point where we expect the objective to decrease, by (approximately) solving the 'trust region subproblem'

$$\mathbf{s}_k \approx \operatorname*{arg\,min}_{\|\mathbf{s}\| \leq \Delta_k} m_k(\mathbf{s}). \tag{2.6}$$

If this new point $\mathbf{x}_k + \mathbf{s}_k$ gives a sufficient objective reduction, we accept the step ($\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \mathbf{s}_k$), otherwise we reject the step ($\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k$). We also use this information to update the trust region radius Δ_k. The measure of 'sufficient objective reduction' is the ratio

$$r_k = \frac{\text{actual reduction}}{\text{predicted reduction}} := \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}. \tag{2.7}$$

This framework applies to both derivative-based and derivative-free settings. However in a DFO setting, we also need to update the interpolation set Y_k to incorporate the new point $\mathbf{x}_k + \mathbf{s}_k$, and have steps to ensure the geometry of Y_k does not become degenerate (see Section 2.3).

A minimal requirement on the calculation of $\mathbf{s}_k$ to ensure global convergence is the following.

Assumption 2.1. Our method for solving (2.6) gives a step $\mathbf{s}_k$ satisfying the sufficient ('Cauchy') decrease condition

$$m_k(\mathbf{0}) - m_k(\mathbf{s}_k) \geq c_1 \|\mathbf{g}_k\| \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)}\right), \tag{2.8}$$

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 [3]. In Zhang et al. [34], convergence is only proven when (2.6) is solved to full optimality; here we use this weaker assumption.

[2] We could have formulated a linear system to solve for the constant term in (2.2) as well as J_k, but this system becomes poorly conditioned as the algorithm progresses and the points Y_k get closer together.

2.3 Geometry Considerations

It is crucial that model-based DFO algorithms ensure the geometry of Y_k does not become degenerate; an example where ignoring geometry causes algorithm failure is given by Scheinberg and Toint [28].

To describe the notion of 'good' geometry, we need the Lagrange polynomials of Y_k. In our context of linear approximation, the Lagrange polynomials are the basis $\{\Lambda_0(\mathbf{x}), \ldots, \Lambda_n(\mathbf{x})\}$ for the $(n+1)$-dimensional space of linear functions on $\mathbb{R}^n$ defined by

$$\Lambda_l(\mathbf{y}_t) = \delta_{l,t}, \qquad \text{for all } l, t \in \{0, \ldots, n\}. \tag{2.9}$$

Such polynomials exist whenever the matrix in (2.4) is invertible [7, Lemma 3.2]; when this condition holds, we say that Y_k is *poised* for linear interpolation.

The notion of geometry quality is then given by the following [5].

Definition 2.2 (Λ-poised)**.** Suppose Y_k is poised for linear interpolation. Let $B \subset \mathbb{R}^n$ be some set, and $\Lambda \geq 1$. Then we say that Y_k is Λ-poised in B if $Y_k \subset B$ and

$$\max_{t=0,\ldots,n} \max_{\mathbf{x} \in B} |\Lambda_t(\mathbf{x})| \leq \Lambda, \tag{2.10}$$

where $\{\Lambda_0(\mathbf{x}), \ldots, \Lambda_n(\mathbf{x})\}$ are the Lagrange polynomials for Y_k.

In general, if Y_k is Λ-poised with a small Λ, then Y_k has 'good' geometry, in the sense that linear interpolation using points Y_k produces a more accurate model. The notion of model accuracy we use is given in [5, 6]:

Definition 2.3 (Fully linear, scalar function)**.** A model $m_k \in C^1$ for $f \in C^1$ is *fully linear* in $B(\mathbf{x}_k, \Delta_k)$ if

$$|f(\mathbf{x}_k + \mathbf{s}) - m_k(\mathbf{s})| \leq \kappa_{ef} \Delta_k^2, \tag{2.11}$$

$$\|\nabla f(\mathbf{x}_k + \mathbf{s}) - \nabla m_k(\mathbf{s})\| \leq \kappa_{eg} \Delta_k, \tag{2.12}$$

for all $\|\mathbf{s}\| \leq \Delta_k$, where κ_{ef} and κ_{eg} are independent of $\mathbf{s}$, $\mathbf{x}_k$ and Δ_k.

In the case of a vector model, such as (2.1), we use an analogous definition as in [13], which is equivalent, up to a change in constants, to the definition in [10].

Definition 2.4 (Fully linear, vector function)**.** A vector model $\mathbf{m}_k \in C^1$ for $\mathbf{r} \in C^1$ is fully linear in $B(\mathbf{x}_k, \Delta_k)$ if

$$\|\mathbf{r}(\mathbf{x}_k + \mathbf{s}) - \mathbf{m}_k(\mathbf{s})\| \leq \kappa_{ef}^r \Delta_k^2, \tag{2.13}$$

$$\|J(\mathbf{x}_k + \mathbf{s}) - J^m(\mathbf{s})\| \leq \kappa_{eg}^r \Delta_k, \tag{2.14}$$

for all $\|\mathbf{s}\| \leq \Delta_k$, where J^m is the Jacobian of $\mathbf{m}_k$, and κ_{ef}^r and κ_{eg}^r are independent of $\mathbf{s}$, $\mathbf{x}_k$ and Δ_k.

In Section 3.1, we show that if Y_k is Λ-poised, then $\mathbf{m}_k$ (2.1) and m_k (2.5) are fully linear in $B(\mathbf{x}_k, \Delta_k)$, with constants that depend on Λ.

2.4 Full Algorithm Specification

A full description of the DFO-GN algorithm is provided in Algorithm 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 Δ_k, as well as $\|\mathbf{g}_k\|$, a good measure of progress towards optimality. After computing the trust region step $\mathbf{s}_k$, we then apply a 'safety phase', also originally from Powell [24]. In this phase, we check if $\|\mathbf{s}_k\|$ is too small compared to the lower bound ρ_k on the trust-region radius, 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 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 which 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-GN, due to Powell [24], is that it maintains not only the (usual) trust region radius Δ_k (used in (2.6) 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.

Algorithm 1 DFO-GN: Derivative-Free Optimization using Gauss-Newton.

Input: Starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and initial trust region radius $\Delta_0^{init} > 0$.
Parameters are $\Delta_{max} \geq \Delta_0^{init}$, criticality threshold $\epsilon_C > 0$, criticality scaling $\mu > 0$, trust region radius scalings $0 < \gamma_{dec} < 1 < \gamma_{inc} \leq \overline{\gamma}_{inc}$ 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 $\Lambda \geq 1$.

1: Build an initial interpolation set Y_0 of size $n + 1$, with $\mathbf{x}_0 \in Y_0$. Set $\rho_0^{init} = \Delta_0^{init}$.
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Given $\mathbf{x}_k$ and Y_k, solve the interpolation problem (2.4) and form m_k^{init} (2.5).
4: **if** $\|\mathbf{g}_k^{init}\| \leq \epsilon_C$ **then**
5: Criticality Phase: using Algorithm 2 (Appendix B), modify Y_k and find $\Delta_k \leq \Delta_k^{init}$ 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 $\rho_k = \min(\rho_k^{init}, \Delta_k)$.
6: **else**
7: Set $m_k = m_k^{init}$, $\Delta_k = \Delta_k^{init}$ and $\rho_k = \rho_k^{init}$.
8: **end if**
9: Approximately solve the trust region subproblem (2.6) to get step $\mathbf{s}_k$ satisfying Assumption 2.1.
10: **if** $\|\mathbf{s}_k\| < \gamma_S \rho_k$ **then**
11: Safety Phase: Set $\mathbf{x}_{k+1} = \mathbf{x}_k$ and $\Delta_{k+1}^{init} = \max(\rho_k, \omega_S \Delta_k)$, and form Y_{k+1} by making Y_k Λ-poised in $B(\mathbf{x}_{k+1}, \Delta_{k+1}^{init})$.
12: If $\Delta_{k+1}^{init} = \rho_k$, set $(\rho_{k+1}^{init}, \Delta_{k+1}^{init}) = (\alpha_1 \rho_k, \alpha_2 \rho_k)$, otherwise set $\rho_{k+1}^{init} = \rho_k$.
13: **goto** line 3.
14: **end if**
15: Calculate ratio r_k (2.7).
16: Accept/reject step and update trust region radius: set

$$\mathbf{x}_{k+1} = \begin{cases} \mathbf{x}_k + \mathbf{s}_k, & r_k \geq \eta_1, \\ \mathbf{x}_k, & r_k < \eta_1, \end{cases} \quad \text{and} \quad \Delta_{k+1}^{init} = \begin{cases} \min(\max(\gamma_{inc}\Delta_k, \overline{\gamma}_{inc}\|\mathbf{s}_k\|), \Delta_{max}), & r_k \geq \eta_2, \\ \max(\gamma_{dec}\Delta_k, \|\mathbf{s}_k\|, \rho_k), & \eta_1 \leq r_k < \eta_2, \\ \max(\min(\gamma_{dec}\Delta_k, \|\mathbf{s}_k\|), \rho_k), & r_k < \eta_1. \end{cases} \tag{2.15}$$

17: **if** $r_k \geq \eta_1$ **then**
18: Form $Y_{k+1} = Y_k \cup \{\mathbf{x}_{k+1}\} \setminus \{\mathbf{y}_t\}$ for some $\mathbf{y}_t \in Y_k$ and set $\rho_{k+1}^{init} = \rho_k$.
19: **else if** Y_k is not Λ-poised in $B(\mathbf{x}_k, \Delta_k)$ **then**
20: Model Improvement Phase: Form Y_{k+1} by making Y_k Λ-poised in $B(\mathbf{x}_{k+1}, \Delta_{k+1}^{init})$ and set $\rho_{k+1}^{init} = \rho_k$.
21: **else** [$r_k < \eta_1$ **and** Y_k is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$]
22: Unsuccessful Phase: Set $Y_{k+1} = Y_k$, and if $\Delta_{k+1}^{init} = \rho_k$, set $(\rho_{k+1}^{init}, \Delta_{k+1}^{init}) = (\alpha_1 \rho_k, \alpha_2 \rho_k)$, otherwise set $\rho_{k+1}^{init} = \rho_k$.
23: **end if**
24: **end for**

Remark 2.5. There are two different geometry-improving phases in Algorithm 1. The first modifies Y_k to ensure it is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$, and is called in the safety and model improvement phases.

This can be achieved by [7, Algorithm 6.3], for instance, where the number of interpolation systems (2.4) to be solved depends only on Λ and n [7, Theorem 6.3].

The second, called in the criticality phase, also ensures Y_k is Λ-poised, but it also modifies Δ_k to ensure $\Delta_k \leq \mu\|\mathbf{g}_k\|$. This is a more complicated procedure [7, Algorithm 10.2], as we have a coupling between Δ_k and Y_k: ensuring Λ-poisedness in $B(\mathbf{x}_k, \Delta_k)$ depends on Δ_k, but since $\mathbf{g}_k$ depends on Y_k, there is a dependency of Δ_k on Y_k. Full details of how to achieve this are given in Appendix B — we show that this procedure terminates as long as $\|\nabla f(\mathbf{x}_k)\| \neq 0$. In addition, there, we also prove the bound

$$\min\left(\Delta_k^{init}, \text{const} \cdot \|\nabla f(\mathbf{x}_k)\|\right) \leq \Delta_k \leq \Delta_k^{init}. \tag{2.16}$$

If the procedure terminates in one iteration, then $\Delta_k = \Delta_k^{init}$, and we have simply made Y_k Λ-poised, just as in the model-improving phase. Otherwise, we do one of these model-improving iterations, then several iterations where both Δ_k is reduced and Y_k is made Λ-poised. The bound (2.16) tells us that these unsuccessful-type iterations do not occur when Δ_k^{init} (but not $\nabla f(\mathbf{x}_k)$) is sufficiently small.[3]

Remark 2.6. In Lemma 3.4, 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 1. Any strategy which achieves fully linear models would be sufficient for the convergence results in Section 3.4.

Remark 2.7. There are several differences between Algorithm 1 and its implementation, which we fully detail in Section 4. In particular, there is no criticality phase in the DFO-GN implementation as we found it is not needed, but the safety step is preserved to keep continuity with the BOBYQA framework[4]; also, the geometry-improving phases are replaced by a simplified calculation.

3 Convergence and complexity results

We first outline the connection between Λ-poisedness of Y_k and fully linear models. We then prove global convergence of Algorithm 1 (i.e. convergence from any starting point $\mathbf{x}_0$) to first-order critical points, and determine its worst-case complexity.

3.1 Interpolation Models are Fully Linear

To begin, we require some assumptions on the smoothness of $\mathbf{r}$.

Assumption 3.1. The function $\mathbf{r}$ is C^1 and its Jacobian $J(\mathbf{x})$ is Lipschitz continuous in $\mathcal{B}$, the convex hull of $\cup_k B(\mathbf{x}_k, \Delta_{max})$, with constant L_J. 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 $\mathbf{x} \in \mathcal{B}$.

Remark 3.2. If the level set $\mathcal{L} := \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}_0)\}$ is bounded, which is assumed in [34], then $\mathbf{x}_k \in \mathcal{L}$ for all k, so $\mathcal{B}$ is compact, from which Assumption 3.1 follows.

Lemma 3.3. *If Assumption 3.1 holds, then ∇f is Lipschitz continuous in $\mathcal{B}$ with constant*

$$L_{\nabla f} := r_{max} L_J + J_{max}^2. \tag{3.1}$$

Proof. We choose $\mathbf{x}, \mathbf{y} \in \mathcal{B}$ and use the Fundamental Theorem of Calculus to compute

$$\|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x})\| = \left\| \int_0^1 J(\mathbf{x} + \alpha(\mathbf{y} - \mathbf{x}))(\mathbf{y} - \mathbf{x}) d\alpha \right\| \leq J_{max} \|\mathbf{y} - \mathbf{x}\|. \tag{3.2}$$

[3] The more common approach in the criticality phase (e.g. [7, 10, 34]) is to use an extra parameter $0 < \beta < \mu$ and floor Δ_k at $\beta\|\mathbf{g}_k\|$, maintaining full linearity with extra assumptions on κ_{ef} and κ_{eg} [6, Lemma 3.2], and requiring all fully linear models have Lipschitz continuous gradient with uniformly bounded Lipschitz constant.

[4] Note that the criticality and safety phases have similar aims, namely, to keep the approximate gradient and the step proportional to Δ_k. However, showing global convergence/complexity without a criticality step is unprecedented in the literature, and left for future work.

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

$$\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\| \leq \|(J(\mathbf{y}) - J(\mathbf{x}))^\top \mathbf{r}(\mathbf{y})\| + \|J(\mathbf{x})^\top (\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x}))\|, \tag{3.3}$$

$$\leq \|J(\mathbf{y}) - J(\mathbf{x})\| \cdot \|\mathbf{r}(\mathbf{y})\| + \|J(\mathbf{x})\| \cdot \|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x})\|, \tag{3.4}$$

$$\leq L_J \|\mathbf{y} - \mathbf{x}\| \cdot r_{max} + J_{max} \cdot J_{max} \|\mathbf{y} - \mathbf{x}\|, \tag{3.5}$$

from which we recover (3.1). □

We now state the connection between Λ-poisedness of Y_k and full linearity of the models $\mathbf{m}_k$ (2.1) and m_k (2.5).

Lemma 3.4. *Suppose Assumption 3.1 holds and Y_k is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$. Then $\mathbf{m}_k$ (2.1) is a fully linear model for $\mathbf{r}$ in $B(\mathbf{x}_k, \Delta_k)$ in the sense of Definition 2.4 with constants*

$$\kappa_{ef}^r = \kappa_{eg}^r + \frac{L_J}{2} \qquad \text{and} \qquad \kappa_{eg}^r = \frac{1}{2} L_J \left(\sqrt{n} C + 2\right), \tag{3.6}$$

in (2.13) and (2.14), where $C = \mathcal{O}(\Lambda)$. Under the same hypotheses, m_k (2.5) is a fully linear model for f in $B(\mathbf{x}_k, \Delta_k)$ in the sense of Definition 2.3 with constants

$$\kappa_{ef} = \kappa_{eg} + \frac{L_{\nabla f} + (\kappa_{eg}^r \Delta_{max} + J_{max})^2}{2} \text{ and } \kappa_{eg} = L_{\nabla f} + \kappa_{eg}^r r_{max} + (\kappa_{eg}^r \Delta_{max} + J_{max})^2, \tag{3.7}$$

in (2.11) and (2.12), where $L_{\nabla f}$ is from (3.1). We also have the bound $\|H_k\| \leq (\kappa_{eg}^r \Delta_{max} + J_{max})^2$, independent of $\mathbf{x}_k$, Y_k and Δ_k.

Proof. See Appendix A. □

3.2 Global Convergence of DFO-GN

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.

The results below are largely based on corresponding results in [34, 7].

Assumption 3.5. We assume that $\|H_k\| \leq \kappa_H$ for all k, for some $\kappa_H \geq 1$[5].

Lemma 3.6. *Suppose Assumption 2.1 holds. If the model m_k is fully linear in $B(\mathbf{x}_k, \Delta_k)$ and*

$$\Delta_k \leq \min\left(\frac{c_1(1-\eta_2)\|\mathbf{g}_k\|}{2\kappa_{ef}}, \frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)}\right), \tag{3.8}$$

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

Proof. We compute

$$|r_k - 1| = \left|\frac{(f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{s}_k)) - (m_k(\mathbf{0}) - m_k(\mathbf{s}_k))}{m_k(\mathbf{0}) - m_k(\mathbf{s}_k)}\right|, \tag{3.9}$$

$$\leq \frac{|f(\mathbf{x}_k + \mathbf{s}_k) - m_k(\mathbf{s}_k)|}{|m_k(\mathbf{0}) - m_k(\mathbf{s}_k)|} + \frac{|f(\mathbf{x}_k) - m_k(\mathbf{0})|}{|m_k(\mathbf{0}) - m_k(\mathbf{s}_k)|}. \tag{3.10}$$

[5]Lemma 3.4 ensures Assumption 3.5 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 Section 3.3.1 for details.

By assumption, $\Delta_k \leq \|\mathbf{g}_k\|/\max(\|H_k\|, 1)$. Applying this to (2.8), we have

$$m_k(\mathbf{0}) - m_k(\mathbf{s}_k) \geq c_1 \|\mathbf{g}_k\| \Delta_k. \tag{3.11}$$

Using this and fully linearity (2.11), we get

$$|r_k - 1| \leq 2\left(\frac{\kappa_{ef}\Delta_k^2}{c_1\|\mathbf{g}_k\|\Delta_k}\right) \leq 1 - \eta_2. \tag{3.12}$$

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.

Lemma 3.7. *Suppose Assumption 2.1 holds. Then the step $\mathbf{s}_k$ satisfies*

$$\|\mathbf{s}_k\| \geq \frac{2c_1}{1+\sqrt{1+2c_1}} \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)}\right). \tag{3.13}$$

Proof. For convenience of notation, let $h_k := \max(\|H_k\|, 1) \geq 1$. Since $m_k(\mathbf{0}) - m_k(\mathbf{s}_k) \geq 0$ from (2.8), we have

$$m_k(\mathbf{0}) - m_k(\mathbf{s}_k) = |m_k(\mathbf{0}) - m_k(\mathbf{s}_k)| = \left|\mathbf{g}_k^\top \mathbf{s}_k + \frac{1}{2}\mathbf{s}_k^\top H_k \mathbf{s}_k\right| \leq \|\mathbf{s}_k\| \cdot \|\mathbf{g}_k\| + \frac{h_k}{2}\|\mathbf{s}_k\|^2. \tag{3.14}$$

Substituting this into (2.8), we get

$$\frac{1}{2}\|\mathbf{s}_k\|^2 + \frac{\|\mathbf{g}_k\|}{h_k} \cdot \|\mathbf{s}_k\| - c_1 \frac{\|\mathbf{g}_k\|}{h_k} \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{h_k}\right) \geq 0. \tag{3.15}$$

For this to be satisfied, we require

$$\|\mathbf{s}_k\| \geq \sqrt{\frac{\|\mathbf{g}_k\|^2}{h_k^2} + 2c_1 \frac{\|\mathbf{g}_k\|}{h_k} \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{h_k}\right)} - \frac{\|\mathbf{g}_k\|}{h_k}, \tag{3.16}$$

$$= \frac{2c_1 \frac{\|\mathbf{g}_k\|}{h_k} \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{h_k}\right)}{\sqrt{\frac{\|\mathbf{g}_k\|^2}{h_k^2} + 2c_1 \frac{\|\mathbf{g}_k\|}{h_k} \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{h_k}\right)} + \frac{\|\mathbf{g}_k\|}{h_k}}, \tag{3.17}$$

$$\geq \frac{2c_1 \frac{\|\mathbf{g}_k\|}{h_k} \min\left(\Delta_k, \frac{\|\mathbf{g}_k\|}{h_k}\right)}{\sqrt{\frac{\|\mathbf{g}_k\|^2}{h_k^2} + 2c_1 \frac{\|\mathbf{g}_k\|}{h_k} \left(\frac{\|\mathbf{g}_k\|}{h_k}\right)} + \frac{\|\mathbf{g}_k\|}{h_k}}, \tag{3.18}$$

from which we recover (3.13). □

Lemma 3.8. *In all iterations, $\|\mathbf{g}_k\| \geq \min(\epsilon_C, \Delta_k/\mu)$. Also, if $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$ then*

$$\|\mathbf{g}_k\| \geq \epsilon_g := \min\left(\epsilon_C, \frac{\epsilon}{1+\kappa_{eg}\mu}\right) > 0. \tag{3.19}$$

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

To show (3.19), first suppose $\|\mathbf{g}_k^{init}\| \geq \epsilon_C$. Then $\mathbf{g}_k = \mathbf{g}_k^{init}$ and (3.19) 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

$$\epsilon \leq \|\nabla f(\mathbf{x}_k)\| \leq \|\nabla f(\mathbf{x}_k) - \mathbf{g}_k\| + \|\mathbf{g}_k\| \leq \kappa_{eg}\mu\|\mathbf{g}_k\| + \|\mathbf{g}_k\|, \tag{3.20}$$

and so $\|\mathbf{g}_k\| \geq \epsilon/(1+\kappa_{eg}\mu)$ and (3.19) holds. □

Lemma 3.9. *Suppose Assumptions 2.1, 3.1 and 3.5 hold. If $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$ for all k, then $\rho_k \geq \rho_{min} > 0$ for all k, where*

$$\rho_{min} := \min\left(\Delta_0^{init}, \frac{\omega_C \epsilon}{\kappa_{eg} + 1/\mu}, \frac{\alpha_1 \epsilon_g}{\kappa_H}, \alpha_1 \left(\kappa_{eg} + \frac{2\kappa_{ef}}{c_1(1-\eta_2)}\right)^{-1} \epsilon\right). \tag{3.21}$$

Proof. From Lemma 3.8, we also have $\|\mathbf{g}_k\| \geq \epsilon_g > 0$ for all k. To find a contradiction, let $k(0)$ be the first k such that $\rho_k < \rho_{min}$. That is, we have

$$\rho_0^{init} \geq \rho_0 \geq \rho_1^{init} \geq \rho_1 \geq \cdots \geq \rho_{k(0)-1}^{init} \geq \rho_{k(0)-1} \geq \rho_{min} \qquad \text{and} \qquad \rho_{k(0)} < \rho_{min}. \tag{3.22}$$

We first show that

$$\rho_{k(0)} = \rho_{k(0)}^{init} < \rho_{min}. \tag{3.23}$$

From Algorithm 1, we know that either $\rho_{k(0)} = \rho_{k(0)}^{init}$ or $\rho_{k(0)} = \Delta_{k(0)}$. Hence we must either have $\rho_{k(0)}^{init} < \rho_{min}$ or $\Delta_{k(0)} < \rho_{min}$. In the former case, there is nothing to prove; in the latter, using Lemma B.1, we have that

$$\rho_{min} > \Delta_{k(0)} \geq \min\left(\Delta_{k(0)}^{init}, \frac{\omega_C \epsilon}{\kappa_{eg} + 1/\mu}\right) \geq \min\left(\rho_{k(0)}^{init}, \frac{\omega_C \epsilon}{\kappa_{eg} + 1/\mu}\right). \tag{3.24}$$

Since $\rho_{min} \leq \omega_C \epsilon/(\kappa_{eg} + 1/\mu)$, we therefore conclude that (3.23) holds.

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

$$\|\mathbf{s}_{k(0)-1}\| \leq \min(\gamma_S, \gamma_{dec}^{-1})\rho_{k(0)-1} \leq \frac{1}{\alpha_1}\min(\gamma_S, \gamma_{dec}^{-1})\rho_{min} = \frac{\gamma_S}{\alpha_1}\rho_{min}, \tag{3.25}$$

since $\gamma_S < 1$ and $\gamma_{dec} < 1$. Hence by Lemma 3.7 we have

$$c_2 \min\left(\Delta_{k(0)-1}, \frac{\epsilon_g}{\kappa_H}\right) \leq \|\mathbf{s}_{k(0)-1}\| \leq \frac{\gamma_S}{\alpha_1}\rho_{min}, \tag{3.26}$$

where $c_2 := 2c_1/(1 + \sqrt{1 + 2c_1})$. 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 1. This inequality and the choice of γ_S, together with (3.26), also imply

$$\Delta_{k(0)-1} \leq \frac{\gamma_S \rho_{min}}{\alpha_1 c_2} < \frac{\rho_{min}}{\alpha_1} \leq \min\left(\frac{\epsilon_g}{\kappa_H}, \left(\kappa_{eg} + \frac{2\kappa_{ef}}{c_1(1-\eta_2)}\right)^{-1} \epsilon\right). \tag{3.27}$$

Then since $\Delta_{k_{(0)}-1} \leq \epsilon_g/\kappa_H$, Lemma 3.7 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 $\rho_{k_{(0)}}^{init} = \rho_{k_{(0)}-1}$, contradicting (3.23). Thus m_k must be fully linear. Now suppose that

$$\Delta_{k_{(0)}-1} > \frac{c_1(1-\eta_2)\|\mathbf{g}_{k_{(0)}-1}\|}{2\kappa_{ef}}. \tag{3.28}$$

Then using full linearity, we have

$$\epsilon \leq \|\nabla f(\mathbf{x}_{k_{(0)}-1})\| \leq \kappa_{eg}\Delta_{k(0)-1} + \|\mathbf{g}_{k_{(0)}-1}\| < \left(\kappa_{eg} + \frac{2\kappa_{ef}}{c_1(1-\eta_2)}\right)\Delta_{k_{(0)}-1}. \tag{3.29}$$

contradicting (3.27). That is, (3.28) is false and so together with (3.27), we have (3.8). Hence Lemma 3.6 implies iteration $(k_0 - 1)$ was very successful (as we have already established the safety phase was not called), so $\rho_{k_{(0)}}^{init} = \rho_{k_{(0)}-1}$, contradicting (3.23). □

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

Lemma 3.10. *Suppose Assumptions 2.1, 3.1 and 3.5 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. 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 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 $\max(\gamma_{dec}, \alpha_2, \omega_S) < 1$, 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

$$\|\nabla f(\mathbf{x}_k)\| \leq \|\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}_{j_k})\| + \|\nabla f(\mathbf{x}_{j_k}) - \mathbf{g}_{j_k}\| + \|\mathbf{g}_{j_k}\|. \tag{3.30}$$

As $k \to \infty$, the first term of the right-hand side of (3.30) is bounded by $L_{\nabla f}\Delta_k \to 0$, while the second term is bounded by $\kappa_{eg}\Delta_{j_k} \to 0$; 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 3.6 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 3.7 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 3.11. *Suppose Assumptions 2.1, 3.1 and 3.5 hold. Then* $\lim_{k\to\infty} \Delta_k = 0$ *and so* $\lim_{k\to\infty} \rho_k = 0$.

Proof. If $|\mathcal{S}| < \infty$, the proof of Lemma 3.10 gives the result. Thus, suppose there are infinitely many successful iterations (i.e. $|\mathcal{S}| = \infty$).

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

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 \left(m_k(\mathbf{0}) - m_k(\mathbf{s}_k)\right) \geq \eta_1 c_1 \|\mathbf{g}_k\| \min\left(\frac{\|\mathbf{g}_k\|}{\kappa_H}, \Delta_k\right) > 0. \tag{3.31}$$

But since $\|\mathbf{g}_k\| \geq \min(\epsilon_C, \Delta_k/\mu)$ (see Lemma 3.8), this means that

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 c_1 \min(\epsilon_C, \mu^{-1}\Delta_k) \min\left(\frac{\min(\epsilon_C, \mu^{-1}\Delta_k)}{\kappa_H}, \Delta_k\right) > 0. \tag{3.32}$$

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 $\overline{\gamma}_{inc}$. 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 $\Delta_k \leq \overline{\gamma}_{inc}\Delta_{j_k} \to 0$. Lastly, $\rho_k \to 0$ since $\rho_k \leq \Delta_k$ throughout the algorithm. □

Theorem 3.12. *Suppose Assumptions 2.1, 3.1 and 3.5 hold. Then*

$$\liminf_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0. \tag{3.33}$$

Proof. If $|\mathcal{S}| < \infty$, then this follows from Lemma 3.10. Otherwise, it follows from Lemma 3.11 and Lemma 3.9. □

Theorem 3.13. *Suppose Assumptions 2.1, 3.1 and 3.5 hold. Then* $\lim_{k\to\infty} \|\nabla f(\mathbf{x}_k)\| = 0$.

Proof. If $|\mathcal{S}| < \infty$, then the result follows from Lemma 3.10. Thus, suppose there are infinitely many successful iterations (i.e. $|\mathcal{S}| = \infty$).

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 3.8, we must have $\|\mathbf{g}_{t_j}\| \geq \epsilon > 0$ for some ϵ, where without loss of generality we assume that

$$\epsilon < \min\left(\epsilon_C, \frac{\epsilon_0}{2 + \kappa_{eg}\mu}\right). \tag{3.34}$$

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 3.12. That is, there exist subsequences $t_j < \ell_j$ satisfying

$$\|\mathbf{g}_k\| \geq \epsilon \quad \text{for } k = t_j, \ldots, \ell_j - 1, \text{ and } \quad \|\mathbf{g}_{\ell_j}\| < \epsilon. \tag{3.35}$$

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 3.11), Lemma 3.6 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 3.10, since $\|\mathbf{g}_k\| \geq \epsilon$, for $k \in \mathcal{K}$ sufficiently large, Lemma 3.7 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 (2.8)

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1(m_k(\mathbf{0}) - m_k(\mathbf{s}_k)) \geq \eta_1 c_1 \epsilon \min\left(\frac{\|\mathbf{g}_k\|}{\kappa_H}, \Delta_k\right) > 0, \tag{3.36}$$

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

$$\Delta_k \leq \frac{f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})}{\eta_1 c_1 \epsilon}. \tag{3.37}$$

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)

$$\|\mathbf{x}_{\ell_j} - \mathbf{x}_{t_j}\| \leq \sum_{k=t_j, k\in\mathcal{K}\cap\mathcal{S}}^{\ell_j - 1} \|\mathbf{x}_k - \mathbf{x}_{k+1}\| \leq \sum_{k=t_j, k\in\mathcal{K}\cap\mathcal{S}}^{\ell_j - 1} \Delta_k \leq \frac{f(\mathbf{x}_{t_j}) - f(\mathbf{x}_{\ell_j})}{\eta_1 c_1 \epsilon}. \tag{3.38}$$

Since $\{f(\mathbf{x}_k) : k \in \mathcal{K}\}$ is a monotone decreasing sequence by (3.36), 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

$$\|\nabla f(\mathbf{x}_{t_j})\| \leq \|\nabla f(\mathbf{x}_{t_j}) - \nabla f(\mathbf{x}_{\ell_j})\| + \|\nabla f(\mathbf{x}_{\ell_j}) - \mathbf{g}_{\ell_j}\| + \|\mathbf{g}_{\ell_j}\|. \tag{3.39}$$

Similarly to Lemma 3.10, 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 $\kappa_{eg}\Delta_{\ell_j} \leq \kappa_{eg}\mu\epsilon$. Lastly, the third term is bounded by ϵ by definition of ℓ_j.

All together, this means that for sufficiently large j,

$$\|\nabla f(\mathbf{x}_{t_j})\| \leq \epsilon + \kappa_{eg}\mu\epsilon + \epsilon = (2 + \kappa_{eg}\mu)\epsilon < \epsilon_0, \tag{3.40}$$

and we have our contradiction. □

3.3 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 3.12. Let i_ϵ be the last iteration before $\|\nabla f(\mathbf{x}_{i_\epsilon+1})\| < \epsilon$ for the first time.

Lemma 3.14. *Suppose Assumptions 2.1, 3.1 and 3.5 hold. Let $|\mathcal{S}_{i_\epsilon}|$ be the number of successful steps up to iteration i_ϵ. Then*

$$|\mathcal{S}_{i_\epsilon}| \leq \frac{f(\mathbf{x}_0)}{\eta_1 c_1} \max\left(\kappa_H \epsilon_g^{-2}, \epsilon_g^{-1} \rho_{min}^{-1}\right), \tag{3.41}$$

where ϵ_g is defined in (3.19)*, and ρ_{min} in* (3.21)*.*

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

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 \left(m_k(\mathbf{0}) - m_k(\mathbf{s}_k)\right) \geq \eta_1 c_1 \|\mathbf{g}_k\| \min\left(\frac{\|\mathbf{g}_k\|}{\kappa_H}, \Delta_k\right). \tag{3.42}$$

Since $\|\mathbf{g}_k\| \geq \epsilon_g$ from Lemma 3.8 and $\Delta_k \geq \rho_k \geq \rho_{min}$ from Lemma 3.9, this means

$$f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}) \geq \eta_1 c_1 \epsilon_g \min\left(\frac{\epsilon_g}{\kappa_H}, \rho_{min}\right). \tag{3.43}$$

Summing (3.43) over all $k \in \mathcal{S}_{i_\epsilon}$, and noting that $0 \leq f(\mathbf{x}_k) \leq f(\mathbf{x}_0)$, we get

$$f(\mathbf{x}_0) \geq |\mathcal{S}_{i_\epsilon}| \eta_1 c_1 \epsilon_g \min\left(\frac{\epsilon_g}{\kappa_H}, \rho_{min}\right), \tag{3.44}$$

from which (3.41) follows. □

We now need to count the number of iterations of Algorithm 1 which are not successful. Following [10], we count each iteration of the loop inside the criticality phase (Algorithm 2) as a separate iteration — in effect, one 'iteration' corresponds to one construction of the model m_k (2.5). We also consider separately the number of criticality phases for which Δ_k is not reduced (i.e. $\Delta_k = \Delta_k^{init}$). 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 — see Remark 2.5 for further details);
- $\mathcal{C}_{i_\epsilon}^U$ 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);
- $\mathcal{F}_{i_\epsilon}$ 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[6].

Lemma 3.15. *Suppose Assumptions 2.1, 3.1 and 3.5 hold. Then we have the bounds*

$$|\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}| \leq |\mathcal{S}_{i_\epsilon}| \cdot \frac{\log \overline{\gamma}_{inc}}{|\log \alpha_3|} + \frac{1}{|\log \alpha_3|} \log\left(\frac{\Delta_0^{init}}{\rho_{min}}\right), \tag{3.45}$$

$$|\mathcal{C}_{i_\epsilon}^M| \leq |\mathcal{F}_{i_\epsilon}| + |\mathcal{S}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}|, \tag{3.46}$$

$$|\mathcal{M}_{i_\epsilon}| \leq |\mathcal{C}_{i_\epsilon}^M| + |\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{S}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}|, \tag{3.47}$$

where $\alpha_3 := \max(\omega_C, \omega_S, \gamma_{dec}, \alpha_2) < 1$ and ρ_{min} is defined in (3.21)*.*

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

[6]Note that the analysis in [13] bounds the number of outer iterations of Algorithm 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 [10] and than here.

at least $\max(\gamma_{dec}, \alpha_2)$. On each successful iteration, we increase Δ_k by a factor of at most $\overline{\gamma}_{inc}$, and on all other iterations, Δ_k is either constant or reduced. Therefore, we must have

$$\rho_{min} \leq \Delta_{i_\epsilon} \leq \Delta_0^{init} \cdot \omega_C^{|\mathcal{C}_{i_\epsilon}^U|} \cdot \max(\omega_S, \alpha_2)^{|\mathcal{F}_{i_\epsilon}|} \cdot \max(\gamma_{dec}, \alpha_2)^{|\mathcal{U}_{i_\epsilon}|} \cdot \overline{\gamma}_{inc}^{|\mathcal{S}_{i_\epsilon}|}, \tag{3.48}$$

$$\leq \Delta_0^{init} \cdot \alpha_3^{|\mathcal{C}_{i_\epsilon}^U|+|\mathcal{F}_{i_\epsilon}|+|\mathcal{U}_{i_\epsilon}|} \cdot \overline{\gamma}_{inc}^{|\mathcal{S}_{i_\epsilon}|}, \tag{3.49}$$

from which (3.45) follows.

After every call of the criticality phase, we have either a safety, successful or unsuccessful step, giving us (3.46). Similarly, after every model-improving phase, the next iteration cannot call a subsequent model-improving phase, giving us (3.47). □

Assumption 3.16. The algorithm parameter $\epsilon_C \geq c_3\epsilon$ for some constant $c_3 > 0$.

Note that Assumption 3.16 can be easily satisfied by appropriate parameter choices in Algorithm 1.

Theorem 3.17. *Suppose Assumptions 2.1, 3.1, 3.5 and 3.16 hold. Then the number of iterations i_ϵ (i.e. the number of times a model m_k (2.5) is built) until $\|\nabla f(\mathbf{x}_{i_\epsilon+1})\| < \epsilon$ is at most*

$$\left\lfloor \frac{4f(\mathbf{x}_0)}{\eta_1 c_1} \left(1 + \frac{\log \overline{\gamma}_{inc}}{|\log \alpha_3|}\right) \max\left(\kappa_H c_4^{-2}\epsilon^{-2}, c_4^{-1}c_5^{-1}\epsilon^{-2}, c_4^{-1}(\Delta_0^{init})^{-1}\epsilon^{-1}\right) \right.$$
$$\left. + \frac{4}{|\log \alpha_3|} \max\left(0, \log\left(\Delta_0^{init} c_5^{-1}\epsilon^{-1}\right)\right) \right\rfloor \tag{3.50}$$

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

$$c_5 := \min\left(\frac{\omega_C}{\kappa_{eg} + 1/\mu}, \frac{\alpha_1 c_4}{\kappa_H}, \alpha_1\left(\kappa_{eg} + \frac{2\kappa_{ef}}{c_1(1-\eta_2)}\right)^{-1}\right). \tag{3.51}$$

Proof. From Assumption 3.16 and Lemma 3.8, we have $\epsilon_g = c_4\epsilon$. Similarly, from Lemma 3.9 we have $\rho_{min} = \min(\Delta_0^{init}, c_5\epsilon)$. Thus using Lemma 3.15, we can bound the total number of iterations by

$$|\mathcal{C}_{i_\epsilon}^M| + |\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{S}_{i_\epsilon}| + |\mathcal{M}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}| \tag{3.52}$$

$$\leq 4|\mathcal{S}_{i_\epsilon}| + 4\left(|\mathcal{C}_{i_\epsilon}^U| + |\mathcal{F}_{i_\epsilon}| + |\mathcal{U}_{i_\epsilon}|\right), \tag{3.53}$$

$$\leq 4|\mathcal{S}_{i_\epsilon}|\left(1 + \frac{\log \overline{\gamma}_{inc}}{|\log \alpha_3|}\right) + \frac{4}{|\log \alpha_3|}\log\left(\frac{\Delta_0^{init}}{\rho_{min}}\right), \tag{3.54}$$

and so (3.50) follows from this and Lemma 3.14. □

We can summarize our results as follows:

Corollary 3.18. *Suppose Assumptions 2.1, 3.1, 3.5 and 3.16 hold. Then for $\epsilon \in (0, 1]$, the number of iterations i_ϵ (i.e. the number of times a model m_k (2.5) 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 $\mathcal{O}(\kappa_H\kappa_d^2 n\epsilon^{-2})$, where $\kappa_d := \max(\kappa_{ef}, \kappa_{eg}) = \mathcal{O}(nL_J^2)$.*

Proof. From Theorem 3.17, we have $c_4^{-1} = \mathcal{O}(\kappa_{eg})$ and so

$$c_5^{-1} = \mathcal{O}(\max(\kappa_{eg}, \kappa_H c_4^{-1}, \kappa_{ef} + \kappa_{eg})) = \mathcal{O}(\kappa_H\kappa_d). \tag{3.55}$$

To leading order, the number of iterations is

$$\mathcal{O}(\max(\kappa_H c_4^{-2}, c_4^{-1}c_5^{-1})\epsilon^{-2}) = \mathcal{O}(\kappa_H\kappa_d^2\epsilon^{-2}), \tag{3.56}$$

as required. In every type of iteration, we change at most $n+1$ points, and so require no more than $n+1$ evaluations. The result $\kappa_d = \mathcal{O}(nL_J^2)$ follows from Lemma 3.4. □

Remark 3.19. Theorem 3.17 gives us a possible termination criterion for Algorithm 1 — we loop until k exceeds the value (3.50) 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, (3.50) is a worst-case bound and so unduly pessimistic.

Remark 3.20. In [10], the authors propose a different criterion to test whether the criticality phase should be entered: $\|\mathbf{g}_k^{init}\| \leq \Delta_k/\mu$ rather than $\|\mathbf{g}_k^{init}\| \leq \epsilon_C$ as found here and in [7]. We are able to use our criterion because of Assumption 3.16. 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 3.17. In practice, Assumption 3.16 is reasonable, as we would not expect a user to prescribe a criticality tolerance much smaller than their desired solution tolerance.

The standard complexity bound for first-order methods is $\mathcal{O}(\kappa_H \kappa_d^2 \epsilon^{-2})$ iterations and $\mathcal{O}(\kappa_H \kappa_d^2 n \epsilon^{-2})$ evaluations [10], where $\kappa_d = \mathcal{O}(\sqrt{n})$ and $\kappa_H = 1$. Corollary 3.18 gives us the same count of iterations and evaluations, but the worse bounds $\kappa_d = \mathcal{O}(n)$ and $\kappa_H = \mathcal{O}(\kappa_d)$, coming from the least-squares structure (Lemma 3.4).

However, our model (2.5) is better than a simple linear model for f, as it captures some of the curvature information in the objective via the term $J_k^T J_k$. This means that DFO-GN produces models which are between fully linear and fully quadratic [7, Definition 10.4], which is the requirement for convergence of second-order methods. It therefore makes sense to also compare the complexity of DFO-GN with the complexity of second-order methods.

Unsurprisingly, the standard bound for second-order methods is worse in general, than for first-order methods, namely, $\mathcal{O}(\max(\kappa_H \kappa_d^2, \kappa_d^3)\epsilon^{-3})$ iterations and $\mathcal{O}(\max(\kappa_H \kappa_d^2, \kappa_d^3) n^2 \epsilon^{-3})$ evaluations [14], where $\kappa_d = \mathcal{O}(n)$, to achieve second-order criticality for the given objective. Note that here $\kappa_d := \max(\kappa_{ef}, \kappa_{eg}, \kappa_{eh})$ for fully quadratic models. If $\|\nabla^2 f\|$ is uniformly bounded, then we would expect $\kappa_H = \mathcal{O}(\kappa_{eh}) = \mathcal{O}(\kappa_d)$.

Thus DFO-GN has the iteration and evaluation complexity of a first-order method, but the problem constants (i.e. dependency on n) of a second-order method. That is, assuming $\kappa_H = \mathcal{O}(\kappa_d)$ (as suggested by Lemma 3.4), DFO-GN requires $\mathcal{O}(n^3\epsilon^{-2})$ iterations and $\mathcal{O}(n^4\epsilon^{-2})$ evaluations, compared to $\mathcal{O}(n\epsilon^{-2})$ iterations and $\mathcal{O}(n^2\epsilon^{-2})$ evaluations for a first-order method, and $\mathcal{O}(n^3\epsilon^{-3})$ iterations and $\mathcal{O}(n^5\epsilon^{-3})$ evaluations for a second-order method.

Remark 3.21. In Lemma 3.4, we used the result $C = \mathcal{O}(\Lambda)$ whenever Y_k is Λ-poised, and wrote κ_{eg} in terms of C; see Appendix A for details on the provenance of C with respect to the interpolation system (2.3). Our approach here matches the presentation of the first- and second-order complexity bounds from [10, 14]. However, [7, Theorem 3.14] shows that C may also depend on n. Including this dependence, we have $C = \mathcal{O}(\sqrt{n}\,\Lambda)$ for DFO-GN and general first-order methods, and $C = \mathcal{O}(n^2\Lambda)$ for general second-order methods (where C is now adapted for quadratic interpolation). This would yield the alternative bounds $\kappa_d = \mathcal{O}(n)$ for first-order methods, $\mathcal{O}(n^2)$ for DFO-GN and $\mathcal{O}(n^3)$ for second-order methods[7]. Either way, we conclude that the complexity of DFO-GN lies between first- and second-order methods.

3.3.1 Discussion of Assumption 3.5

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 3.4. This means that Assumption 3.5, which typically necessary for first-order convergence (e.g. [7, 10]), is not required for Theorem 3.12 and our complexity analysis. To remove the assumption, we need to change Algorithm 1 in two places:

1. Replace the test for entering the criticality phase with

$$\min\left(\|\mathbf{g}_k^{init}\|, \frac{\|\mathbf{g}_k^{init}\|}{\max(\|H_k^{init}\|, 1)}\right) \leq \epsilon_C; \qquad \text{and} \tag{3.57}$$

2. Require the criticality phase to output m_k fully linear and Δ_k satisfying

$$\Delta_k \leq \mu \min\left(\|\mathbf{g}_k\|, \frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)}\right). \tag{3.58}$$

[7]For second-order methods, the fully quadratic bound is $\kappa_d = \mathcal{O}(nC)$.

With these changes, the criticality phase still terminates, but instead of (B.1) we have

$$\min\left(\Delta_k^{init}, \frac{\omega_C \epsilon}{\kappa_{eg} + 1/\mu}, \frac{\omega_C \epsilon}{\kappa_{eg} + \widetilde{\kappa}_H/\mu}\right) \leq \Delta_k \leq \Delta_k^{init}. \tag{3.59}$$

We can also augment Lemma 3.8 with the following, which can be used to arrive at a new value for ρ_{min}.

Lemma 3.22. *In all iterations,* $\|\mathbf{g}_k\|/\max(\|H_k\|, 1) \geq \min(\epsilon_C, \Delta_k/\mu)$. *If* $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$ *then*

$$\frac{\|\mathbf{g}_k\|}{\max(\|H_k\|, 1)} \geq \epsilon_H := \min\left(\epsilon_C, \frac{\epsilon}{(1 + \kappa_{eg}\mu)\widetilde{\kappa}_H}\right) > 0. \tag{3.60}$$

Ultimately, we arrive at complexity bounds which match Corollary 3.18, but replacing κ_H with $\widetilde{\kappa}_H$. However, Assumption 3.5 is still necessary for Theorem 3.13 to hold.

4 Implementation

In this section, we describe the key differences between Algorithm 1 and its software implementation DFO-GN. These differences largely come from Powell's implementation of BOBYQA [27] and are also features of DFBOLS, the implementation of the algorithm from Zhang et al. [34]. We also obtain a unified approach for analysing and improving the geometry of the interpolation set due to our particular choice of local Gauss-Newton-like models.

4.1 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 optimization methods. To this end, DFO-GN, like BOBYQA and DFBOLS, does not implement a criticality phase; but the safety step is implemented to encourage convergence.

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 maximizing the absolute value of each in $B(\mathbf{x}_k, \Delta_k)$. To modify Y_k to make it Λ-poised, we can repeat the following procedure [7, Algorithm 6.3]:

1. Select the point $\mathbf{y}_t \in Y_k$ ($\mathbf{y}_t \neq \mathbf{x}_k$) for which $\max_{\mathbf{y} \in B(\mathbf{x}_k, \Delta_k)} |\Lambda_t(\mathbf{y})|$ is maximized (c.f. (2.10));
2. Replace $\mathbf{y}_t$ in Y_k with $\mathbf{y}^+$, where

$$\mathbf{y}^+ = \operatorname*{arg\,max}_{\mathbf{y} \in B(\mathbf{x}_k, \Delta_k)} |\Lambda_t(\mathbf{y})|, \tag{4.1}$$

until Y_k is Λ-poised in $B(\mathbf{x}_k, \Delta_k)$. This procedure terminates after at most N iterations, where N depends only on Λ and n [7, Theorem 6.3], and in particular does not depend on $\mathbf{x}_k$, Y_k or Δ_k.

In DFO-GN, we follow BOBYQA and replace these geometry-checking and improvement algorithms (which are called in the safety and model-improvement phases of Algorithm 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 $\mathbf{y}^+$ satisfying (4.1). That is, we effectively perform one iteration of the full geometry-improving procedure.

4.2 Model Updating

In Algorithm 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 reasons for choosing a particular point to remove from the interpolation set:

Furthest Point: It is the furthest away from $\mathbf{x}_k$ (or $\mathbf{x}_{k+1}$);

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

Stable Update: Replacing it with $\mathbf{x}_k + \mathbf{s}_k$ would induce the most stable update to the interpolation system (2.4). As introduced by Powell [25] for quadratic models, moving $\mathbf{y}_t$ to $\mathbf{y}^+$ induces a low-rank update of the matrix $W \to W_{new}$ in the interpolation system, here (2.4). From the Sherman-Morrison-Woodbury formula, this induces a low-rank update of $H = W^{-1}$, which has the form

$$H_{new} \leftarrow H + \frac{1}{\sigma_t}\left[A_t B_t^\top\right], \tag{4.2}$$

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 (4.2) is 'stable'. In [25], it is shown that for underdetermined quadratic interpolation, $\sigma_t \geq \Lambda_t(\mathbf{y}^+)^2$.

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

$$t = \underset{j=0,\ldots,n}{\arg\max} \left\{ |\sigma_j| \max\left(\frac{\|\mathbf{y}_j - \mathbf{x}_k\|^4}{\Delta_k^4}, 1 \right) \right\}. \tag{4.3}$$

Alternatively, Scheinberg and Toint [28] combine the 'furthest point' and 'optimal Λ-poisedness' measures:

$$t = \underset{j=0,\ldots,n}{\arg\max} \left\{ |\Lambda_j(\mathbf{y}^+)| \, \|\mathbf{y}_j - \mathbf{x}_k\|^2 \right\}. \tag{4.4}$$

In DFO-GN, we use the BOBYQA criterion (4.3). However, as we now show, in DFO-GN, the two measures 'optimal Λ-poisedness' and 'stable update' coincide, meaning our framework allows a unification of the perspectives from [27] and [28], rather than having the indirect relationship via the bound $\sigma_t \geq \Lambda_t(\mathbf{y}^+)^2$.

To this end, define W as the matrix in (2.4), and let $H := W^{-1}$. The Lagrange polynomials for Y_k can then be found by applying the interpolation conditions (2.9). That is, we have

$$\Lambda_t(\mathbf{y}) = 1 + \mathbf{g}_t^\top (\mathbf{y} - \mathbf{y}_t), \tag{4.5}$$

where $\mathbf{g}_t$ solves

$$W\mathbf{g}_t = \begin{bmatrix} \Lambda_t(\mathbf{y}_1) - \Lambda_t(\mathbf{x}_k) \\ \vdots \\ \Lambda_t(\mathbf{y}_n) - \Lambda_t(\mathbf{x}_k) \end{bmatrix} = \begin{cases} \mathbf{e}_t, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ -\mathbf{e}, & \text{if } \mathbf{y}_t = \mathbf{x}_k, \end{cases} \tag{4.6}$$

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

$$\Lambda_t(\mathbf{y}^+) = \begin{cases} 1 + (H\mathbf{e}_t)^\top(\mathbf{y}^+ - \mathbf{y}_t), & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ 1 - (H\mathbf{e})^\top(\mathbf{y}^+ - \mathbf{x}_k), & \text{if } \mathbf{y}_t = \mathbf{x}_k. \end{cases} \tag{4.7}$$

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

$$W_{new} = W + \begin{cases} \mathbf{e}_t(\mathbf{y}^+ - \mathbf{y}_t)^\top, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ \mathbf{e}(\mathbf{x}_k - \mathbf{y}^+)^\top, & \text{if } \mathbf{y}_t = \mathbf{x}_k. \end{cases} \tag{4.8}$$

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

$$H_{new} = H - \frac{1}{\sigma_t} \begin{cases} H\mathbf{e}_t(\mathbf{y}^+ - \mathbf{y}_t)^\top H, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ H\mathbf{e}(\mathbf{x}_k - \mathbf{y}^+)^\top H, & \text{if } \mathbf{y}_t = \mathbf{x}_k. \end{cases} \tag{4.9}$$

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

$$\sigma_t = \begin{cases} 1 + (\mathbf{y}^+ - \mathbf{y}_t)^\top H\mathbf{e}_t, & \text{if } \mathbf{y}_t \neq \mathbf{x}_k, \\ 1 + (\mathbf{x}_k - \mathbf{y}^+)^\top H\mathbf{e}, & \text{if } \mathbf{y}_t = \mathbf{x}_k, \end{cases} \tag{4.10}$$

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

4.3 Termination Criteria

The specification in Algorithm 1 does not include any termination criteria. In the implementation of DFO-GN, we use the same termination criteria as DFBOLS [34], namely terminating whenever any of the following are satisfied:

- Small objective value: since $f \geq 0$ for least-squares problems, we terminate when

  $$f(\mathbf{x}_k) \leq \max\{10^{-12}, 10^{-20} f(\mathbf{x}_0)\}. \tag{4.11}$$

  For nonzero residual problems (i.e. where $f(\mathbf{x}^*) > 0$ at the true minimum $\mathbf{x}^*$), it is unlikely that termination will occur by this criterion;

- Small trust region: ρ_k, which converges to zero as $k \to \infty$ from Lemma 3.11, falls below a user-specified threshold; and

- Computational budget: a (user-specified) maximum number of evaluations of $\mathbf{r}$ is reached.

4.4 Other Implementation Differences

Addition of bound constraints This is allowed in the implementation of DFO-GN as it is important to practical applications. That is, we solve (2.1) subject to $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$ for given bounds $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$. This requires no change to the logic as specified in Algorithm 1, but does require the addition of the same bound constraints in the algorithms for the trust region subproblem (2.6) and calculating geometry-improving steps (4.1). For the trust-region subproblem, 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). Calculating geometry-improving steps (4.1) is easier, since the Lagrange polynomials are linear rather than quadratic. That is, we need to maximize a linear objective subject to Euclidean ball and bound constraints In this case, we use our own routine, given in Appendix C, which handles the bound constraints via an active set method.

Internal representation of interpolation points In the interpolation system (2.4), we are required to calculate the vectors $\mathbf{y}_t - \mathbf{x}_k$. As the algorithm progresses, we expect $\|\mathbf{y}_t - \mathbf{x}_k\| = \mathcal{O}(\Delta_k) \to 0$ as $k \to \infty$, and hence the evaluation $\mathbf{y}_t - \mathbf{x}_k$ to be sensitive to rounding errors, especially if $\|\mathbf{x}_k\|$ is large. To reduce the impact of this, internally the points are stored with respect to a 'base point' $\mathbf{x}_b$, which is periodically updated to remain 'close' to the points in Y_k. That is, we actually maintain the set $Y_k = \mathbf{x}_b + \{(\mathbf{y}_0 - \mathbf{x}_b), \ldots, (\mathbf{y}_n - \mathbf{x}_b)\}$. This means that in (2.4) we are computing $\|(\mathbf{y}_t - \mathbf{x}_b) - (\mathbf{x}_k - \mathbf{x}_b)\|$, where $\|\mathbf{y}_t - \mathbf{x}_b\|$ and $\|\mathbf{x}_k - \mathbf{x}_b\|$ are much smaller than $\|\mathbf{y}_t\|$ and $\|\mathbf{x}_k\|$, so the calculation is less sensitive to cancellation.

Once we have determined J_k by solving (2.4), the model $\mathbf{m}_k$ (2.1) is internally represented as

$$\mathbf{r}(\mathbf{y}) \approx \widetilde{\mathbf{m}}_k(\mathbf{y} - \mathbf{x}_b) := \mathbf{c}_k + J_k(\mathbf{y} - \mathbf{x}_b), \tag{4.12}$$

where $\mathbf{c}_k = \mathbf{r}(\mathbf{x}_k) - J_k(\mathbf{x}_k - \mathbf{x}_b)$, to ensure that $\widetilde{\mathbf{m}}_k(\mathbf{x}_k - \mathbf{x}_b) = \mathbf{r}(\mathbf{x}_k)$. Note that we take the argument of $\widetilde{\mathbf{m}}_k$ to be $\mathbf{y} - \mathbf{x}_b$, rather than $\mathbf{y} - \mathbf{x}_k$ as in (2.1).

When we update $\mathbf{x}_b$, say to $\mathbf{x}_b + \Delta\mathbf{b}$, we rewrite $\widetilde{\mathbf{m}}_k$ as

$$\widetilde{\mathbf{m}}_k(\mathbf{y} - \mathbf{x}_b) = [\mathbf{c}_k + J_k\Delta\mathbf{b}] + J_k(\mathbf{y} - \mathbf{x}_b - \Delta\mathbf{b}), \tag{4.13}$$

and so we update $\mathbf{x}_b \leftarrow \mathbf{x}_b + \Delta\mathbf{b}$ and $\mathbf{c}_k \leftarrow \mathbf{c}_k + J_k\Delta\mathbf{b}$. Lastly, the interpolation points have their representation changed to $Y_k \leftarrow (\mathbf{x}_b + \Delta\mathbf{b}) + \{(\mathbf{y}_0 - \mathbf{x}_b - \Delta\mathbf{b}), \ldots, (\mathbf{y}_n - \mathbf{x}_b - \Delta\mathbf{b})\}$.

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

- 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 22) only occurs when $r_k < 0$;
- Since we update the model on every iteration, we only reduce ρ_k after 3 consecutive unsuccessful iterations; 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);
- 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}:

$$\rho_{k+1} = \begin{cases} \alpha_1\rho_k, & \text{if } \rho_k > 250\rho_{end}, \\ \sqrt{\rho_k\rho_{end}}, & \text{if } 16\rho_{end} < \rho_k \leq 250\rho_{end}, \\ \rho_{end}, & \text{if } \rho_k \leq 16\rho_{end}. \end{cases} \tag{4.14}$$

- In some calls of the safety phase, we only reduce ρ_k and Δ_k, without improving the geometry of Y_k.

4.5 Comparison to DFBOLS

As has been discussed at length, there are many similarities between DFO-GN and DFBOLS from Zhang et al. [34]. Although the algorithm described in [34] allows Gauss-Newton type models in principle, in practice DFO-GN is simpler in several respects:

- The use of linear models for each residual (2.1) means we require only $n + 1$ interpolation points. In DFBOLS, quadratic models for each r_i are used, which requires between $n + 2$ and $(n + 1)(n + 2)/2$ points, where the remaining degrees of freedom are taken up by minimizing the change in the model Hessian. This results in both a more complicated interpolation problem compared to our system (2.4), and a larger 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 models, there is no ambiguity in how to construct the full model m_k (2.5). In DFBOLS, simply taking a sum of squares of 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, depending on the sizes of $\|\mathbf{g}_k\|$ and $f(\mathbf{x}_k)$. This requires the introduction of three new algorithm parameters, each of which may require calibration.
- DFO-GN's method for choosing a point to replace when doing model updating, as discussed in Section 4.2, 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 (4.3) with $\sigma_t \geq \Lambda_t(\mathbf{y}^+)^2$. As discussed in [27], 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-GN does not need or have this routine.

The first of these points also applies to Wild's POUNDERS [30], which builds quadratic models for each residual, and constructs a model for the full objective which is equivalent to a full Newton model (i.e. taking all available second-order information).

It is also important to note that neither [34] nor [30] test the use of DFO-GN type linear models for each residual in practice.

5 Numerical Results

Now we compare the practical performance of DFO-GN to two versions of DFBOLS [34], and show that DFO-GN has comparable budget performance and significantly faster runtime. The two versions of DFBOLS are the original Fortran implementation from [34], and the other is our own Python implementation (which we will call 'Py-DFBOLS'), designed to be as similar as possible to the implementation of DFO-GN. Both DFO-GN and Py-DFBOLS use Python 3.5.2[8]. We also compare with the general-objective solver BOBYQA [27] and POUNDERS [30], another least-squares DFO code which uses quadratic interpolation models for each residual.

The parameter values used for DFO-GN are: $\Delta_{max} = 10^{10}$, $\gamma_{dec} = 0.5$, $\gamma_{inc} = 2$, $\overline{\gamma}_{inc} = 4$, $\eta_1 = 0.1$, $\eta_2 = 0.7$, $\alpha_1 = 0.1$, $\alpha_2 = 0.5$, $\omega_S = 0.1$ and $\gamma_S = 0.5$. For all solvers, we use an initial trust region radius of $\rho_0 = \Delta_0 = 0.1 \max(\|\mathbf{x}\|_\infty, 1)$ and final trust region radius $\rho_{end} = 10^{-10}$ where possible[9], to avoid this being the termination condition as often as possible.

We tested BOBYQA and (Py-)DFBOLS with $n+2$, $2n+1$ and $(n+1)(n+2)/2$ interpolation points. In the results which follow, for simplicity 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.

5.1 Test Problems and Methodology

We tested the solvers on the test suite from Moré and Wild [18], a collection of 53 unconstrained nonlinear least-squares problems with dimension $2 \leq n \leq 12$ and $2 \leq m \leq 65$. For each problem, we optionally allowed evaluations of the residuals r_i to have stochastic noise. Specifically, we allowed the following noise models:

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

To compare solvers, we use data and performance profiles [18]. 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':

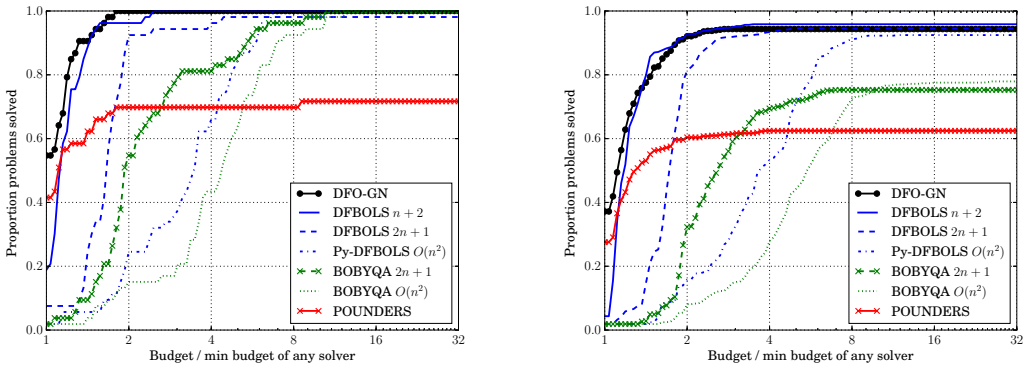
$$N_p(\mathcal{S}; \tau) := \#\text{ objective evals required to get } f(\mathbf{x}_k) \leq \mathbb{E}[f^* + \tau(f(\mathbf{x}_0) - f^*)], \tag{5.1}$$

where f^* is an estimate of the true minimum[10] $f(\mathbf{x}^*)$. A full list of the values used is provided in Appendix D. We define $N_p(\mathcal{S}; \tau) = \infty$ if this was not achieved in the maximum computational budget allowed.

[8] With NumPy 1.12.1. Linear solves use LAPACK's LU decomposition routines, wrapped by SciPy 0.19.0.

[9] POUNDERS does not have this as a user input. Instead we set all gradient tolerances to zero.

[10] Note that in [18], and subsequent other papers such as [34], the value f^* is usually taken to be the smallest objective value achieved by any of the solvers under consideration within a fixed budget. The main motivation in [18] for this choice is for when f is expensive, and so we have small computational budgets and it is possible that no solver converges. In our setting, this is not the case, so we use our (stronger) choice of f^*, which comes from [17] or the results of running these and other (derivative-based) solvers.



(a) Smooth objective *(b) Mult. Gaussian noise $\sigma = 10^{-2}$*

Figure 1. Performance profile comparison of DFO-GN 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 Figure 1b, results shown are an average of 10 runs for each solver.

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

$$d_{\mathcal{S},\tau}(\alpha) := \frac{|\{p \in \mathcal{P} : N_p(\mathcal{S};\tau) \leq \alpha(n_p+1)\}|}{|\mathcal{P}|}, \qquad \text{for } \alpha \in [0, N_g], \tag{5.2}$$

where N_g is the maximum computational budget, measured in simplex gradients (i.e. $N_g(n_p+1)$ objective evaluations are allowed for problem p).

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

$$\pi_{\mathcal{S},\tau}(\alpha) := \frac{|\{p \in \mathcal{P} : N_p(\mathcal{S};\tau) \leq \alpha N_p^*(\tau)\}|}{|\mathcal{P}|}, \qquad \text{for } \alpha \geq 1, \tag{5.3}$$

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

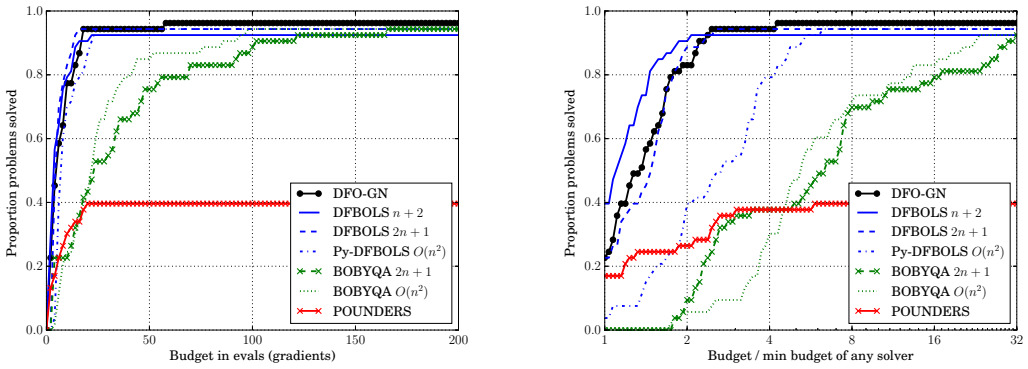
For test runs where we added stochastic noise, we took average data and performance profiles over multiple runs of each solver; that is, for each α we take an average of $d_{\mathcal{S},\tau}(\alpha)$ and $\pi_{\mathcal{S},\tau}(\alpha)$. When plotting performance profiles, we took $N_p^*(\tau)$ to be the minimum budget required by any solver in any run.

5.2 Test Results

For our testing, we used a budget of $N_g = 200$ gradients (i.e. $200(n+1)$ objective evaluations) for each problem, noise level $\sigma = 10^{-2}$, and took 10 runs of each solver[11]. Most results use an accuracy level of $\tau = 10^{-5}$ in (5.1).

Firstly, Figure 1 shows two performance profiles under the low accuracy requirement $\tau = 10^{-1}$. Here we see an important benefit of DFO-GN compared to BOBYQA, DFBOLS and POUNDERS — the smaller interpolation set means that it can begin the main iteration and make progress sooner. This is reflected in Figure 1, where DFO-GN is the fastest solver more frequently than any other, both with smooth and noisy objective evaluations. We also note that POUNDERS is the faster solver more frequently than DFBOLS at this accuracy level. However, after the full budget of 200 gradients, POUNDERS is unable to solve a high proportion of problems to this accuracy — this holds both here and for the remainder of the results. We believe this is because it uses a built-in termination condition for sufficiently small trust region radius, which is set

[11] Scheduled using [29].



(a) Data Profile *(b) Performance Profile*

Figure 2. Comparison of DFO-GN with BOBYQA, DFBOLS and POUNDERS for smooth objectives, 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.

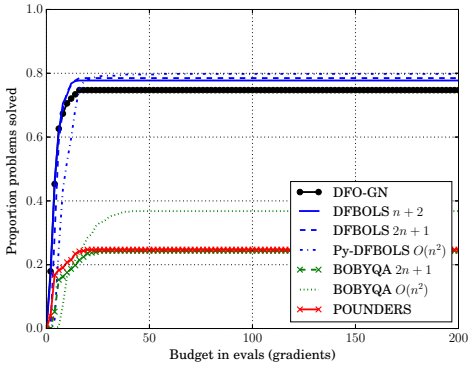
too high to achieve (5.1) for many problems. In line with the results from [34], BOBYQA does not perform as well as DFBOLS or DFO-GN, as it does not exploit the least-squares problem structure.

Next, Figure 2 shows results for accuracy $\tau = 10^{-5}$ and smooth objective evaluations. It is important to note is that our simplification from quadratic to linear residual models has not led to a loss of performance at 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 profiles, we can still see the substantially higher startup cost of using $(n+1)(n+2)/2$ interpolation points.

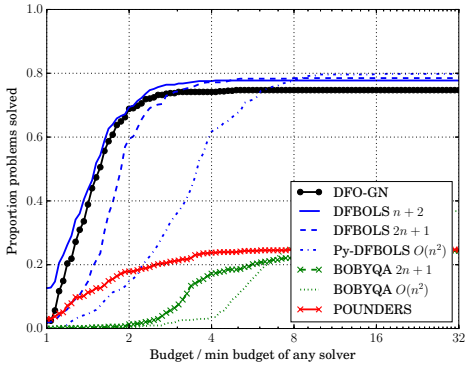
Similarly, Figure 3 shows the same plots but for noisy problems (multiplicative Gaussian, additive Gaussian and additive χ^2 respectively). Here, DFO-GN suffers a small performance penalty (of approximately 5-10%) compared to DFBOLS, particularly when using $2n+1$ and $(n+1)(n+2)/2$ interpolation points, 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 (approximately 10% vs. 5%). Note that additive noise makes all our test problems nonzero residual (i.e. $f(\mathbf{x}^*) > 0$ for the true minimum $\mathbf{x}^*$). Thus the worse performance of DFO-GN compared to DFBOLS for additive noise is similar to the derivative-based case, where for nonzero residual problems the Gauss-Newton method has a lower asymptotic convergence rate than Newton's method [19].

Also of note is that although BOBYQA suffers a substantial performance penalty when moving from smooth to noisy problems, this penalty (compared to DFO-GN and DFBOLS) is much less for additive χ^2 noise. This is likely because this noise model makes each residual function nonsmooth by taking square roots, but the change to the full objective is relatively benign — simply adding χ^2 random variables.

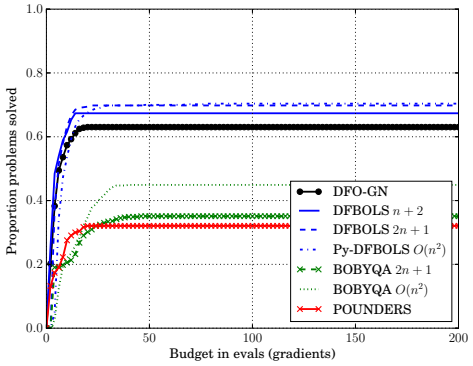
Nonzero Residual Problems We saw above that DFO-GN suffered a higher — but still small — loss of performance, compared to DFBOLS, for problems with additive noise, where all problems become nonzero residual. We investigate this further by extracting the performance of the nonzero residual problems only from the test set results we already presented; Figure 4 shows the resulting performance profiles for accuracy $\tau = 10^{-5}$, for smooth objectives and multiplicative Gaussian noise ($\sigma = 10^{-2}$). We notice that in the smooth case, DFBOLS with $2n+1$ points is now the fastest solver on 30% of problems versus 20% for DFO-GN, compared to looking at all problems (seen in Figure 2b); but DFO-GN is comparably robust and we do not see the same loss of performance as in the additive noise case. In the case of multiplicative Gaussian noise, DFO-GN, and in fact (Py-)DFBOLS, see a loss of performance compared to looking at all problems; also, the difference between DFO-GN and DFBOLS with increasing numbers



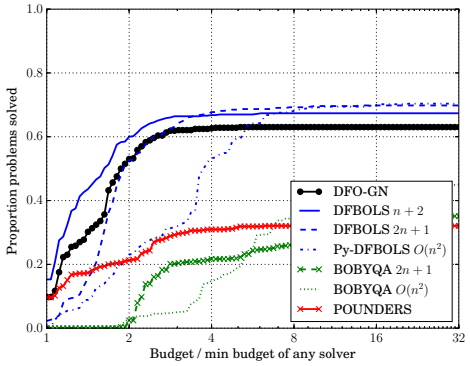
(a) Mult. Gaussian, data profile



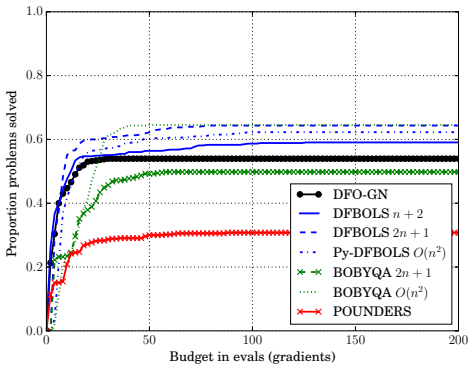
(b) Mult. Gaussian, performance profile



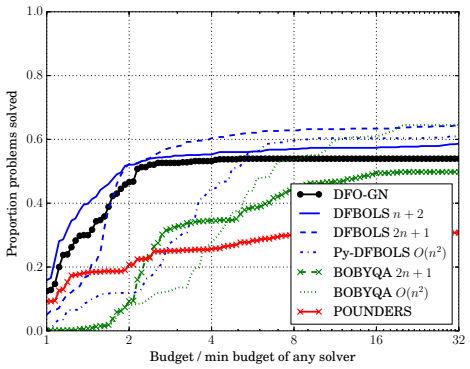
(c) Add. Gaussian, data profile



(d) Add. Gaussian, performance profile

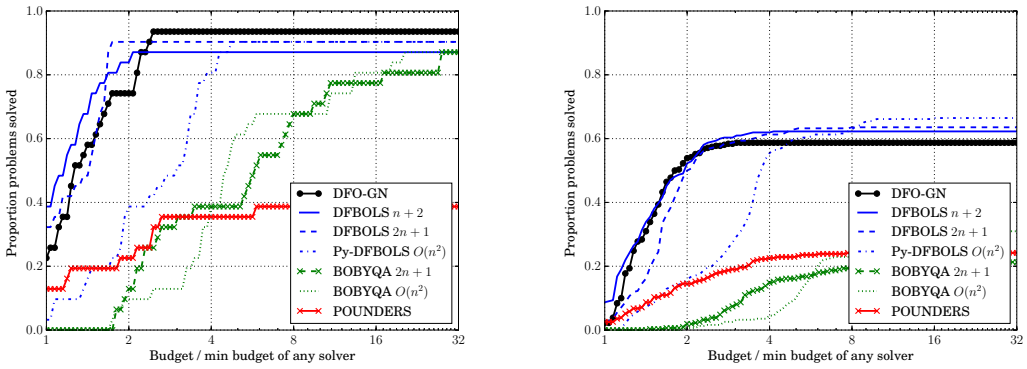


(e) Add. χ^2, data profile



(f) Add. χ^2, performance profile

Figure 3. Comparison of DFO-GN with BOBYQA, DFBOLS and POUNDERS for objectives with multiplicative Gaussian, additive Gaussian and additive χ^2 noise with $\sigma = 10^{-2}$, to accuracy $\tau = 10^{-5}$ (average of 10 runs for each solver). 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.



(a) Smooth objective *(b) Mult. Gaussian noise $\sigma = 10^{-2}$*

Figure 4. Performance profile comparison of DFO-GN 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 Figure 4b, results shown are an average of 10 runs for each solver.

of interpolation points is slightly more clear for nonzero residual problems (8% vs. 5% for all problems).

Conclusions to evaluation comparisons The numerical results in this section show that DFO-GN performs comparably to DFBOLS in terms of evaluation counts, and outperforms BOBYQA and POUNDERS, in both smooth and noisy settings, and for low and high accuracy. DFO-GN exhibits a slight performance loss compared to DFBOLS for additive noisy problems and for noisy non-zero residual problems. We note that we also tested other noise models — such as multiplicative uniform noise and also biased variants of the Gaussian noise; all these performed either better (such as in the case of uniform noise) or essentially indistinguishable to the results already presented above. We also tried other noise variance levels, smaller than $\sigma = 10^{-2}$, for which the performance of both DFO-GN and DFBOLS solvers vary similarly/comparably. We may explain the similar performance of DFO-GN to DFBOLS, despite the higher order models used by the latter, 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.

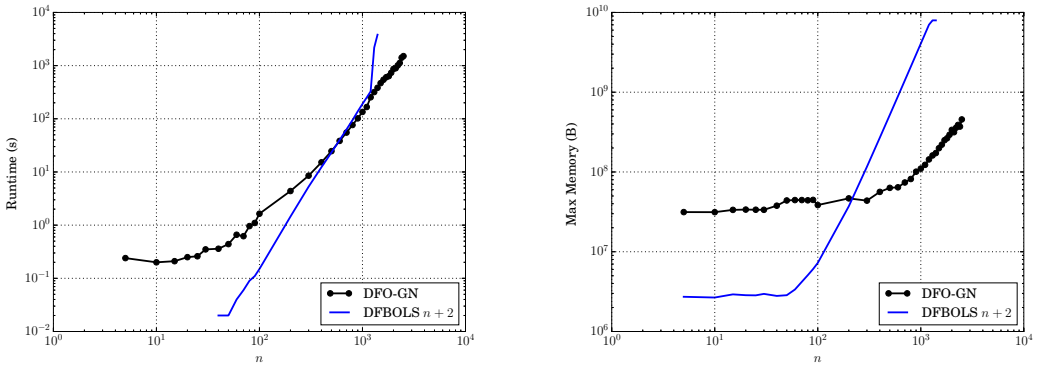
Though an accuracy level $\tau = 10^{-5}$ is common and considered reasonably high in DFO, to ensure that our results are robust, we also performed the same tests for higher accuracy levels $\tau \in \{10^{-7}, 10^{-9}, 10^{-11}\}$. The resulting profiles are given in Appendix F. For smooth problems, DFO-GN is still able to solve essentially the same proportion of problems as (Py-)DFBOLS. For noisy problems, the results are more mixed: on average, DFO-GN 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.

5.3 Runtime Comparison

The use of linear models in DFO-GN also leads to reduced linear algebra cost. The interpolation system for DFO-GN (2.4) is of size n. By comparison, for underdetermined quadratic interpolation, such as in (Py-)DFBOLS, the interpolation system has size $p+n+1$ when we have $p \geq n+2$ interpolation points. Hence the DFBOLS system is at least twice the size of the DFO-GN system, so we would expect the computational cost of solving this system to be at least 8 times larger than for DFO-GN. To verify this, in this section we compare the runtime of DFO-GN with

Solver	Smooth	Mult. Gaussian	Add. Gaussian	Add. χ^2
DFO-GN	51s [1x]	8s [1x]	8s [1x]	11s [1x]
Py-DFBOLS $n+2$	426s [8.4x]	67s [8.5x]	59s [7.6x]	94s [8.8x]
Py-DFBOLS $2n+1$	600s [11.8x]	192s [24.3x]	174s [22.4x]	219s [20.4x]
Py-DFBOLS $\mathcal{O}(n^2)$	2157s [42.3x]	3052s [386.5x]	2930s [377.4x]	3563s [331.1x]

Table 1. Runtimes for DFO-GN and Py-DFBOLS. Noisy results are an average of 10 runs with $\sigma = 10^{-2}$, and all runs used a budget of $200(n+1)$ objective evaluations. Values are raw (in seconds) and ratio compared to the DFO-GN runtime. For Py-DFBOLS, $\mathcal{O}(n^2) = (n+1)(n+2)/2$ interpolation points.



(a) Runtime *(b) Peak memory usage*

Figure 5. Comparison of runtime and peak memory usage of DFBOLS (original Fortran implementation with $n+2$ interpolation points) and DFO-GN for solving the discretized integral equation, as problem dimension n increases. The largest values tested were $n = 1400$ for DFBOLS and $n = 2500$ for DFO-GN.

Py-DFBOLS. The other solvers are implemented in lower-level languages (Fortran and C), and so a runtime comparison against DFO-GN does not provide a fair comparison.

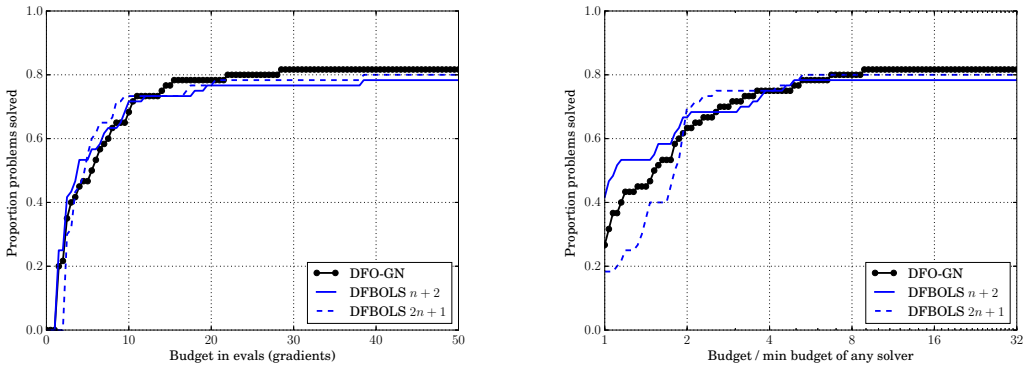
The wall time required by each solver to run the above testing (with a budget of $200(n+1)$ objective evaluations) on a Lenovo ThinkCentre M900 (with one 64-bit Intel i5 processor, 8GB of RAM), is shown in Table 1 for both smooth and noisy evaluations. We find that DFO-GN is 7-9x faster than Py-DFBOLS with $n+2$ points, 11-25x faster than Py-DFBOLS with $2n+1$ points, and 42-387x 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-GN and (Py-)DFBOLS described in Section 5.2.

We note that these runtimes include objective evaluations. However, all solvers used the same Python implementation of the objective functions and the same total budget, so the inclusion of objective evaluations in the runtime will not materially affect the results.

5.4 Scalability Features

We saw in Section 5.3 that DFO-GN runs faster 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-GN should scale to large problems better than DFBOLS — in this section we demonstrate this. We consider Problem 29 from Moré, Garbow & Hillstrom [17] (which is Problem 33 (INTEGREQ) in the CUTEst set in Table 3). This is a zero-residual least-squares problem with $m = n$ variable for solving a one-dimensional integral equation, using an n-point discretization of $(0, 1)$. Both DFO-GN and DFBOLS[12] solve this problem — terminating on small objective value $f(\mathbf{x}) \leq 10^{-12}$ — quickly, in no more than 20 iterations, regardless of n.

[12]We used $|Y_k| = n+2$ for DFBOLS, as it has the smallest memory usage and runtime of all possible values.



(a) Data Profile *(b) Performance Profile*

Figure 6. Comparison of DFO-GN with DFBOLS for smooth objectives from the set of medium-sized CUTEst problems, to accuracy $\tau = 10^{-5}$. For the DFBOLS runs, $n+2$, $2n+1$ are the number of interpolation points.

In Figure 5 we compare the runtime and peak memory usage of DFBOLS and DFO-GN as n increases. Note that we are comparing DFO-GN (implemented in Python) against DFBOLS (implemented in Fortran) rather than Py-DFBOLS (as used in Section 5.3), to put ourselves at a substantial disadvantage. We see that for small n, DFBOLS has significantly lower runtime and memory requirements than DFO-GN (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-GN as n is increased. In fact, for $n > 500$, DFO-GN actually runs faster than DFBOLS. 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-GN does not suffer from this issue, and can continue solving problems quickly for substantially larger n. For instance, DFO-GN solves the $n = 2500$ problem over 2.5 times faster than DFBOLS solves the much smaller $n = 1400$ problem.

Similarly to before, it is important to gain an understanding of whether this improved scalability comes at the cost of performance. To assess this, we consider a set of 60 medium-sized problems ($25 \leq n \leq 120$ and $25 \leq m \leq 400$, with $n \approx 100$ for most problems) from the CUTEst test set [11]. The full list of problems is given in Appendix E. For these problems, we compare DFO-GN with DFBOLS using a smaller budget of $50(n+1)$ evaluations, commensurate with the greater cost of objective evaluation. Given this small budget, we only test DFBOLS with $n+2$ and $2n+1$ interpolation points; using $(n+1)(n+2)/2$ interpolation points would mean in most cases the full budget is entirely used building the initial sample set.

In Figure 6, we show data and performance profiles for accuracy $\tau = 10^{-5}$. As before, we see that DFO-GN has very similar performance to DFBOLS, and although we have gained improved scalability, we have not lost in terms of performance on medium-sized test problems.

6 Concluding Remarks

It is well-known that, for nonlinear least-squares problems, using only linear 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 our derivative-free, model-based trust-region variant here, called DFO-GN.

In [34, 30], quadratic local models are constructed by interpolation for each residual, and these are aggregated to produce a quadratic model for the objective. By contrast, in DFO-GN we build linear models for each residual, which retains first-order convergence and worst-case complexity, and reduces both the computational cost of solving the interpolation problem (leading to a runtime reduction of at least a factor of 7) and the memory cost of storing the models (from $\mathcal{O}(mn^2)$

to $\mathcal{O}(mn)$). These savings result in a substantially faster runtime and improved scalability of DFO-GN compared to DFBOLS, the implementation from [34]. Furthermore, the simpler local models do not adversely affect the algorithm's performance numerically, in terms of evaluation counts: DFO-GN performs as well as DFBOLS and better than POUNDERS, the implementation from [30], on smooth test problems from the Moré & Wild and CUTEst collections. When the objective has noise, DFO-GN suffers a small performance penalty compared to DFBOLS (but not to POUNDERS), which is larger for additive than multiplicative noise as all problems become nonzero residual. Nonetheless, this, together with the substantial improvements in runtime and scalability, make DFO-GN an appealing choice for both zero and nonzero residuals, and in the presence of noise. We delegate to future work showing local quadratic rate of convergence for DFO-GN when applied to nondegenerate zero-residual problems, and generally improving the performance of DFO methods in the presence of noise.

References

[1] Y. Aoki, B. 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 J. Sci. Comput., 36 (2014), pp. B14–B44.

[2] 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 J. Uncertain. Quantif., 4 (2016), pp. 924–951.

[3] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, MPS-SIAM Series on Optimization, MPS/SIAM, Philadelphia, 2000.

[4] A. R. Conn, K. Scheinberg, and P. L. Toint, *Recent progress in unconstrained nonlinear optimization without derivatives*, Math. Program., 79 (1997), pp. 397–414.

[5] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Geometry of interpolation sets in derivative free optimization*, Math. Program., 111 (2007), pp. 141–172.

[6] ——, *Global Convergence of General Derivative-Free Trust-Region Algorithms to First- and Second-Order Critical Points*, SIAM J. Optim., 20 (2009), pp. 387–415.

[7] ——, *Introduction to Derivative-Free Optimization*, vol. 8 of MPS-SIAM Series on Optimization, MPS/SIAM, Philadelphia, 2009.

[8] A. R. Conn and P. L. Toint, *An Algorithm using Quadratic Interpolation for Unconstrained Derivative Free Optimization*, in Nonlinear Optim. Appl., G. Di Pillo and F. Gianessi, eds., Plenum Publishing, New York, 1996, pp. 27–47.

[9] A. L. Custódio, K. Scheinberg, and L. N. Vicente, *Methodologies and Software for Derivative-free Optimization*, in Adv. Trends Optim. with Eng. Appl., T. Terlaky, M. F. Anjos, and S. Ahmed, eds., MOS-SIAM Book Series on Optimization, SIAM, Philadelphia, 2017.

[10] R. Garmanjani, D. Júdice, and L. N. Vicente, *Trust-Region Methods Without Using Derivatives: Worst Case Complexity and the Nonsmooth Case*, SIAM J. Optim., 26 (2016), pp. 1987–2011.

[11] N. I. M. Gould, D. Orban, and P. L. Toint, *CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization*, Comput. Optim. Appl., 60 (2015), pp. 545–557.

[12] N. I. M. Gould, M. Porcelli, and P. L. Toint, *Updating the regularization parameter in the adaptive cubic regularization algorithm*, Comput. Optim. Appl., 53 (2012), pp. 1–22.

[13] G. N. Grapiglia, J. Yuan, and Y.-x. Yuan, *A derivative-free trust-region algorithm for composite nonsmooth optimization*, Comput. Appl. Math., 35 (2016), pp. 475–499.

[14] D. Júdice, *Trust-Region Methods without using Derivatives : Worst-Case Complexity and the Non-Smooth Case*, PhD thesis, University of Coimbra, 2015.

[15] T. G. Kolda, R. M. Lewis, and V. Torczon, *Optimization by Direct Search : New Perspectives on Some Classical and Modern Methods*, SIAM Rev., 45 (2003), pp. 385–482.

[16] L. Lukšan, *Hybrid Methods for Large Sparse Nonlinear Least Squares*, J. Optim. Theory Appl., 89 (1996), pp. 575–595.

[17] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *Testing Unconstrained Optimization Software*, ACM Trans. Math. Softw., 7 (1981), pp. 17–41.

[18] J. J. Moré and S. M. Wild, *Benchmarking Derivative-Free Optimization Algorithms*, SIAM J. Optim., 20 (2009), pp. 172–191.

[19] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, 2nd ed., 2006.

[20] R. Oeuvray and M. Bierlaire, *BOOSTERS: A Derivative-Free Algorithm Based on Radial Basis Functions*, Int. J. Model. Simul., 29 (2009), pp. 26–36.

[21] M. J. D. Powell, *A direct search optimization method that models the objective and constraint functions by linear interpolation*, in Adv. Optim. Numer. Anal., S. Gomez and J.-P. Hennart, eds., Kluwer Academic Publishers, Dordrecht, 1994, pp. 51–67.

[22] ——, *Direct search algorithms for optimization calculations*, Acta Numerica, 7 (1998), pp. 287–336.

[23] ——, *UOBYQA: Unconstrained optimization by quadratic approximation*, Math. Program., 92 (2002), pp. 555–582.

[24] ——, *On trust region methods for unconstrained minimization without derivatives*, Math. Program., 97 (2003), pp. 605–623.

[25] ——, *Least Frobenius norm updating of quadratic models that satisfy interpolation conditions*, Math. Program., 100 (2004), pp. 183–215.

[26] ——, *A view of algorithms for optimization without derivatives*, Tech. Rep. DAMTP 2007/NA03, University of Cambridge, 2007.

[27] ——, *The BOBYQA algorithm for bound constrained optimization without derivatives*, Tech. Rep. DAMTP 2009/NA06, University of Cambridge, 2009.

[28] K. Scheinberg and P. L. Toint, *Self-Correcting Geometry in Model-Based Algorithms for Derivative-Free Unconstrained Optimization*, SIAM J. Optim., 20 (2010), pp. 3512–3532.

[29] O. Tange, *GNU Parallel: the command-line power tool*, ;login USENIX Mag., 36 (2011), pp. 42–47.

[30] S. M. Wild, *POUNDERS in TAO: Solving Derivative-Free Nonlinear Least-Squares Problems with POUNDERS*, in Adv. Trends Optim. with Eng. Appl., SIAM, Philadelphia, PA, 2017, ch. 40, pp. 529–539.

[31] S. M. Wild, R. G. Regis, and C. A. Shoemaker, *ORBIT: Optimization by Radial Basis Function Interpolation in Trust-Regions*, SIAM J. Sci. Comput., 30 (2008), pp. 3197–3219.

[32] S. M. Wild and C. A. Shoemaker, *Global Convergence of Radial Basis Function Trust-Region Algorithms for Derivative-Free Optimization*, SIAM Rev., 55 (2013), pp. 349–371.

[33] D. Winfield, *Function minimization by interpolation in a data table*, IMA J. Appl. Math., 12 (1973), pp. 339–347.

[34] H. Zhang, A. R. Conn, and K. Scheinberg, *A Derivative-Free Algorithm for Least-Squares Minimization*, SIAM J. Optim., 20 (2010), pp. 3555–3576.

[35] Z. Zhang, *Software by Professor M. J. D. Powell*. `http://mat.uc.pt/~zhang/software.html`, 2017.

A Proof of Lemma 3.4

This proof is similar to [34, Lemma 4.3] and [7, Theorems 2.11 and 2.12]. Define $B := B(\mathbf{x}_k, \Delta_k)$ for convenience. We recall the standard bound [19, Appendix A]

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

From the interpolation conditions (2.3), we have

$$J_k(\mathbf{y}_t - \mathbf{x}_k) = \mathbf{r}(\mathbf{y}_t) - \mathbf{r}(\mathbf{x}_k), \qquad \text{for } t = 1, \ldots, n. \tag{A.2}$$

Using (A.1), we compute for any $t = 1, \ldots, n$,

$$\|[J_k - J(\mathbf{x}_k)](\mathbf{y}_t - \mathbf{x}_k)/\Delta_k\| = \Delta_k^{-1} \|\mathbf{r}(\mathbf{y}_t) - \mathbf{r}(\mathbf{x}_k) - J(\mathbf{x}_k)(\mathbf{y}_t - \mathbf{x}_k)\| \leq \frac{1}{2} L_J \Delta_k. \tag{A.3}$$

Let $\hat{W}_k$ be the interpolation matrix of the system (2.4) scaled by Δ_k^{-1}. Considering the matrix $[J_k - J(\mathbf{x}_k)]\hat{W}_k^\top$, with columns $[J_k - J(\mathbf{x}_k)](\mathbf{y}_t - \mathbf{x}_k)/\Delta_k$, we have

$$\|[J_k - J(\mathbf{x}_k)]\hat{W}_k^\top\|^2 \leq \|[J_k - J(\mathbf{x}_k)]\hat{W}_k^\top\|_F^2 = \sum_{t=1}^{n} \|[J_k - J(\mathbf{x}_k)](\mathbf{y}_t - \mathbf{x}_k)/\Delta_k\|^2, \tag{A.4}$$

and so using the identity $\|\hat{W}_k^{-1}\| = \|\hat{W}_k^{-\top}\|$, we get

$$\|J_k - J(\mathbf{x}_k)\| \leq \|[J_k - J(\mathbf{x}_k)]\hat{W}_k^\top\| \cdot \|\hat{W}_k^{-\top}\| \leq \frac{1}{2} L_J \sqrt{n} \|\hat{W}_k^{-1}\| \Delta_k. \tag{A.5}$$

Thus we conclude that for any $\mathbf{y} \in B$

$$\|J_k - J(\mathbf{y})\| \leq \|J_k - J(\mathbf{x}_k)\| + \|J(\mathbf{y}) - J(\mathbf{x}_k)\| \leq L_J \left(1 + \frac{1}{2}\sqrt{n}\|\hat{W}_k^{-1}\|\right) \Delta_k. \tag{A.6}$$

Since Y_k is Λ-poised in B, we have $\|\hat{W}_k^{-1}\| = \mathcal{O}(\Lambda)$ from [7, Theorem 3.14]. Thus (2.14) holds with $\kappa_{eg}^r := L_J\left(1 + \frac{1}{2}\sqrt{n}C\right)$, where $C = \mathcal{O}(\Lambda)$. Next, we prove (2.13) by computing

$$\begin{aligned}
\|\mathbf{m}_k(\mathbf{y} - \mathbf{x}_k) - \mathbf{r}(\mathbf{y})\| &= \|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x}_k) - J_k(\mathbf{y} - \mathbf{x}_k)\|, &\text{(A.7)}\\
&\leq \|\mathbf{r}(\mathbf{y}) - \mathbf{r}(\mathbf{x}_k) - J(\mathbf{x}_k)(\mathbf{y} - \mathbf{x}_k)\| + \|J(\mathbf{x}_k) - J_k\| \cdot \|\mathbf{y} - \mathbf{x}_k\|, &\text{(A.8)}\\
&\leq \left(\frac{L_J}{2} + \kappa_{eg}^r\right) \Delta_k^2, &\text{(A.9)}
\end{aligned}$$

where we use (2.14) and (A.1). Hence we have (2.13) with $\kappa_{ef}^r = \kappa_{eg}^r + L_J/2$, as required.

Since $\mathbf{m}_k$ is fully linear, we also get from (2.14) the bound

$$\|J_k\| \leq \|J(\mathbf{x}_k) - J_k\| + \|J(\mathbf{x}_k)\| \leq \kappa_{eg}^r \Delta_{max} + J_{max}, \tag{A.10}$$

so $\|J_k\|$ is uniformly bounded for all k. Since $H_k = J_k^\top J_k$, this means that $\|H_k\| = \|J_k\|^2$ is uniformly bounded for all k.

To prove full linearity of m_k, we first compute

$$\|\nabla m_k(\mathbf{y}-\mathbf{x}_k)-\nabla f(\mathbf{y})\| = \|\nabla f(\mathbf{y}) - J_k^\top \mathbf{r}(\mathbf{x}_k) - J_k^\top J_k(\mathbf{y}-\mathbf{x}_k)\|, \tag{A.11}$$

$$\leq \|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x}_k)\| + \|(J(\mathbf{x}_k)-J_k)^\top \mathbf{r}(\mathbf{x}_k)\| + \|J_k^\top J_k\| \cdot \|\mathbf{y}-\mathbf{x}_k\|, \tag{A.12}$$

$$\leq L_{\nabla f}\Delta_k + \kappa_{eg}^r r_{max}\Delta_k + \left(\kappa_{eg}^r \Delta_{max} + J_{max}\right)^2 \Delta_k, \tag{A.13}$$

recovering (2.12) with $\kappa_{eg} = L_{\nabla f} + \kappa_{eg}^r r_{max} + (\kappa_{eg}^r \Delta_{max} + J_{max})^2$, as required.

Lastly, to show (2.11), we recall the scalar version of (A.1)

$$|f(\mathbf{y}) - f(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)^\top(\mathbf{y}-\mathbf{x}_k)| \leq \frac{1}{2}L_{\nabla f}\|\mathbf{y}-\mathbf{x}_k\|^2. \tag{A.14}$$

We use this and (2.12) to compute

$$|m_k(\mathbf{y}-\mathbf{x}_k) - f(\mathbf{y})| = \left| f(\mathbf{y}) - f(\mathbf{x}_k) - \mathbf{g}_k^\top(\mathbf{y}-\mathbf{x}_k) - \frac{1}{2}(\mathbf{y}-\mathbf{x}_k)^\top H_k(\mathbf{y}-\mathbf{x}_k)\right|, \tag{A.15}$$

$$\leq \left|f(\mathbf{y}) - f(\mathbf{x}_k) - \nabla f(\mathbf{x}_k)^\top(\mathbf{y}-\mathbf{x}_k)\right| + \left\|\nabla f(\mathbf{x}_k) - \mathbf{g}_k - \frac{1}{2}H_k(\mathbf{y}-\mathbf{x}_k)\right\| \cdot \|\mathbf{y}-\mathbf{x}_k\|, \tag{A.16}$$

$$\leq \frac{1}{2}L_{\nabla f}\Delta_k^2 + \left[\|\nabla f(\mathbf{x}_k) - \nabla m_k(\mathbf{y}-\mathbf{x}_k)\| + \frac{1}{2}\|H_k\| \cdot \|\mathbf{y}-\mathbf{x}_k\|\right] \cdot \Delta_k, \tag{A.17}$$

$$\leq \frac{1}{2}L_{\nabla f}\Delta_k^2 + \left[\kappa_{eg}\Delta_k + \frac{1}{2}(\kappa_{eg}^r\Delta_{max} + J_{max})^2\Delta_k\right]\Delta_k, \tag{A.18}$$

and so we have (2.11) with $\kappa_{ef} = \kappa_{eg} + L_{\nabla f}/2 + (\kappa_{eg}^r\Delta_{max} + J_{max})^2/2$. □

B Geometry Improvement in Criticality Phase

Here, we describe the geometry-improvement step performed in the criticality phase of Algorithm 1, and prove its convergence. The proof of Lemma B.1 can be derived from the proof of [7, Lemma 10.5].

Algorithm 2 Geometry-Improvement for Criticality Phase

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

1: Set $Y_k^{(0)} = Y_k$.
2: **for** $i = 1, 2, \ldots$ **do**
3: Form $Y_k^{(i)}$ by modifying $Y_k^{(i-1)}$ until it is Λ-poised in $B(\mathbf{x}_k, \omega_C^{i-1}\Delta_k^{init})$.
4: Solve the interpolation system for $Y_k^{(i)}$ and form $m_k^{(i)}$ (2.5).
5: **if** $\omega_C^{i-1}\Delta_k^{init} \leq \mu\|\mathbf{g}_k^{(i)}\|$ **then**
6: **return** $Y_k^{(i)}$, $m_k^{(i)}$, $\Delta_k \leftarrow \omega_C^{i-1}\Delta_k^{init}$.
7: **end if**
8: **end for**

We recall from Lemma 3.4 that if $Y_k^{(i)}$ is Λ-poised in $B(\mathbf{x}_k, \omega_C^{i-1}\Delta_k^{init})$, then $m_k^{(i)}$ is fully linear in the sense of Definition 2.3, with associated constants κ_{ef} and κ_{eg} in (2.11) and (2.12) respectively given by (3.7).

Lemma B.1. *Suppose $\|\nabla f(\mathbf{x}_k)\| \geq \epsilon > 0$. Then for any $\mu > 0$ and $\omega_C \in (0,1)$, Algorithm 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*

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

Proof. First, suppose Algorithm 2 terminates on the first iteration. Then $\Delta_k = \Delta_k^{init}$, and the result holds.

Otherwise, consider some iteration i where Algorithm 2 does not terminate; that is, where $\omega_C^{i-1}\Delta_k^{init} > \mu\|\mathbf{g}_k^{(i)}\|$. Then since $m_k^{(i)}$ is fully linear in $B(\mathbf{x}_k, \omega_C^{i-1}\Delta_k^{init})$, we have

$$\epsilon \leq \|\nabla f(\mathbf{x}_k)\| \leq \|\nabla f(\mathbf{x}_k) - \mathbf{g}_k^{(i)}\| + \|\mathbf{g}_k^{(i)}\| \leq \left(\kappa_{eg} + \frac{1}{\mu}\right)\omega_C^{i-1}\Delta_k^{init}, \tag{B.2}$$

or equivalently

$$\omega_C^{i-1} \geq \frac{\epsilon}{(\kappa_{eg} + 1/\mu)\Delta_k^{init}}. \tag{B.3}$$

That is, if termination does not occur on iteration i, we must have

$$i \leq 1 + \frac{1}{|\log \omega_C|}\log\left(\frac{(\kappa_{eg} + 1/\mu)\Delta_k^{init}}{\epsilon}\right), \tag{B.4}$$

so Algorithm 2 terminates in finite time. We also have

$$\omega_C^{i-1}\Delta_k^{init} \geq \frac{\epsilon}{\kappa_{eg} + 1/\mu}, \tag{B.5}$$

from which (B.1) follows. □

C Calculating Geometry-Improving Steps

Here, we provide Algorithm 3, a method for solving one subproblem for calculating a geometry-improving step (4.1) subject to bound constraints. That is, we solve the problem

$$\mathbf{y}^+ := \underset{\mathbf{y} \in B(\mathbf{x}_k, \Delta_k)}{\arg\max}\ \mathbf{g}^\top \mathbf{y} \quad \text{subject to } \mathbf{a} \leq \mathbf{y} \leq \mathbf{b}. \tag{C.1}$$

Algorithm 3 Solve geometry-improving subproblem (C.1).

Initialize $\mathbf{y}_0 := \mathbf{x}_k$, step direction $\mathbf{s}_0 = -\mathbf{g}$ and inactive set for bounds $\mathcal{I} = \{1, \ldots, n\}$.
for $j = 0, 1, 2, \ldots, n-1$ **do**
  if $\|\mathbf{s}_j\| = 0$ **then**
    return $\mathbf{y}_j$.
  end if
  Find tentative step length $\alpha_j \geq 0$ by finding the largest solution to $\|\mathbf{y}_j + \alpha_j \mathbf{s}_j\|^2 = \Delta_k^2$.
  if $\mathbf{a} \leq \mathbf{y}_j + \alpha_j \mathbf{s}_j \leq \mathbf{b}$ **then**
    return $\mathbf{y}_{j+1} = \mathbf{y}_j + \alpha_j \mathbf{s}_j$.
  else
    Let $i \in \mathcal{I}$ be any index such that $y_{j,i} + \alpha_j s_{j,i} \notin [a_i, b_i]$.
    Let $\beta_j \leq \alpha_j$ be the largest value such that $y_{j,i} + \beta_j s_{j,i} \in [a_i, b_i]$
    Set $\mathbf{y}_{j+1} = \mathbf{y}_j + \beta_j \mathbf{s}_j$ and remove i from $\mathcal{I}$.
    Set $\mathbf{s}_{j+1} = \mathbf{s}_j$, except for setting $s_{j+1,i} = 0$.
  end if
end for
return $\mathbf{y}_n$.

Lemma C.1. *Suppose* $\mathbf{a} \leq \mathbf{x}_k \leq \mathbf{b}$ *and* $\mathbf{a} < \mathbf{b}$. *Then Algorithm 3 returns a minimizer of* (C.1).

Proof. Without loss of generality, we may assume that $\mathbf{x}_k = \mathbf{0}$, as otherwise we can simply shift $\mathbf{a} \to \mathbf{a} - \mathbf{x}_k$, and similarly for $\mathbf{b}$. Since (C.1) is convex, so it suffices to show convergence to a KKT point.

Suppose $\mathbf{y}$ is a KKT point where $y_i \in \{a_i, b_i\}$ for some $i \in \{1, \ldots, n\}$. If $y_i = a_i$, then if we took $\widetilde{\mathbf{y}} = \mathbf{y}$ except for $\widetilde{y}_i = b_i$, then

$$\mathbf{g}^\top \widetilde{\mathbf{y}} = \mathbf{g}^\top \mathbf{y} + g_i(b_i - a_i) \geq \mathbf{g}^\top \mathbf{y}, \tag{C.2}$$

since $\mathbf{y}$ is a global minimizer. Since $b_i - a_i > 0$ by assumption, we must have $g_i \geq 0$. Similarly, if $y_i = b_i$ is to hold at a KKT point, we require $g_i \leq 0$.

If $g_i = 0$, any value $y_i \in [a_i, b_i]$ gives the same objective value, so we can take $y_i = 0$. Given this, which Algorithm 3 provides, the KKT conditions reduce to: there exist μ and λ_i for $i = 1, \ldots, n$ such that (for all $i = 1, \ldots, n$, where relevant)

$$\mu \geq 0, \qquad \|\mathbf{y}\|^2 \leq \Delta^2, \tag{C.3a}$$

$$\lambda_i \geq 0, \qquad a_i \leq y_i \leq b_i, \tag{C.3b}$$

$$0 = \mu(\Delta^2 - \|\mathbf{y}\|^2), \tag{C.3c}$$

$$0 = \begin{cases} \lambda_i(b_i - x_i), & g_i > 0, \\ \lambda_i(x_i - a_i), & g_i < 0, \end{cases} \tag{C.3d}$$

$$\mathbf{g} = -\mu\mathbf{y} + \sum_{i=1}^{n} \lambda_i \begin{cases} \mathbf{e}_i, & g_i > 0, \\ -\mathbf{e}_i, & g_i < 0. \end{cases} \tag{C.3e}$$

Since $y_i = 0$ whenever $g_i = 0$, and we always write $\mathbf{g}$ as a sum of $\pm\mathbf{e}_i$, we only need to check $\mu \geq 0$ and $\lambda_i \geq 0$. For convenience, let $\gamma_j := \sum_{l=0}^{j} \beta_l$.

Consider the start of some iteration j, where currently $y_{j,i} = -\gamma_{j-1} g_i$ for all $i \in \mathcal{I}$. The equation for α_j is then

$$\sum_{i \notin \mathcal{I}} y_{j,i}^2 + \sum_{i \in \mathcal{I}} [(-\gamma_{j-1} - \alpha_j) g_i]^2 = \Delta^2, \tag{C.4}$$

for which the largest solution is

$$\alpha_j = -\gamma_{j-1} + \sqrt{\frac{\Delta^2 - \sum_{i \notin \mathcal{I}} y_{j,i}^2}{\sum_{i \in \mathcal{I}} g_i^2}}. \tag{C.5}$$

Hence $y_{j,i} = -\widetilde{\alpha}_j g_i$ for all $i \in \mathcal{I}$, where $\widetilde{\alpha}_j := \alpha_j + \gamma_{j-1} \geq 0$.

If Algorithm 3 terminates at line 8 for this iteration j, then we have $\mu = 1/\widetilde{\alpha}_j > 0$, and $y_i \in \{a_i, b_i\}$ for all $i \notin \mathcal{I}$. Thus the i-th component of (C.3e) is satisfied for all $i \in \mathcal{I}$ with $\lambda_i = 0$.

Now consider $i \notin \mathcal{I}$, and suppose $g_i > 0$. Then at some iteration $l < j$, we had $-\widetilde{\alpha}_l g_i < a_i \leq 0$, and set $y_{l+1,i} = a_i$, by choosing β_l so that $-\gamma_l g_i = a_i$. Thus, we have $\gamma_l \geq 0$. Similarly, if $g_i < 0$, we had $-\widetilde{\alpha}_l g_i > b_i \geq 0$, and needed β_l so that $-\gamma_l g_i = b_i \geq 0$, so we also need $\gamma_l \geq 0$ for this l. Thus $0 \leq \gamma_j \leq \widetilde{\alpha}_j$ for all j.

For termination at line 8 of Algorithm 3 and for (C.3e) to hold with $\lambda_i \geq 0$, we need $-\widetilde{\alpha}_j g_i \leq a_i$ or $-\widetilde{\alpha}_j g_i \geq b_i$, depending on the sign of g_i. Equivalently, we need $\widetilde{\alpha}_j \geq \gamma_l$ for all $l < j$. Since $\gamma_l \leq \widetilde{\alpha}_l$, it suffices to show that $\widetilde{\alpha}_{j-1} \leq \widetilde{\alpha}_j$ for all j.

To show this, let $\widetilde{\mathbf{y}}_{j-2}$ be the vector with the fixed components of $\mathbf{y}_{j-2}$ (i.e. those $i \notin \mathcal{I}$ at iteration $j-2$), and zeros otherwise. Then if i is the index removed from $\mathcal{I}$ at the end of iteration $j-1$, the equation for $\widetilde{\alpha}_{j-1}$ is

$$\|\mathbf{y}_{j-1} + \alpha_{j-1}\mathbf{s}_{j-1}\|^2 = \Delta^2, \tag{C.6}$$

$$\|\widetilde{\mathbf{y}}_{j-2}\|^2 + \widetilde{\alpha}_{j-1}^2 g_i^2 + \widetilde{\alpha}_{j-1}^2 \|\mathbf{s}_j\|^2 = \Delta^2. \tag{C.7}$$

Similarly, the equation for $\widetilde{\alpha}_j$ is

$$\|\widetilde{\mathbf{y}}_{j-2}\|^2 + c_i^2 + \widetilde{\alpha}_{j-1}^2 \|\mathbf{s}_j\|^2 = \Delta^2, \tag{C.8}$$

where $c_i \in \{a_i, b_i\}$ is whichever value we fixed $y_{j-1,i}$ to be. Equating (C.7) and (C.8) and using $|\widetilde{\alpha}_{j-1} g_i| \geq |c_i|$ (from above), we get

$$\widetilde{\alpha}_{j-1}^2 g_i^2 + \widetilde{\alpha}_{j-1}^2 \|\mathbf{s}_j\|^2 = c_i^2 + \widetilde{\alpha}_j^2 \|\mathbf{s}_j\|^2 \leq \widetilde{\alpha}_{j-1}^2 g_i^2 + \widetilde{\alpha}_j^2 \|\mathbf{s}_j\|^2, \tag{C.9}$$

and so using $\|\mathbf{s}_j\| > 0$ (from line 4 of Algorithm 3) and $\widetilde{\alpha}_j \geq 0$ for all j (shown above), we get $\widetilde{\alpha}_{j-1} \leq \widetilde{\alpha}_j$. This finishes the proof for when we terminate at line 8 of Algorithm 3.

Now suppose we terminate at line 16 of Algorithm 3. Then we have fixed $y_i \in \{a_i, b_i\}$ for all i (depending on the sign of g_i), with $\|\mathbf{y}\| \leq \Delta$ still valid. Hence we can have $\mu = 0$ (satisfying (C.3c)) and $\lambda_i = |g_i|$, and we have a KKT point. □

D Test Problems

#	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 with zero row & column)	7	35	4.989195×10^6	9.880597
6	Linear (rank 1 with zero row & column)	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 2. Details of test problems from [18], including the value of f^* used in (5.1) *for each problem. Note that, in line with the implementation of DFO-GN, we show $2f(\mathbf{x}_0)$ and $2f^*$; i.e. excluding the* 1/2 *factor in* (2.1).

E Medium-Scale Test Problems

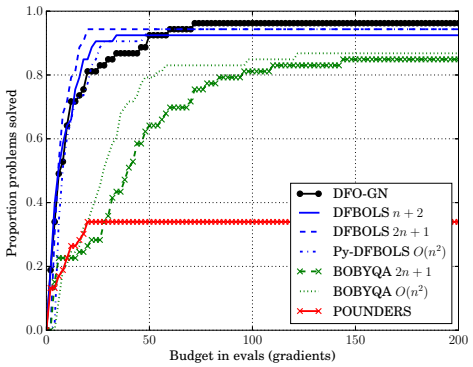
#	Problem	n	m	$2f(\mathbf{x}_0)$	$2f^*$	Parameters
1	ARGLALE	100	400	700	300	$N = 100$
2	ARGLBLE	100	400	5.460944×10^{14}	99.62547	$N = 100$
3	ARGTRIG	100	100	32.99641	0	$N = 100$
4	ARTIF	100	100	36.59115	0	$N = 100$
5	ARWHDNE	100	198	495	27.66203	$N = 100$
6	BDVALUES	100	100	1.943417×10^{7}	0	$NDP = 102$
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 3. Details of medium-scale test problems from the CUTEst test set (showing $2f(\mathbf{x}_0)$ and $2f^*$, as per Table 2). The set of problems are taken primarily from [12, 17, 16]. 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.*

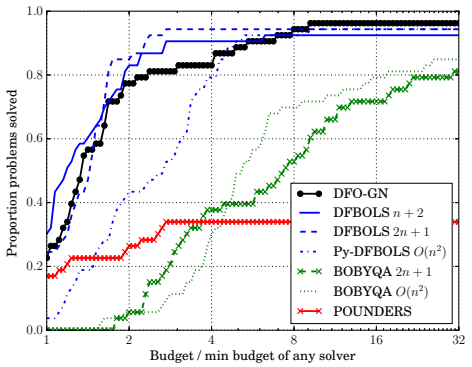
F Numerical Results for Higher Accuracy Levels

To supplement the results provided in Section 5, we provide here a number of the same comparisons of solvers, but for higher accuracy levels $\tau \in \{10^{-7}, 10^{-9}, 10^{-11}\}$, compared to $\tau = 10^{-5}$ in the main text. For each set of plots below, the title indicates the content of the plots and the figure in the main text with the corresponding results for $\tau = 10^{-5}$.

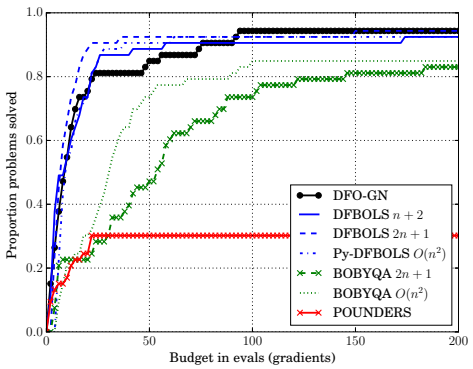
F.1 Moré & Wild test set — smooth objectives (compare to Figure 2)



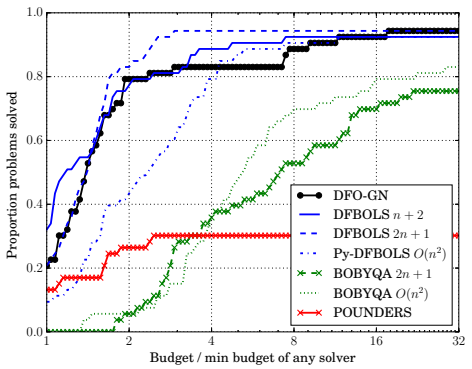
(a) Data profile, $\tau = 10^{-7}$



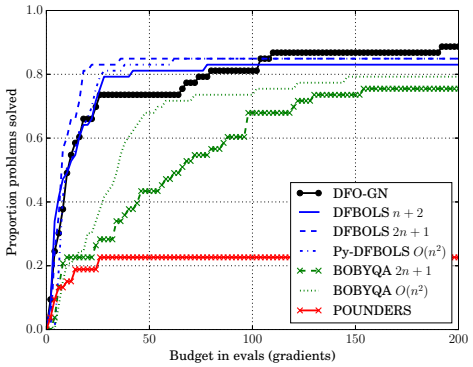
(b) Perf profile, $\tau = 10^{-7}$



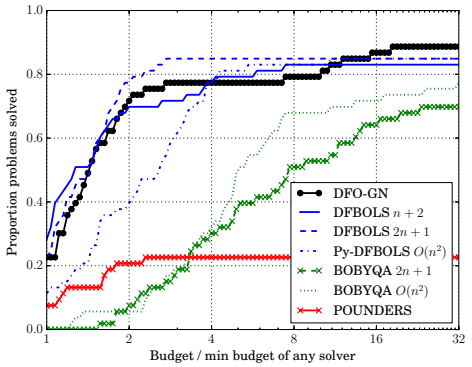
(c) Data profile, $\tau = 10^{-9}$



(d) Perf profile, $\tau = 10^{-9}$



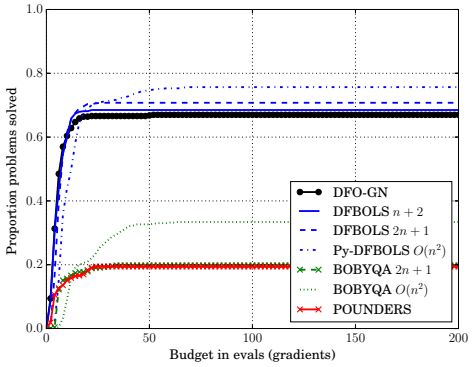
(e) Data profile, $\tau = 10^{-11}$



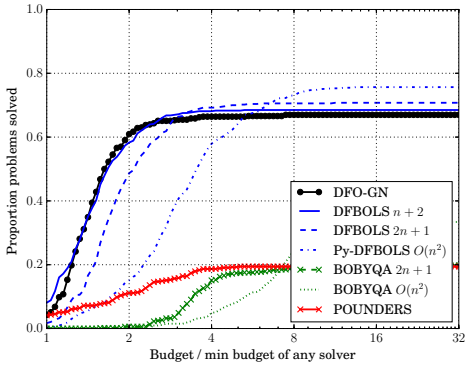
(f) Perf profile, $\tau = 10^{-11}$

Figure 7. Comparison of DFO-GN with BOBYQA, DFBOLS and POUNDERS for smooth objectives, to accuracy $\tau \in \{10^{-7}, 10^{-9}, 10^{-11}\}$. 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.

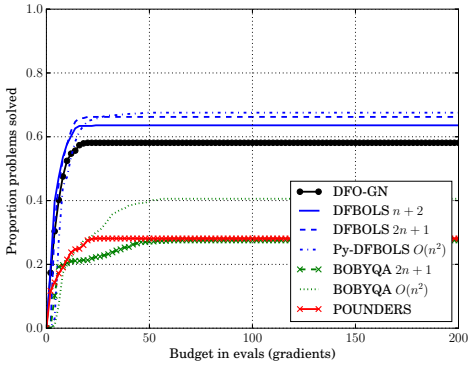
F.2 Moré & Wild test set — noisy objectives for $\tau = 10^{-7}$ (compare to Figure 3)



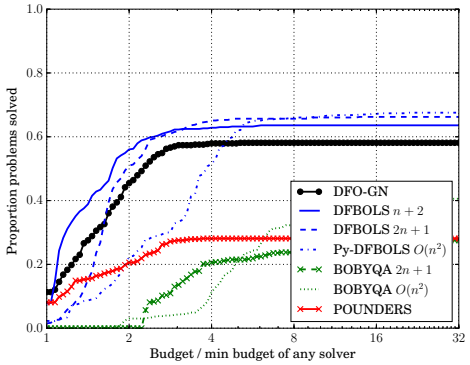
(a) Mult. Gaussian, data profile



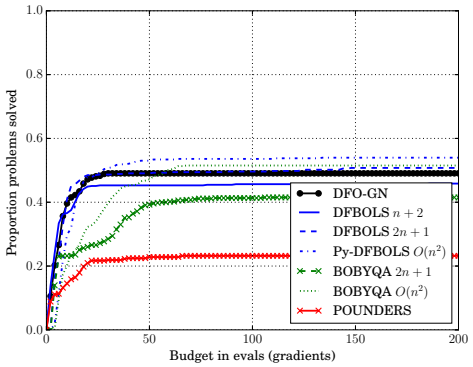
(b) Mult. Gaussian, performance profile



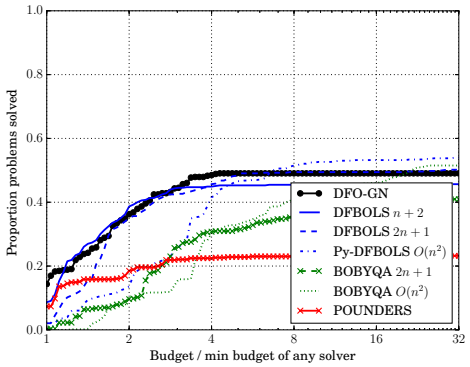
(c) Add. Gaussian, data profile



(d) Add. Gaussian, performance profile



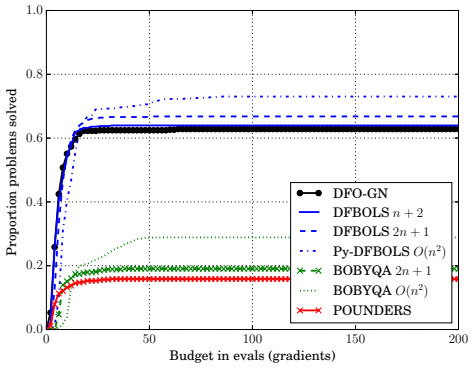
(e) Add. χ^2, data profile



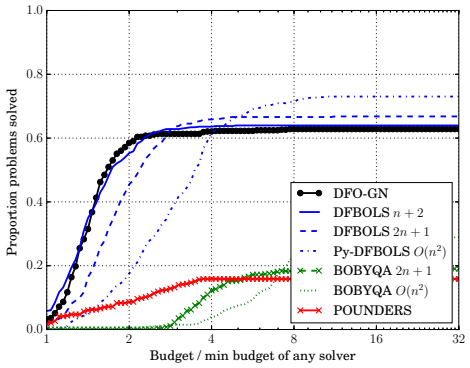
(f) Add. χ^2, performance profile

Figure 8. Comparison of DFO-GN with BOBYQA, DFBOLS and POUNDERS for objectives with multiplicative Gaussian, additive Gaussian and additive χ^2 noise with $\sigma = 10^{-2}$, to accuracy $\tau = 10^{-7}$ (average of 10 runs for each solver). 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.

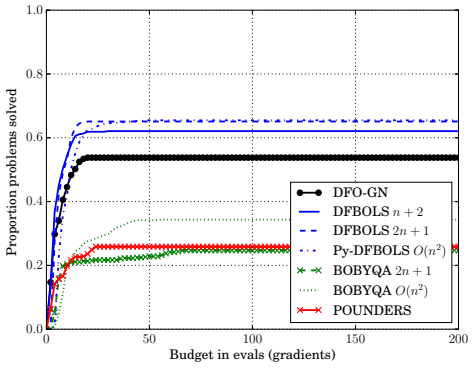
F.3 Moré & Wild test set — noisy objectives for $\tau = 10^{-9}$ (compare to Figure 3)



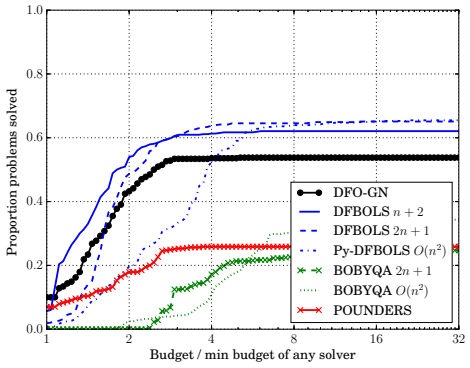
(a) Mult. Gaussian, data profile



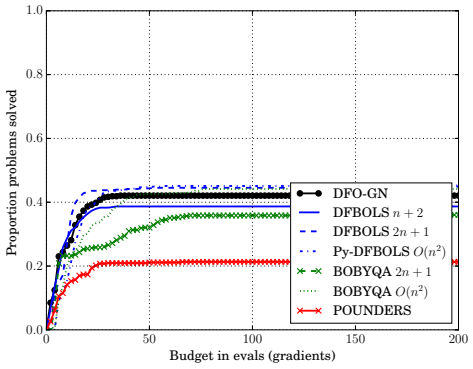
(b) Mult. Gaussian, performance profile



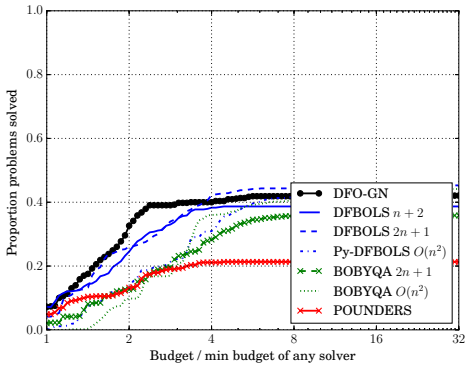
(c) Add. Gaussian, data profile



(d) Add. Gaussian, performance profile



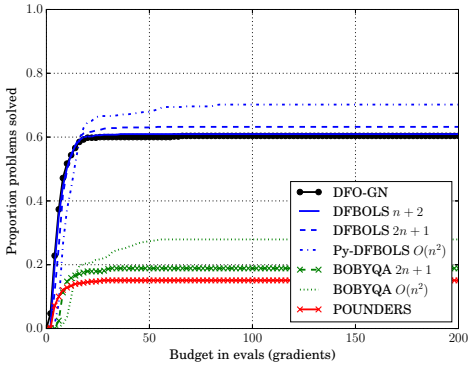
(e) Add. χ^2, data profile



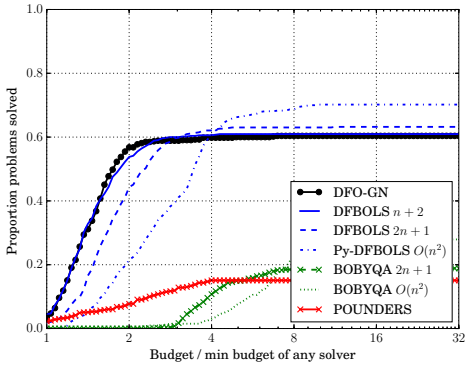
(f) Add. χ^2, performance profile

Figure 9. Comparison of DFO-GN with BOBYQA, DFBOLS and POUNDERS for objectives with multiplicative Gaussian, additive Gaussian and additive χ^2 noise with $\sigma = 10^{-2}$, to accuracy $\tau = 10^{-9}$ (average of 10 runs for each solver). 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.

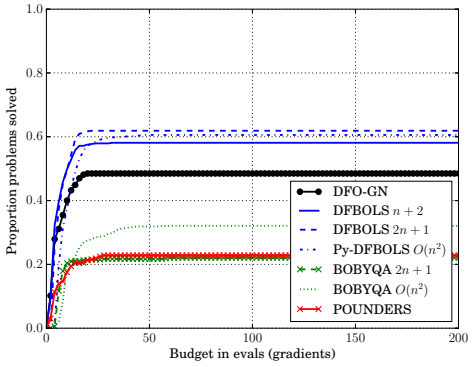
F.4 Moré & Wild test set — noisy objectives for $\tau = 10^{-11}$ (compare to Figure 3)



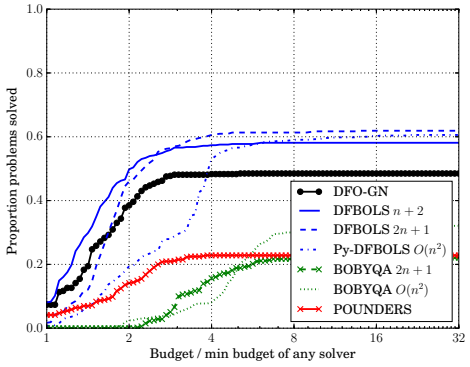
(a) Mult. Gaussian, data profile



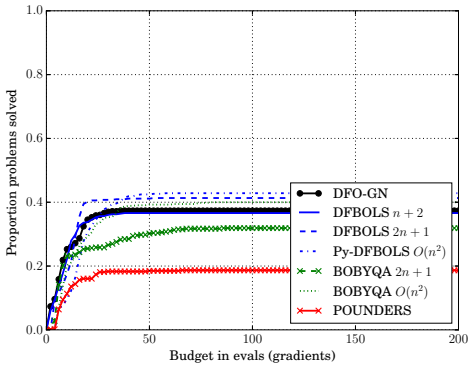
(b) Mult. Gaussian, performance profile



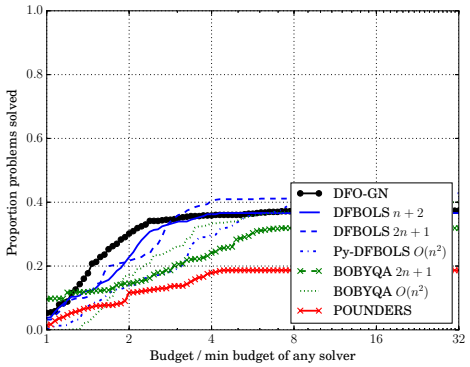
(c) Add. Gaussian, data profile



(d) Add. Gaussian, performance profile



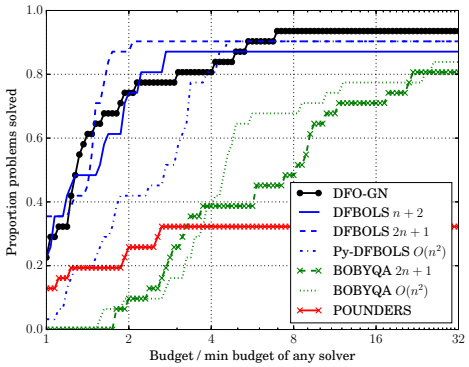
(e) Add. χ^2, data profile



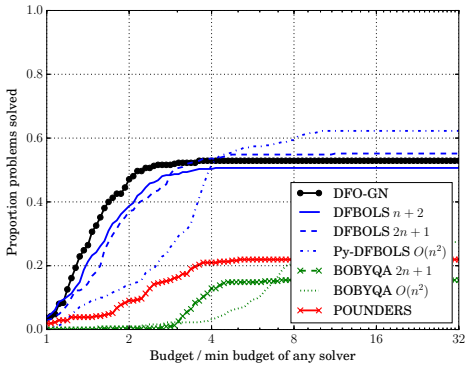
(f) Add. χ^2, performance profile

Figure 10. Comparison of DFO-GN with BOBYQA, DFBOLS and POUNDERS for objectives with multiplicative Gaussian, additive Gaussian and additive χ^2 noise with $\sigma = 10^{-2}$, to accuracy $\tau = 10^{-11}$ (average of 10 runs for each solver). 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.

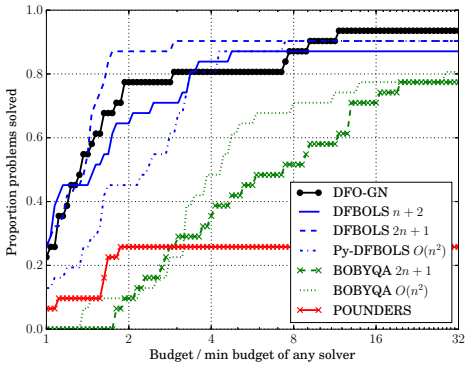
F.5 Moré & Wild test set — nonzero residual problems only (compare to Figure 4)



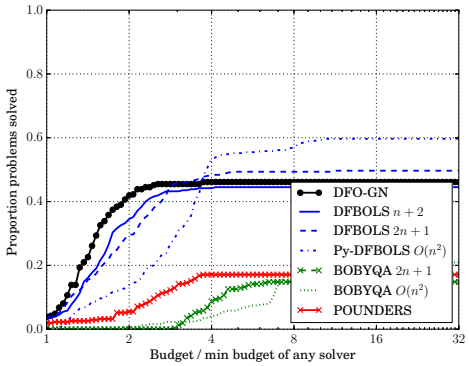
(a) Smooth objective, $\tau = 10^{-7}$



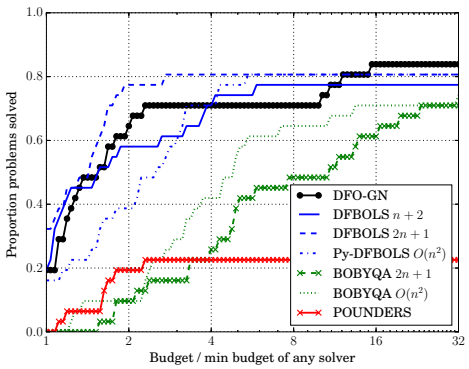
(b) Mult. Gaussian, $\tau = 10^{-7}$



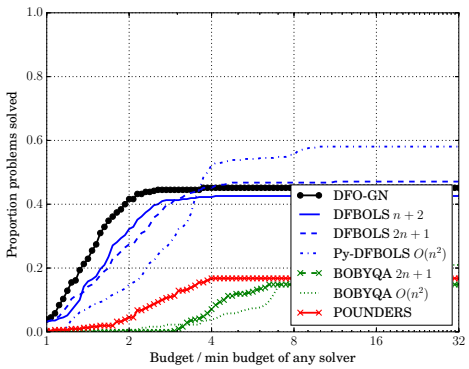
(c) Smooth objective, $\tau = 10^{-9}$



(d) Mult. Gaussian, $\tau = 10^{-9}$



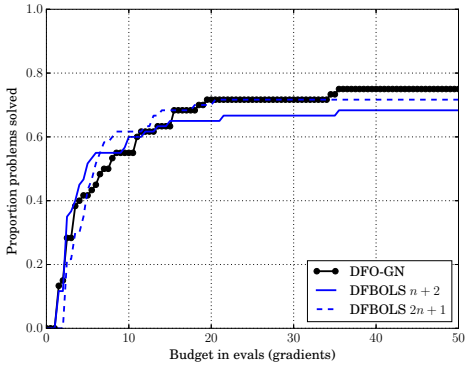
(e) Smooth objective, $\tau = 10^{-11}$



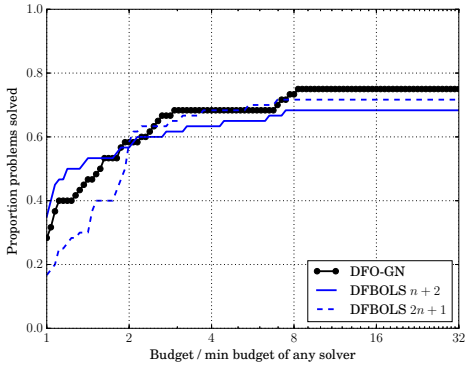
(f) Mult. Gaussian, $\tau = 10^{-11}$

Figure 11. Performance profile comparison of DFO-GN with BOBYQA, DFBOLS and POUNDERS for nonzero residual problems only, to accuracy $\tau \in \{10^{-5}, 10^{-9}, 10^{-11}\}$. 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 noisy objectives, results shown are an average of 10 runs for each solver.

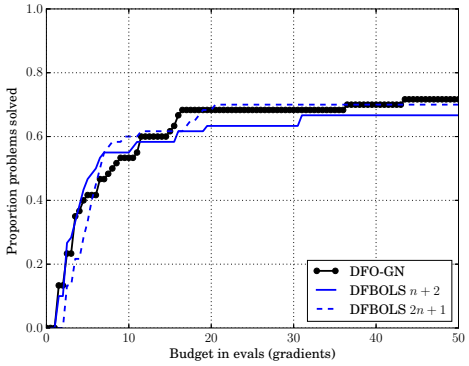
F.6 CUTEst test problems (compare to Figure 6)



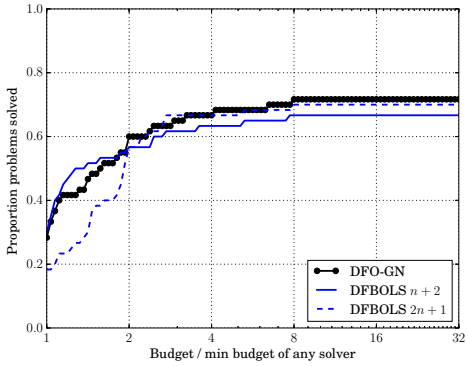
(a) Data profile, $\tau = 10^{-7}$



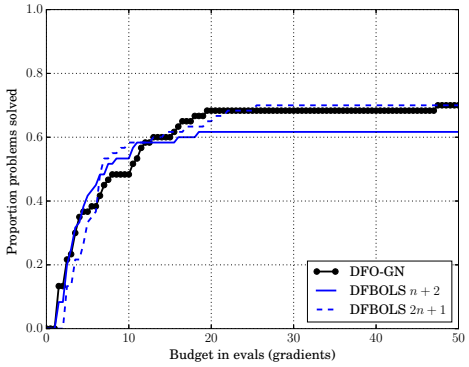
(b) Perf profile, $\tau = 10^{-7}$



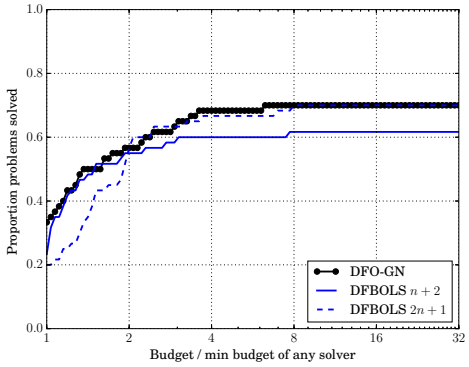
(c) Data profile, $\tau = 10^{-9}$



(d) Perf profile, $\tau = 10^{-9}$



(e) Data profile, $\tau = 10^{-11}$



(f) Perf profile, $\tau = 10^{-11}$

Figure 12. Comparison of DFO-GN with DFBOLS for smooth objectives from the set of medium-sized CUTEst problems, to accuracy $\tau \in \{10^{-7}, 10^{-9}, 10^{-11}\}$. For the DFBOLS runs, $n+2$, $2n+1$ are the number of interpolation points.