Mathematical Programming (2023) 199:461–524
https://doi.org/10.1007/s10107-022-01836-1

FULL LENGTH PAPER

Series A

Scalable subspace methods for derivative-free nonlinear least-squares optimization

Coralia Cartis[1] · Lindon Roberts[2]

Received: 23 February 2021 / Accepted: 10 May 2022 / Published online: 9 June 2022

Abstract
We introduce a general framework for large-scale model-based derivative-free optimization based on iterative minimization within random subspaces. We present a probabilistic worst-case complexity analysis for our method, where in particular we prove high-probability bounds on the number of iterations before a given optimality is achieved. This framework is specialized to nonlinear least-squares problems, with a model-based framework based on the Gauss–Newton method. This method achieves scalability by constructing local linear interpolation models to approximate the Jacobian, and computes new steps at each iteration in a subspace with user-determined dimension. We then describe a practical implementation of this framework, which we call DFBGN. We outline efficient techniques for selecting the interpolation points and search subspace, yielding an implementation that has a low per-iteration linear algebra cost (linear in the problem dimension) while also achieving fast objective decrease as measured by evaluations. Extensive numerical results demonstrate that DFBGN has improved scalability, yielding strong performance on large-scale nonlinear least-squares problems.

Keywords Derivative-free optimization · Large-scale optimization · Nonlinear least-squares · Worst case complexity

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. We note that the work in Sects. 4 and 5 originally appeared in the second author's thesis [69, Chapter 7].

✉ Lindon Roberts
lindon.roberts@anu.edu.au

Coralia Cartis
cartis@maths.ox.ac.uk

[1] Mathematical Institute, University of Oxford, Radcliffe Observatory Quarter, Woodstock Road, Oxford OX2 6GG, UK

[2] Mathematical Sciences Institute, Building 145, Science Road, Australian National University, Canberra, ACT 2601, Australia

Mathematics Subject Classification 65K05 · 90C30 · 90C56

1 Introduction

An important class of nonlinear optimization methods is so-called derivative-free optimization (DFO). In DFO, we consider problems where derivatives of the objective (and/or constraints) are not available to be evaluated, and we only have access to function values. This topic has received growing attention in recent years, and is primarily used for objectives which are black-box (so analytic derivatives or algorithmic differentiation are not available), and expensive to evaluate or noisy (so finite differencing is impractical or inaccurate). There are many types of DFO methods, such as model-based, direct and pattern search, implicit filtering and others (see [55] for a recent survey), and these techniques have been used in a variety of applications [1].

Here, we consider model-based DFO methods for unconstrained optimization, which are based on iteratively constructing and minimizing interpolation models for the objective. We also specialize these methods for nonlinear least-squares problems, by constructing interpolation models for each residual term rather than for the full objective [19, 79, 85].

This paper aims to provide a method that attempts to answer a key question regarding model-based DFO: how to improve the scalability of this class. Existing model-based DFO techniques are primarily designed for small- to medium-scale problems, as the linear algebra cost of each iteration—largely due to the cost of constructing interpolation models—means that their runtime increases rapidly for large problems. There are several settings where scalable DFO algorithms may be useful, such as data assimilation [3, 10], machine learning [39, 71], generating adversarial examples for deep neural networks [2, 75], image analysis [34], and as a possible proxy for global optimization methods [21].

To address this, we introduce RSDFO, a scalable algorithmic framework for model-based DFO. At each iteration of RSDFO we select a random low-dimensional subspace, build and minimize a model to compute a step in this space, then change the subspace at the next iteration. We provide a probabilistic worst-case complexity analysis of RSDFO. To our knowledge, this is the first *subspace model-based DFO method with global complexity and convergence guarantees*. We then describe how this general framework can be specialized to the case of nonlinear least-squares minimization through a model construction technique inspired by the Gauss–Newton method, yielding a new algorithm RSDFO-GN with associated worst-case complexity bounds. We then present an efficient implementation of RSDFO-GN, which we call DFBGN. DFBGN is available on Github[1] and includes several algorithmic features that yield strong performance on large-scale problems and a low per-iteration linear algebra cost that is typically linear in the problem dimension.

[1] https://github.com/numericalalgorithmsgroup/dfbgn.

1.1 Existing literature

The contributions in this paper are connected to several areas of research. We briefly review these topics below.

Block coordinate descent There is a large body of work on (derivative-based) block coordinate descent (BCD) methods, typically motivated by machine learning applications. BCD extends coordinate search methods [81] by updating a subset of the variables at each iteration, typically using a coordinate-wise variant of a first-order method. For nonconvex problems, the first convergence result for a randomized coordinate descent method based on proximal gradient descent was given in [62]. Here, the sampling of coordinates was uniform and required step sizes based on Lipschitz constants associated with the objective. This was extended in [57] to general randomized block selection with a nonomonotone linesearch-type method (to allow for unknown Lipschitz constants), and to a (possibly deterministic) 'essentially cyclic' block selection and extrapolation (but requiring Lipschitz constants) in [83]. Several extensions of this approach have been developed, including the use of stochastic gradients [82], parallel block updating [36] and inexact step calculations [36, 84].

BCD methods have been extended to nonlinear least-squares problems, leading to so-called Subspace Gauss–Newton methods. These are derivative-based methods where a Gauss–Newton step is computed for a subset of variables. This approach was initially proposed in [74] for parameter estimation in climate models—where derivative estimates were computed using implicit filtering [53]—and analyzed in quadratic regularization and trust-region settings for general unconstrained objectives in [16, 17, 72].

Sketching Sketching is an alternative dimensionality reduction technique for least-squares problems, reducing the number of residuals rather than the number of variables. Sketching ideas have been applied to linear [50, 58, 80] and nonlinear [35] least-squares problems, as well as model-based DFO for nonlinear least-squares [14], as well as subsampling algorithms for finite sum-of-functions minimization such as Newton's method [7, 70].

There are also alternative approaches to sketching which lead to subspace-type methods, where local gradients and Hessians are estimated only within a subspace (possibly used in conjunction with random subsampling). Sketching in this context has been applied to, for example, Newton's method [7, 44, 63], BFGS [43], and SAGA [45], as well as to trust-region and quadratic regularization methods [16, 17, 72].

Random embeddings for global optimization Some global optimization methods have been proposed which randomly project a high-dimensional problem into a low-dimensional subspace and solve this smaller problem using existing (global or local) methods. Though applicable to general global optimization problems (as a more sophisticated variant of random search), this technique has been explored particularly for defeating the curse of dimensionality when optimising functions which have low effective dimensionality [18, 68, 78]. For the latter class, often only one random subspace projection is needed, though the addition of constraints leads to multiple embeddings being required [18]. Our approach here differs from these works in both theoretical and numerical aspects, as it is focused on a specific random subspace technique for local optimization.

Probabilistic model-based DFO For model-based DFO, several algorithms have been developed and analyzed where the local model at each iteration is only sufficiently accurate with a certain probability [5, 12, 23]. Similar analysis also exists for derivative-based algorithms [22, 47]. Our approach is based on deterministic model-based DFO within subspaces, and we instead require a very weak probabilistic condition on the (randomly chosen) subspaces (Assumption 4).

Randomized direct search DFO In randomized direct search methods, iterates are perturbed in a random subset of directions (rather than a positive spanning set) when searching for local improvement. In this framework, effectively only a random subspace is searched in each iteration. Worst-case complexity bounds for this technique are given under predetermined step length regimes in [11, 40], and with adaptive step sizes in [46, 48], where [48] extends [46] to linearly constrained problems.

Large-scale DFO There have been several alternative approaches considered for improving the scalability of DFO. These often consider problems with specific structure which enable efficient model construction, such as partial separability [26, 64], sparse Hessians [4], and minimization over the convex hull of finitely many points [31]. On the other hand, there is a growing body of literature on 'gradient sampling' techniques for machine learning problems. These methods typically consider stochastic first-order methods but with a gradient approximation based on finite differencing in random directions [60], i.e. approximations of the form $\nabla f(\boldsymbol{x}) \approx \frac{f(\boldsymbol{x}+h\boldsymbol{u})-f(\boldsymbol{x})}{h}\boldsymbol{u}$ for a random Gaussian vector $\boldsymbol{u}$.[2] This framework has lead to variants of methods such as stochastic gradient descent [38], SVRG [56] and Adam [24], for example. We note that linear interpolation to orthogonal directions—more similar to traditional model-based DFO—has been shown to outperform gradient sampling as a gradient estimation technique [8, 9].

Subspace DFO methods A model-based DFO method with similarities to our subspace approach is the moving ridge function method from [49]. Here, existing objective evaluations are used to determine an 'active subspace' which captures the largest variability in the objective and build an interpolation model within this subspace. We also note the VXQR method from [61], which performs line searches along a direction chosen from a subspace determined by previous iterates. Both of these methods do not include convergence theory. By comparison, aside from our focus on nonlinear least-squares problems, both our general theoretical framework and our implemented method select their working subspaces randomly, and we provide (probabilistic) convergence guarantees. Lastly, the unpublished works [77, 86] propose a similar construction to ours, but based on full minimization of the objective within

[2] This is different from gradient sampling methods for nonsmooth optimization.

each subspace, and allowing potentially multiple simultaneous parallel subspace minimizations.

1.2 Contributions

We introduce RSDFO (Random Subspace Derivative-Free Optimization), a generic model-based DFO framework that relies on constructing a model in a subspace at each iteration. Our novel approach enables model-based DFO methods to be applied in a large-scale regime by giving the user explicit control over the subspace dimension, and hence control over the per-iteration linear algebra cost of the method. This framework is then specialized to the case of nonlinear least-squares problems, yielding a new algorithm RSDFO-GN (Random Subspace DFO with Gauss–Newton). The subspace model construction framework of RSDFO-GN is based on DFO Gauss–Newton methods [15, 19], and retains the same theoretical guarantees as RSDFO. We then describe a practical implementation of RSDFO-GN, which we call DFBGN (Derivative-Free Block Gauss–Newton).[3] Compared to existing methods, DFBGN reduces the linear algebra cost of model construction and the initial objective evaluation cost by allowing fewer interpolation points at every iteration. In order for DFBGN to have both scalability and a similar evaluation efficiency to existing methods (i.e. objective reduction achieved for a given number of objective evaluations), several modifications to the theoretical framework, regarding the selection of interpolation points and the search subspace, are necessary.

Theoretical results We consider a generic theoretical framework RSDFO, where the subspace dimension is a user-chosen algorithm hyperparameter, and no specific model construction approach is specified. Our framework is not specific to a least-squares problem structure, and holds for any objective with Lipschitz continuous gradient, and allows for a general class of random subspace constructions (not relying on a specific class of embeddings or projections). The theoretical results here extend the approach and techniques in [16, 17, 72] to model-based DFO methods. In particular, we use the notion of a well-aligned subspace (Definition 2) from [16, 17, 72], one in which sufficient decrease is achievable, and assume that our search subspace is well-aligned with some probability (Assumption 4). This is achieved provided we select a sufficiently large subspace dimension (depending on the desired failure probability and subspace alignment quality).

We derive a high probability worst-case complexity bound for RSDFO. Specifically, our main bounds are of the form $\mathbb{P}\left[\min_{j\leq k}\|\nabla f(\boldsymbol{x}_j)\| \leq Ck^{-1/2}\right] \geq 1-e^{-ck}$ and $\mathbb{P}\left[K_\epsilon \leq C\epsilon^{-2}\right] \leq 1-e^{-c\epsilon^{-2}}$, where K_ϵ is the first iteration to achieve first-order optimality ϵ (see Theorem 1 and Corollary 1). This result then implies a variety of alternative convergence results, such as expectation bounds and almost-sure convergence. Based on [16, 17, 54, 72], we give several constructions for determining our random subspace, and show that we can achieve convergence with a subspace dimension that is *independent of the ambient dimension*.

[3] Technically, DFBGN is not a block method as its subspaces are not coordinate-aligned, but has already been released with this name.

Our analysis matches the standard deterministic $\mathcal{O}(\epsilon^{-2})$ complexity bounds for model-based DFO methods built on linear interpolating models(e.g. [37]). However, when measuring the complexity in objective evaluations, our method yields a lower explicit dependency on the (ambient) problem dimension. Compared to the analysis of derivative-based methods (e.g. BCD [83] and probabilistically accurate models [22]) we need to incorporate the possibility that the interpolation model is not accurate (not fully linear, see Definition 1). However, unlike [5, 12, 23] we do not assume that full linearity is a stochastic property; instead, our stochasticity comes from the subspace selection and we explicitly handle non-fully linear models similar to [19, 29]. This gives us a framework which is similar to standard model-based DFO and with weak probabilistic conditions. Compared to the analysis of derivative-based random subspace methods in [16, 17, 72], our analysis is complicated substantially by the possibility of inaccurate models and the intricacies of model-based DFO algorithms. Although our approach could have considered situations where models are always guaranteed to be fully linear, we have developed our analysis to cope with this greater generality and to closely align with the traditional analysis of model-based DFO methods [19, 29]. The possibility of inaccurate models is similarly considered in [5, 12, 22, 23], but as an event that happens with some probability.

We then consider RSDFO-GN, which explicitly describes how interpolation models can be constructed for nonlinear least-squares problems, thus providing a concrete implementation of RSDFO in this context. Here we consider quadratic local models formed by linear interpolation for each residual function, which have strong practical performance [19]. We prove that RSDFO-GN retains the same $\mathcal{O}(\epsilon^{-2})$ complexity bound as RSDFO, again matching existing deterministic bounds [19]. However as in the general case, RSDFO-GN has an oracle complexity bound with a lower dependency on problem dimension compared to existing results.

In both cases, our subspace approach improves on existing oracle complexity analysis in terms of its dependency on problem dimension. However our method also benefits from a significantly reduced linear algebra cost per iteration, and so also improves on existing complexity bounds when measuring the algorithm's overall computational cost. For high-dimensional problems, both of these considerations (cost of objective evaluations and of linear algebra) are potentially relevant to overall algorithm performance.

Implementation Although it has beneficial evaluation and linear algebra complexity results, because of the random subspace framework, RSDFO-GN is not able to recycle objective evaluation information across multiple iterations. This is a key element of the practical success of model-based DFO methods tailored to the setting where objective evaluations are expensive. To address this, we introduce a practical, implementable variant of RSDFO-GN called DFBGN, which is based on the solver DFO-LS [15]. DFBGN achieves its practicality by using existing interpolation points to determine the relevant search subspace, coupled with a geometry-aware approach for selecting interpolation points for removal (inspired by the approach from [67]), and an adaptive randomized approach for selecting new interpolation points/subspace directions. We study the per-iteration linear algebra cost of DFBGN, and show that it is *linear in the problem dimension*, a substantial improvement over existing methods, which are cubic in the problem dimension, and equal to RSDFO-GN (although with significantly better practical performance than RSDFO-GN in terms of objective evaluations). This improvement comes from being able to perform almost all computations in the subspace, including model construction, step calculation and geometry-aware point removal. Our per-iteration linear algebra costs are also linear in the number of residuals, the same as existing methods, but with a substantially smaller constant (quadratic in the subspace dimension, which is user-determined, rather than quadratic in the problem dimension).

Numerical results We compare DFBGN with DFO-LS (which itself is shown to have state-of-the-art performance in [15]) on collections of both medium-scale (approx. 100 dimensions) and large-scale test problems (approx. 1000 dimensions). We show that DFBGN with a full-sized subspace has similar performance to DFO-LS in terms of objective evaluations, but shows improved performance on runtime.[4] This indicates that DFBGN's practical approach for recycling objective evaluations across iterations yields state-of-the-art performance while inheriting the low linear algebra cost of RSDFO-GN. As the dimension of the subspace reduces (i.e. the size of the interpolation set reduces), we demonstrate a tradeoff between reduced linear algebra costs and increased evaluation counts required to achieve a given objective reduction. The flexibility of DFBGN allows this tradeoff to be explicitly managed. When tested on large-scale problems, DFO-LS frequently reaches a reasonable runtime limit without making substantial progress, whereas DFBGN with small subspace size can perform many more iterations and hence make better progress than DFO-LS. In the case of expensive objectives with small evaluation budgets, we show that DFBGN can make progress with few objective evaluations in a similar way to DFO-LS (which has a mechanism to make progress from as few as 2 objective evaluations independent of problem dimension), but with substantially lower linear algebra costs.

Structure of paper In Sect. 2 we describe RSDFO and provide our probabilistic worst-case complexity analysis. We specialize RSDFO to RSDFO-GN in Sect. 3. Then we describe the practical implementation DFBGN and its features in Sect. 4. Our numerical results are given in Sect. 5.

Implementation A Python implementation of DFBGN is available on Github.[5]

Notation We use $\|\cdot\|$ to refer to the Euclidean norm of vectors and the operator 2-norm of matrices, and $B(\boldsymbol{x}, \Delta)$ for $\boldsymbol{x} \in \mathbb{R}^n$ and $\Delta > 0$ to be the closed ball $\{\boldsymbol{y} \in \mathbb{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\| \leq \Delta\}$.

[4] The main difference between DFBGN with a full-sized subspace (i.e. the subspace dimension is the same as the problem dimension) and DFO-LS is the way that interpolation points are added/removed between iterations, with DFBGN using the approach described in Sect. 4.4, but there are also small differences between the two algorithms in trust-region management, for example.

[5] https://github.com/numericalalgorithmsgroup/dfbgn.

2 Random subspace model-based DFO

In this section we outline our general model-based DFO algorithmic framework based on minimization in random subspaces. We consider the nonconvex problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}), \tag{1}$$

where we assume that $f:\mathbb{R}^n\to\mathbb{R}$ is continuously differentiable, but that access to its gradient is not possible (e.g. for the reasons described in Sect. 1). In a standard model-based DFO framework (e.g. [30, 55]), at each iteration k we construct a quadratic model $m_k:\mathbb{R}^n\to\mathbb{R}$ which approximates f near our iterate $\boldsymbol{x}_k$:

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

for some $\boldsymbol{g}_k\in\mathbb{R}^n$ and $H_k\in\mathbb{R}^{n\times n}$ symmetric. Based on this model, we build a globally convergent algorithm using a trust-region framework [27]. This algorithmic framework is suitable providing that—when necessary—we can guarantee m_k is a sufficiently accurate model for f near $\boldsymbol{x}_k$. Details about how to construct sufficiently accurate models based on interpolation are given in [28, 30].

Our core idea here is to construct interpolation models which only approximate the objective in a subspace, rather than in the full space $\mathbb{R}^n$. This allows us to use interpolation sets with fewer points, since we do not have to capture the objective's behaviour outside our subspace, which improves the scalability of the method.

In this section, we outline our general algorithmic framework and provide a worst-case complexity analysis showing convergence to first-order stationary points with high probability. We then describe how this framework may be specialized to the case of nonlinear least-squares minimization.

2.1 RSDFO algorithm

In our general framework, which we call RSDFO (Random Subspace DFO), we modify the above approach by randomly choosing a p-dimensional subspace (where $p\le n$ is user-chosen) and constructing an interpolation model defined only in that subspace.[6] Specifically, in each iteration k we randomly choose a p-dimensional affine space $\mathcal{Y}_k\subset\mathbb{R}^n$ given by the range of $Q_k\in\mathbb{R}^{n\times p}$, i.e.

$$\mathcal{Y}_k=\{\boldsymbol{x}_k+Q_k\hat{\boldsymbol{s}}:\hat{\boldsymbol{s}}\in\mathbb{R}^p\}. \tag{3}$$

We then construct a model which interpolates f at points in $\mathcal{Y}_k$ and ultimately construct a local quadratic model for f only on $\mathcal{Y}_k$. That is, given Q_k, we assume that

[6] Formally, we define our model in an affine space, but we call it a subspace throughout as this fits with an intuitive view of what RSDFO aims to achieve.

we have $\hat{m}_k : \mathbb{R}^p \to \mathbb{R}$ given by

$$f(\boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}}) \approx \hat{m}_k(\hat{\boldsymbol{s}}) := f(\boldsymbol{x}_k) + \hat{\boldsymbol{g}}_k^T\hat{\boldsymbol{s}} + \frac{1}{2}\hat{\boldsymbol{s}}^T\hat{H}_k\hat{\boldsymbol{s}}, \tag{4}$$

where $\hat{\boldsymbol{g}}_k \in \mathbb{R}^p$ and $\hat{H}_k \in \mathbb{R}^{p\times p}$ are the low-dimensional model gradient and Hessian respectively, adopting the convention of using hats on variables to denote low-dimensional quantities. In Sect. 3 we specialize this to a model construction process for nonlinear least-squares problems.

For our trust-region algorithm, we (approximately) minimize $\hat{m}_k$ inside the trust region to get a tentative step

$$\hat{\boldsymbol{s}}_k \approx \arg\min_{\hat{\boldsymbol{s}}\in\mathbb{R}^p} \hat{m}_k(\hat{\boldsymbol{s}}), \quad \text{s.t.} \quad \|\hat{\boldsymbol{s}}\| \le \Delta_k, \tag{5}$$

for the current trust-region radius $\Delta_k > 0$, yielding a tentative step $\boldsymbol{s}_k = Q_k\hat{\boldsymbol{s}}_k \in \mathbb{R}^n$. We thus also get the computational advantage coming from solving a p-dimensional trust-region subproblem.

In our setting we are only interested in the approximation properties of $\hat{m}_k$ in the space $\mathcal{Y}_k$, and so we introduce the following notion of a "sufficiently accurate" model:

Definition 1 Given $Q \in \mathbb{R}^{n\times p}$, a model $\hat{m} : \mathbb{R}^p \to \mathbb{R}$ is Q-fully linear in $B(\boldsymbol{x}, \Delta) \subset \mathbb{R}^n$ if

$$|f(\boldsymbol{x} + Q\hat{\boldsymbol{s}}) - \hat{m}(\hat{\boldsymbol{s}})| \le \kappa_{\text{ef}}\Delta^2, \tag{6a}$$

$$\|Q^T\nabla f(\boldsymbol{x} + Q\hat{\boldsymbol{s}}) - \nabla\hat{m}(\hat{\boldsymbol{s}})\| \le \kappa_{\text{eg}}\Delta, \tag{6b}$$

for all $\boldsymbol{s} \in \mathbb{R}^p$ with $\|\hat{\boldsymbol{s}}\| \le \Delta$. The constants κ_{ef} and κ_{eg} must be independent of Q, $\hat{m}$, $\boldsymbol{x}$ and Δ.

The gradient condition (6b) comes from noting that if $\hat{f}(\hat{\boldsymbol{s}}) := f(\boldsymbol{x} + Q\hat{\boldsymbol{s}})$ then $\nabla\hat{f}(\hat{\boldsymbol{s}}) = Q^T\nabla f(\boldsymbol{x} + Q\hat{\boldsymbol{s}})$. We note that if we have full-dimensional subspaces $p = n$ and take $Q = I$, then we recover the standard notion of fully linear models [29, Definition 3.1]. In our analysis, we will generally assume that Definition 1 is satisfied by finding $\hat{\boldsymbol{g}}_k$ using linear interpolation and taking $\hat{H}_k$ to be zero, but underdetermined quadratic interpolation techniques could instead be used [28, 30].

Complete RSDFO algorithm The complete RSDFO algorithm is stated in Algorithm 1. The overall structure is common to model-based DFO methods [29, 30]. In particular, we assume that we have procedures to verify whether or not a model is Q_k-fully linear in $B(\boldsymbol{x}_k, \Delta_k)$ and (if not) to generate a Q_k-fully linear model. When we specialize RSDFO to nonlinear least-squares problems in Sect. 3, we will describe how we can obtain such procedures.

The broad structure of RSDFO is as follows:

1. First generate a subspace Q_k and, by linear interpolation on a new set of points in the subspace, generate an interpolating model $\hat{m}_k$.

Algorithm 1 RSDFO (Random Subspace Derivative-Free Optimization) for solving (1).

```
Input: Starting point $x_0 \in \mathbb{R}^n$, initial trust region radius $\Delta_0 > 0$, and subspace dimension $p \in \{1,\ldots,n\}$.

  Parameters: maximum trust-region radius $\Delta_{\max} \geq \Delta_0$, trust-region radius scalings $0 < \gamma_{\text{dec}} < 1 < \gamma_{\text{inc}} \leq \overline{\gamma}_{\text{inc}}$, criticality constants $\epsilon_C, \mu > 0$ and trust-region scaling $0 < \gamma_C < 1$, safety step threshold $\beta_F > 0$ and trust-region scaling $0 < \gamma_F < 1$, and acceptance thresholds $0 < \eta_1 \leq \eta_2 < 1$.

1: Set flag CHECK_MODEL=FALSE.
2: for $k = 0, 1, 2, \ldots$ do
3:   if CHECK_MODEL=TRUE then
4:     Set $Q_k = Q_{k-1}$.
5:     Construct a reduced model $\hat{m}_k : \mathbb{R}^p \to \mathbb{R}$ (4) which is $Q_k$-fully linear in $B(x_k, \Delta_k)$.
6:   else
7:     Define a subspace by randomly sampling $Q_k \in \mathbb{R}^{n\times p}$.
8:     Construct a reduced model $\hat{m}_k : \mathbb{R}^p \to \mathbb{R}$ (4) which need not be $Q_k$-fully linear.
9:   end if
10:  if $\|\hat{g}_k\| < \epsilon_C$ and ($\|\hat{g}_k\| < \mu^{-1}\Delta_k$ or $\hat{m}_k$ is not $Q_k$-fully linear in $B(x_k, \Delta_k)$) then
11:    (Criticality step) Set $x_{k+1} = x_k$, $\Delta_{k+1} = \gamma_C \Delta_k$ and CHECK_MODEL=TRUE.
12:  else ← $\|\hat{g}_k\| \geq \epsilon_C$ or ($\|\hat{g}_k\| \geq \mu^{-1}\Delta_k$ and $\hat{m}_k$ is $Q_k$-fully linear in $B(x_k, \Delta_k)$)
13:    Approximately solve the subspace trust-region subproblem in $\mathbb{R}^p$ (5) and calculate the step $s_k = Q_k \hat{s}_k \in \mathbb{R}^n$.
14:    if $\|\hat{s}_k\| < \beta_F \Delta_k$ then
15:      (Safety step) Set $x_{k+1} = x_k$ and $\Delta_{k+1} = \gamma_F \Delta_k$.
16:      If $\hat{m}_k$ is $Q_k$-fully linear in $B(x_k, \Delta_k)$, then set CHECK_MODEL=FALSE, otherwise CHECK_MODEL=TRUE.
17:    else
18:      Evaluate $f(x_k + s_k)$ and calculate ratio

             $$\rho_k := \frac{f(x_k) - f(x_k + s_k)}{\hat{m}_k(\mathbf{0}) - \hat{m}_k(\hat{s}_k)}. \qquad (7)$$

19:      Accept/reject step and update trust region radius: set

             $$x_{k+1} = \begin{cases} x_k + s_k, & \rho_k \geq \eta_1, \\ x_k, & \rho_k < \eta_1, \end{cases} \quad \text{and} \quad \Delta_{k+1} = \begin{cases} \min(\max(\gamma_{\text{inc}}\Delta_k, \overline{\gamma}_{\text{inc}}\|\hat{s}_k\|), \Delta_{\max}), & \rho_k \geq \eta_2, \\ \max(\gamma_{\text{dec}}\Delta_k, \|\hat{s}_k\|), & \eta_1 \leq \rho_k < \eta_2, \\ \min(\gamma_{\text{dec}}\Delta_k, \|\hat{s}_k\|), & \rho_k < \eta_1. \end{cases} \qquad (8)$$

20:      If $\rho_k \geq \eta_2$ or $\hat{m}_k$ is $Q_k$-fully linear in $B(x_k, \Delta_k)$, then set CHECK_MODEL=FALSE, otherwise set CHECK_MODEL=TRUE.
21:    end if
22:  end if
23: end for
```

2. If we suspect we are close to first-order stationarity, perform one iteration of a criticality step [29, 30] to ensure we have an accurate model and appropriately sized trust-region radius.
3. Compute a step by solving the trust-region subproblem (5).
4. Evaluate the quality of the step and use this to determine the new iterate x_{k+1} and trust-region radius Δ_{k+1}. Our updating mechanism follows [19, 67]. In particular, we consider a very short step to be unsuccessful and invoke a safety step [65] without evaluating $f(x_k + s_k)$.

An important feature of RSDFO is that in some iterations, we reuse the previous subspace, $\mathcal{Y}_k = \mathcal{Y}_{k-1}$, corresponding to the flag `CHECK_MODEL=TRUE`. In this case, we had an inaccurate model in iteration $k-1$ and require that our new model $\hat{m}_k$ is accurate (Q_k-fully linear). This mechanism essentially ensures that Δ_k is not decreased too quickly as a result of inaccurate models, and is mostly decreased to achieve sufficient objective reduction.

We now give our convergence and worst-case complexity analysis of Algorithm 1. For brevity, we defer proofs based on standard model-based DFO techniques to Appendix A.

2.2 Assumptions and preliminary results

We begin our analysis with some basic assumptions and preliminary results.

Assumption 1 (*Smoothness*) The objective function $f : \mathbb{R}^n \to \mathbb{R}$ is bounded below by f_{low} and continuously differentiable, and ∇f is $L_{\nabla f}$-Lipschitz continuous in the level set $\{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) \le f(\boldsymbol{x}_0)\}$, for some constant $L_{\nabla f} > 0$.

We also need two standard assumptions for trust-region methods: uniformly bounded above model Hessians and sufficiently accurate solutions to the trust-region subproblem (5).

Assumption 2 (*Bounded model Hessians*) We assume that $\|\hat{H}_k\| \le \kappa_H$ for all k, for some $\kappa_H \ge 1$.

Assumption 3 (*Cauchy decrease*) Our method for solving the trust-region subproblem (5) gives a step $\hat{\boldsymbol{s}}_k$ satisfying the sufficient decrease condition

$$\hat{m}_k(\boldsymbol{0}) - \hat{m}_k(\hat{\boldsymbol{s}}_k) \ge c_1 \|\hat{\boldsymbol{g}}_k\| \min\left(\Delta_k, \frac{\|\hat{\boldsymbol{g}}_k\|}{\max(\|\hat{H}_k\|, 1)}\right), \tag{9}$$

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

A useful consequence, needed for the analysis of our trust-region radius updating scheme, is the following.

Lemma 1 (Lemma 3.6, [19]) *Suppose Assumption 3 holds. Then*

$$\|\hat{\boldsymbol{s}}_k\| \ge c_2 \min\left(\Delta_k, \frac{\|\hat{\boldsymbol{g}}_k\|}{\max(\|\hat{H}_k\|, 1)}\right), \tag{10}$$

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

Lemma 2 *Suppose Assumptions 2 and 3 hold, and we run RSDFO with* $\beta_F \le c_2$ *(where* c_2 *is introduced in Lemma 1). If* $\hat{m}_k$ *is* Q_k*-fully linear in* $B(\boldsymbol{x}_k, \Delta_k)$ *and*

$$\Delta_k \le c_0 \|\hat{\boldsymbol{g}}_k\|, \qquad \textit{where} \qquad c_0 := \min\left(\mu, \frac{1}{\kappa_H}, \frac{c_1(1-\eta_2)}{2\kappa_{\text{ef}}}\right), \tag{11}$$

then the criticality and safety steps are not called, and $\rho_k \ge \eta_2$.

Proof See Appendix A.1. □

Remark 1 The requirement $\beta_F \le c_2$ in Lemma 2 is not restrictive. Since have $c_1 \ge 1/2$ in Assumption 3, it suffices to choose $\beta_F \le \sqrt{2} - 1$, for example.

Our key new assumption is on the quality of our subspace selection, as introduced in [16, 17, 72]:

Definition 2 The matrix Q_k is well-aligned if

$$\|Q_k^T \nabla f(\boldsymbol{x}_k)\| \ge \alpha_Q \|\nabla f(\boldsymbol{x}_k)\|, \tag{12}$$

for some $\alpha_Q > 0$ independent of k.

Assumption 4 (*Subspace quality*) Our subspace selection (determined by Q_k) satisfies the following two properties:

(a) At each iteration k of RSDFO in which `CHECK_MODEL = FALSE`, our subspace selection Q_k is well-aligned for some fixed $\alpha_Q > 0$ with probability at least $1 - \delta_S$, for some $\delta_S \in (0, 1)$, independently of $\{Q_0, \dots, Q_{k-1}\}$.
(b) $\|Q_k\| \le Q_{\max}$ for all k and some $Q_{\max} > 0$.

Of these two properties, (a) is needed for our complexity analysis, while (b) is only needed in order to construct Q_k-fully linear models (in Sect. 3). Note that if Assumption 4 holds then (12) and $\|Q_k\| \le Q_{\max}$ together imply that we must have $\alpha_Q \le Q_{\max}$. The constructions we will consider will be based on $\alpha_Q \in (0, 1)$. We will discuss how to achieve Assumption 4 in more detail in Sect. 2.6.

Lemma 3 *In all iterations k of RSDFO where the criticality step is not called, we have $\|\hat{\boldsymbol{g}}_k\| \ge \min(\epsilon_C, \mu^{-1}\Delta_k)$. If the criticality step is not called in iteration k, Q_k is well-aligned and $\|\nabla f(\boldsymbol{x}_k)\| \ge \epsilon$, then*

$$\|\hat{\boldsymbol{g}}_k\| \ge \epsilon_g(\epsilon) := \min\left(\epsilon_C, \frac{\alpha_Q \epsilon}{\kappa_{\text{eg}}\mu + 1}\right) > 0. \tag{13}$$

Proof See Appendix A.2. □

2.3 Counting iterations

We now provide a series of results counting the number of iterations of RSDFO of different types, following the style of analysis from [17, 22, 72]. First we introduce some notation to enumerate our iterations. Suppose we run RSDFO until the end of iteration K. We then define the following subsets of $\{0, \dots, K\}$:

- $\mathcal{C}$ is the set of iterations in $\{0, \dots, K\}$ where the criticality step is called.
- $\mathcal{F}$ is the set of iterations in $\{0, \dots, K\}$, where the safety step is called (i.e. $\|\hat{\boldsymbol{s}}_k\| < \beta_F \Delta_k$).
- $\mathcal{VS}$ is the set of very successful iterations in $\{0, \dots, K\}$, where $\rho_k \ge \eta_2$.

- $\mathcal{S}$ is the set of successful iterations in $\{0, \ldots, K\}$, where $\rho_k \geq \eta_1$. Note that $\mathcal{VS} \subset \mathcal{S}$.
- $\mathcal{U}$ is the set of unsuccessful iterations in $\{0, \ldots, K\}$, where $\rho_k < \eta_1$.
- $\mathcal{A}$ is the set of well-aligned iterations in $\{0, \ldots, K\}$, where (12) holds.
- $\mathcal{A}^C$ is the set of poorly aligned iterations in $\{0, \ldots, K\}$, where (12) does not hold.
- $\mathcal{D}(\Delta)$ is the set of iterations in $\{0, \ldots, K\}$ where $\Delta_k \geq \Delta$ for some $\Delta > 0$.
- $\mathcal{D}^C(\Delta)$ is the set of iterations in $\{0, \ldots, K\}$ where $\Delta_k < \Delta$.
- $\mathcal{L}$ is the set of iterations in $\{0, \ldots, K\}$ where $\hat{m}_k$ is Q_k-fully linear in $B(\boldsymbol{x}_k, \Delta_k)$.
- $\mathcal{L}^C$ is the set of iterations in $\{0, \ldots, K\}$ where $\hat{m}_k$ is not Q_k-fully linear in $B(\boldsymbol{x}_k, \Delta_k)$.

In particular, we have the partitions, for any $\Delta > 0$,

$$\{0, \ldots, K\} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{S} \cup \mathcal{U} = \mathcal{A} \cup \mathcal{A}^C = \mathcal{D}(\Delta) \cup \mathcal{D}^C(\Delta) = \mathcal{L} \cup \mathcal{L}^C. \quad (14)$$

First, we bound the number of successful iterations with large Δ_k using standard arguments from trust-region methods. Throughout, we use $\#(\cdot)$ to refer to the cardinality of a set of iterations.

Lemma 4 *Suppose Assumptions 1, 2 and 3 hold. If* $\|\nabla f(\boldsymbol{x}_k)\| \geq \epsilon$ *for all* $k = 0, \ldots, K$*, then*

$$\#(\mathcal{A} \cap \mathcal{D}(\Delta) \cap \mathcal{S}) \leq \phi(\Delta, \epsilon) := \frac{f(\boldsymbol{x}_0) - f_{\text{low}}}{\eta_1 c_1 \epsilon_g(\epsilon) \min(\epsilon_g(\epsilon)/\kappa_H, \Delta)}, \quad (15)$$

for all $\Delta > 0$.

Proof See Appendix A.3. □

Lemma 5 *Suppose Assumptions 1, 2 and 3 hold, and* $\beta_F \leq c_2$*. If* $\|\nabla f(\boldsymbol{x}_k)\| \geq \epsilon$ *for all* $k = 0, \ldots, K$*, then*

$$\#(\mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L} \backslash \mathcal{VS}) = 0, \quad (16)$$

for all $\Delta > 0$ *satisfying*

$$\Delta \leq \Delta^*(\epsilon) := \min\left(\mu\epsilon_g(\epsilon), \frac{\epsilon_g(\epsilon)}{\kappa_H}, \left(\kappa_{\text{eg}} + \frac{2\kappa_{\text{ef}}}{c_1(1-\eta_2)}\right)^{-1} \alpha_Q \epsilon, \frac{\alpha_Q \epsilon}{\kappa_{\text{eg}} + \mu^{-1}}\right). \quad (17)$$

Proof See Appendix A.4. □

Lemma 6 *Suppose Assumptions 1, 2 and 3 hold. Then we have*

$$\#(\mathcal{D}(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\Delta) \backslash \mathcal{S}) \leq C_1 \#(\mathcal{D}(\overline{\gamma}_{\text{inc}}^{-1}\Delta) \cap \mathcal{S}) + C_2, \quad (18)$$

for all $\Delta \leq \Delta_0$, where

$$C_1 := \frac{\log(\overline{\gamma}_{\text{inc}})}{\log(1/\max(\gamma_C, \gamma_F, \gamma_{\text{dec}}))} \quad \text{and} \quad C_2 := \frac{\log(\Delta_0/\Delta)}{\log(1/\max(\gamma_C, \gamma_F, \gamma_{\text{dec}}))}. \tag{19}$$

Proof See Appendix A.5. □

Lemma 7 *Suppose Assumptions 1, 2 and 3 hold. Then*

$$\#(\mathcal{D}^C(\gamma_{\text{inc}}^{-1}\Delta) \cap \mathcal{VS}) \leq C_3 \cdot \#(\mathcal{D}^C(\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-1}\Delta)\setminus\mathcal{VS}), \tag{20}$$

for all $\Delta \leq \min(\Delta_0, \gamma_{\text{inc}}^{-1}\Delta_{\max})$, where

$$C_3 := \frac{\log(1/\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F))}{\log(\gamma_{\text{inc}})}. \tag{21}$$

Proof See Appendix A.6. □

Lemma 8 *Suppose Assumptions 1, 2 and 3 hold. Then*

$$\#(\mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L}^C\setminus\mathcal{VS}) \leq \#(\mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L}) + 1, \tag{22}$$

for all $\Delta > 0$.

Proof See Appendix A.7. □

We are now in a position to bound the total number of well-aligned iterations.

Lemma 9 *Suppose Assumptions 1, 2 and 3 hold, and both $\beta_F \leq c_2$ and $\gamma_{\text{inc}} > \min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-2}$ hold. Then if $\|\nabla f(\boldsymbol{x}_k)\| \geq \epsilon$ for all $k = 0, \ldots, K$, we have*

$$\#(\mathcal{A}) \leq \psi(\epsilon) + \frac{C_4}{1 + C_4}(K + 1), \tag{23}$$

where

$$\psi(\epsilon) := \frac{1}{1 + C_4}\left[(C_1 + 2)\phi(\Delta_{\min}(\epsilon), \epsilon)\right.$$
$$+ \frac{4\phi(\gamma_{\text{inc}}^{-1}\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)\Delta_{\min}(\epsilon), \epsilon)}{1 - 2C_3} \tag{24}$$
$$\left. + C_2 + \frac{2}{1 - 2C_3} + 1\right], \tag{25}$$

$$\Delta_{\min}(\epsilon) := \min\left(\overline{\gamma}_{\text{inc}}^{-1}\Delta_0, \min(\gamma_C, \gamma_F, \gamma_{\text{dec}})\overline{\gamma}_{\text{inc}}^{-1}\Delta^*(\epsilon)\right), \tag{26}$$

$$C_4 := \max\left(C_1, \frac{4C_3}{1 - 2C_3}\right) > 0. \tag{27}$$

In these expressions, the values C_1 and C_2 are defined in Lemma 6, C_3 is defined in Lemma 7, $\phi(\cdot, \epsilon)$ is defined in Lemma 4, and $\epsilon_g(\epsilon)$ and $\Delta^*(\epsilon)$ are defined in Lemmas 3 and 5 respectively.

Proof For ease of notation, we will write $\Delta_{\min}$ in place of $\Delta_{\min}(\epsilon)$. We begin by noting that $\gamma_{\text{inc}} > \min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-2}$ implies that $C_3 \in (0, 1/2)$, which we will use later.

Next, we have

$$\begin{aligned}
\#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min})) &= \#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + \#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min}) \backslash \mathcal{S}), \quad (28)\\
&\leq \phi(\Delta_{\min}, \epsilon) + \#(\mathcal{A} \cap \mathcal{D}(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \backslash \mathcal{S})\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \backslash \mathcal{S}), \quad (29)\\
&\leq \phi(\Delta_{\min}, \epsilon) + C_1\#(\mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + C_2\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \backslash \mathcal{S}), \quad (30)\\
&= \phi(\Delta_{\min}, \epsilon) + C_1\#(\mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + C_2\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \cap \mathcal{L} \backslash \mathcal{S})\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \cap \mathcal{L}^C \backslash \mathcal{S}), \quad (31)\\
&\leq \phi(\Delta_{\min}, \epsilon) + C_1\#(\mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + C_2\\
&\quad + 2\#(\mathcal{A} \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \cap \mathcal{L} \backslash \mathcal{S})\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \cap \mathcal{L} \cap \mathcal{S}) + 1, \quad (32)
\end{aligned}$$

where the first inequality follows from Lemma 4, the second inequality follows from Lemma 6 and $\Delta_{\min} \leq \overline{\gamma}_{\text{inc}}^{-1}\Delta_0$, and the last line follows from Lemma 8 and $\mathcal{VS} \subset \mathcal{S}$. Now we use Lemma 5 with $\Delta_{\min} \leq \max(\gamma_C, \gamma_F, \gamma_{\text{dec}})\overline{\gamma}_{\text{inc}}^{-1}\Delta^*(\epsilon)$ to get

$$\begin{aligned}
\#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min})) &\leq \phi(\Delta_{\min}, \epsilon) + C_1\#(\mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + C_2\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min}) \cap \mathcal{L} \cap \mathcal{S}) + 1, \quad (33)\\
&= \phi(\Delta_{\min}, \epsilon) + C_1\#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{S})\\
&\quad + C_1\#(\mathcal{A}^C \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + C_2\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}^C(\Delta_{\min}) \cap \mathcal{L} \cap \mathcal{S})\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{D}^C(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\overline{\gamma}_{\text{inc}}\Delta_{\min})\\
&\quad \cap \mathcal{L} \cap \mathcal{S}) + 1, \quad (34)\\
&\leq \phi(\Delta_{\min}, \epsilon) + C_1\#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{S})\\
&\quad + C_1\#(\mathcal{A}^C \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + C_2\\
&\quad + \#(\mathcal{A} \cap \mathcal{D}^C(\Delta_{\min})) + \#(\mathcal{A} \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + 1, \quad (35)\\
&\leq (C_1 + 2)\phi(\Delta_{\min}, \epsilon) + C_1\#(\mathcal{A}^C \cap \mathcal{D}(\Delta_{\min}) \cap \mathcal{S}) + C_2
\end{aligned}$$

$$+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min}))+1, \tag{36}$$

where the last line follows from Lemma 4.

Separately, we use Lemma 8, and apply Lemma 5 with $\Delta_{\min}\le\Delta^*(\epsilon)$ to get

$$\begin{aligned}
\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})) &= \#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{VS})+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{L}\backslash\mathcal{VS})\\
&\quad+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{L}^C\backslash\mathcal{VS}), &(37)\\
&\le \#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{VS})+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{L}\backslash\mathcal{VS})\\
&\quad+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{L})+1, &(38)\\
&= \#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{VS})+2\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{L}\backslash\mathcal{VS})\\
&\quad+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{L}\cap\mathcal{VS})+1, &(39)\\
&= \#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{VS})+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{L}\cap\mathcal{VS})+1, &(40)\\
&\le 2\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{VS})+1. &(41)
\end{aligned}$$

We then get

$$\begin{aligned}
\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})) &\le 2\#(\mathcal{A}\cap\mathcal{D}^C(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min})\cap\mathcal{VS})\\
&\quad+2\#(\mathcal{A}\cap\mathcal{D}(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min})\\
&\quad\cap\mathcal{D}^C(\Delta_{\min})\cap\mathcal{VS})+1, &(42)\\
&\le 2\#(\mathcal{D}^C(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min})\cap\mathcal{VS})\\
&\quad+2\#(\mathcal{A}\cap\mathcal{D}(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min})\cap\mathcal{VS})+1, &(43)\\
&\le 2C_3\#(\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS})\\
&\quad+2\phi(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min},\epsilon)+1, &(44)\\
&= 2C_3\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS})+2C_3\#(\mathcal{A}^C\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS})\\
&\quad+2\phi(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min},\epsilon)+1, &(45)\\
&\le 2C_3\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min}))+2C_3\#(\mathcal{A}^C\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS})\\
&\quad+2\phi(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min},\epsilon)+1, &(46)
\end{aligned}$$

where the third inequality follows from Lemmas 4 and 7 with

$$\begin{aligned}
\Delta_{\min} &\le \overline{\gamma}_{\text{inc}}^{-1}\Delta_0 \le \gamma_{\text{inc}}^{-1}\Delta_0 \le \min(\Delta_0,\gamma_{\text{inc}}^{-1}\Delta_{\max})\\
&\le \min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)^{-1}\min(\Delta_0,\gamma_{\text{inc}}^{-1}\Delta_{\max}). &(47)
\end{aligned}$$

Since $C_3\in(0,1/2)$, we can rearrange (46) to conclude that

$$\begin{aligned}&\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min}))\\&\quad\le\frac{1}{1-2C_3}\Big[2C_3\#(\mathcal{A}^C\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS})\\&\qquad+2\phi(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min},\epsilon)+1\Big].\end{aligned}\tag{48}$$

Now, we combine (36) and (48) to get

$$\#(\mathcal{A})=\#(\mathcal{A}\cap\mathcal{D}(\Delta_{\min}))+\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min})),\tag{49}$$

$$\begin{aligned}&\le(C_1+2)\phi(\Delta_{\min},\epsilon)+C_1\#(\mathcal{A}^C\cap\mathcal{D}(\Delta_{\min})\cap\mathcal{S})\\&\quad+C_2+2\#(\mathcal{A}\cap\mathcal{D}^C(\Delta_{\min}))+1,\end{aligned}\tag{50}$$

$$\begin{aligned}&\le(C_1+2)\phi(\Delta_{\min},\epsilon)+\frac{4\phi(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min},\epsilon)}{1-2C_3}\\&\quad+C_1\#(\mathcal{A}^C\cap\mathcal{D}(\Delta_{\min})\cap\mathcal{S})\\&\quad+C_2+\frac{2}{1-2C_3}+1+\frac{4C_3}{1-2C_3}\#(\mathcal{A}^C\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS}),\end{aligned}\tag{51}$$

$$\begin{aligned}&\le(C_1+2)\phi(\Delta_{\min},\epsilon)+\frac{4\phi(\gamma_{\text{inc}}^{-1}\min(\gamma_C,\gamma_F,\gamma_{\text{dec}},\beta_F)\Delta_{\min},\epsilon)}{1-2C_3}\\&\quad+C_2+\frac{2}{1-2C_3}+1\\&\quad+\max\left(C_1,\frac{4C_3}{1-2C_3}\right)\Big[\#(\mathcal{A}^C\cap\mathcal{D}(\Delta_{\min})\cap\mathcal{S})+\#(\mathcal{A}^C\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS})\Big].\end{aligned}\tag{52}$$

Since $\mathcal{A}^C\cap\mathcal{D}(\Delta_{\min})\cap\mathcal{S}$ and $\mathcal{A}^C\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS}$ are disjoint subsets of $\mathcal{A}^C$, we have

$$\begin{aligned}&\#(\mathcal{A}^C\cap\mathcal{D}(\Delta_{\min})\cap\mathcal{S})+\#(\mathcal{A}^C\cap\mathcal{D}^C(\Delta_{\min})\backslash\mathcal{VS})\le\#(\mathcal{A}^C)\\&\quad=(K+1)-\#(\mathcal{A}).\end{aligned}\tag{53}$$

Substituting this into (52) and rearranging, we get the desired result. That $C_4>0$ follows from $C_1>0$ and $C_3\in(0,1/2)$. □

2.4 Overall complexity bound

The key remaining step is to compare $\#(\mathcal{A})$ with K. Since each event "Q_k is well aligned" is effectively an independent Bernoulli trial with success probability at least $1-\delta_S$, we derive the below result based on a concentration bound for Bernoulli trials [25, Lemma 2.1].

Lemma 10 *Suppose Assumptions 1, 2, 3 and 4 hold. Then we have*

$$\mathbb{P}\left[\#(\mathcal{A})+1\le(1-\delta_S)(1-\delta)(K+1)\right]\le e^{-\delta^2(1-\delta_S)K/4},\tag{54}$$

for all $\delta \in (0,1)$.

Proof The `CHECK_MODEL=FALSE` case of this proof has a general framework based on [46, Lemma 4.5]—also followed in [17, 72]—with a probabilistic argument from [25, Lemma 2.1].

First, we consider only the subsequence of iterations $\mathcal{K}_1 := \{k_0, \ldots, k_J\} \subset \{0, \ldots, K\}$ when Q_k is resampled (i.e. where `CHECK_MODEL=FALSE`, so $Q_k \neq Q_{k-1}$). For convenience, we define $\mathcal{A}_1 := \mathcal{A} \cap \mathcal{K}_1$ and $\mathcal{A}_1^C := \mathcal{A}^C \cap \mathcal{K}_1$.

Let T_{k_j} be the indicator function for the event "Q_{k_j} is well-aligned", and so $\#(\mathcal{A}_1) = \sum_{j=0}^J T_{k_j}$. Since $T_{k_j} \in \{0,1\}$, and denoting $p_{k_j} := \mathbb{P}\left[T_{k_j} = 1 | \boldsymbol{x}_{k_j}\right]$, for any $t > 0$ we have

$$\begin{aligned} \mathbb{E}\left[e^{-t(T_{k_j} - p_{k_j})} | \boldsymbol{x}_{k_j}\right] &= p_{k_j} e^{-t(1-p_{k_j})} + (1 - p_{k_j}) e^{t p_{k_j}} \\ &= e^{t p_{k_j} + \log(1 - p_{k_j} + p_{k_j} e^{-t})} \leq e^{t^2 p_{k_j}/2}, \end{aligned} \tag{55}$$

where the inequality from the identity $px + \log(1 - p + pe^{-x}) \leq px^2/2$, for all $p \in [0,1]$ and $x \geq 0$, shown in [25, Lemma 2.1].

Using the tower property of conditional expectations and the fact that, since $k_j \in \mathcal{K}_1$, T_{k_j} only depends on $\boldsymbol{x}_{k_j}$ and not any previous iteration, we then get

$$\mathbb{E}\left[e^{-t(\#(\mathcal{A}_1) - \sum_{j=0}^J p_{k_j})}\right]$$

$$= \mathbb{E}\left[e^{-t\sum_{j=0}^J (T_{k_j} - p_{k_j})}\right], \tag{56}$$

$$= \mathbb{E}\left[\mathbb{E}\left[e^{-t\sum_{j=0}^J (T_{k_j} - p_{k_j})} | Q_0, \ldots, Q_{k_J - 1}, \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{k_J}\right]\right], \tag{57}$$

$$= \mathbb{E}\left[e^{-t\sum_{j=0}^{J-1} (T_{k_j} - p_{k_j})} \mathbb{E}\left[e^{-t(T_{k_J} - p_{k_J})} | Q_0, \ldots, Q_{k_J - 1}, \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{k_J}\right]\right], \tag{58}$$

$$= \mathbb{E}\left[e^{-t\sum_{j=0}^{J-1} (T_{k_j} - p_{k_j})} \mathbb{E}\left[e^{-t(T_{k_J} - p_{k_J})} | \boldsymbol{x}_{k_J}\right]\right], \tag{59}$$

$$\leq e^{t^2 p_{k_J}/2} \mathbb{E}\left[e^{-t\sum_{j=0}^{J-1} (T_{k_j} - p_{k_j})}\right], \tag{60}$$

$$\leq e^{t^2 (\sum_{j=0}^J p_{k_j})/2}, \tag{61}$$

where the second-last line follows from (55) and the last line follows by induction. This means that

$$\mathbb{P}\left[\#(\mathcal{A}_1) \leq \sum_{j=0}^J p_{k_j} - \lambda\right] = \mathbb{P}\left[e^{-t\left(\#(\mathcal{A}_1) - \sum_{j=0}^J p_{k_j}\right)} > e^{t\lambda}\right], \tag{62}$$

$$\leq e^{-t\lambda} \mathbb{E}\left[e^{-t\left(\#(\mathcal{A}_1) - \sum_{j=0}^J p_{k_j}\right)}\right], \tag{63}$$

$$\leq e^{t^2\left(\sum_{j=0}^J p_{k_j}\right)/2 - t\lambda}, \tag{64}$$

where the inequalities follow from Markov's inequality and (61) respectively. Taking $t = \lambda / \sum_{j=0}^J p_{k_j}$, we get

$$\mathbb{P}\left[\#(\mathcal{A}_1) \leq \sum_{j=0}^J p_{k_j} - \lambda\right] \leq e^{-\lambda^2/\left(2\sum_{j=0}^J p_{k_j}\right)}. \tag{65}$$

Finally, we take $\lambda = \delta \sum_{j=0}^J p_{k_j}$ for some $\delta \in (0, 1)$ and note that $p_{k_j} \geq (1 - \delta_S)$ (from Assumption 4), to conclude

$$\mathbb{P}\left[\#(\mathcal{A}_1) \leq (1-\delta)(1-\delta_S)(J+1)\right] \leq \mathbb{P}\left[\#(\mathcal{A}_1) \leq (1-\delta)\sum_{j=0}^J p_{k_j}\right]$$
$$\leq e^{-\delta^2\left(\sum_{j=0}^J p_{k_j}\right)/2}, \tag{66}$$

or equivalently, using the partition $\mathcal{K}_1 = \mathcal{A}_1 \cup \mathcal{A}_1^C$,

$$\mathbb{P}\left[\#(\mathcal{A}_1) \leq (1-\delta)(1-\delta_S)[\#(\mathcal{A}_1) + \#(\mathcal{A}_1^C)]\right] \leq e^{-\delta^2(1-\delta_S)[\#(\mathcal{A}_1)+\#(\mathcal{A}_1^C)]/2}. \tag{67}$$

Now we must consider the iterations for which CHECK_MODEL=TRUE (so $Q_k = Q_{k-1}$), which we denote $\mathcal{K}_1^C$. The algorithm ensures that if $k \in \mathcal{K}_1^C$, then $k + 1 \in \mathcal{K}_1$ (unless we are in the last iteration we consider, $k = K$). Futher, the algorithm guarantees that if $k \in \mathcal{K}_1^C$, then $k > 0$ and $k \in \mathcal{A}$ if and only if $k - 1 \in \mathcal{A}$. These are the key implications of RSDFO that we will now use.

Firstly, we have $\#(\mathcal{K}_1^C) \leq \#(\mathcal{K}_1) + 1$, and so

$$K + 1 = \#(\mathcal{K}_1) + \#(\mathcal{K}_1^C) \leq 2[\#(\mathcal{A}_1) + \#(\mathcal{A}_1^C)] + 1, \tag{68}$$

which means (67) becomes

$$\mathbb{P}\left[\#(\mathcal{A}_1) \leq (1-\delta)(1-\delta_S)[\#(\mathcal{A}_1) + \#(\mathcal{A}_1^C)]\right] \leq e^{-\delta^2(1-\delta_S)K/4}. \tag{69}$$

Setting $\alpha := \delta + \delta_S + \delta\delta_S$, we have $(1-\delta)(1-\delta_S) = 1 - \alpha$, and so

$$\mathbb{P}\left[\#(\mathcal{A}_1) \leq \frac{1-\alpha}{\alpha}\#(\mathcal{A}_1^C)\right] = \mathbb{P}\left[\#(\mathcal{A}_1) \leq (1-\alpha)[\#(\mathcal{A}_1) + \#(\mathcal{A}_1^C)]\right]$$
$$\leq e^{-\delta^2(1-\delta_S)K/4}. \tag{70}$$

Secondly, we have $\#(\mathcal{K}_1^C \cap \mathcal{A}^C) \le \#(\mathcal{A}_1^C) + 1$, and so $\#(\mathcal{A}^C) \le 2\#(\mathcal{A}_1^C) + 1$. This and $\mathcal{A}_1 \subset \mathcal{A}$ give

$$\mathbb{P}\left[\#(\mathcal{A}) \le \frac{1-\alpha}{2\alpha}[\#(\mathcal{A}^C) - 1]\right] \le e^{-\delta^2(1-\delta_S)K/4}. \tag{71}$$

We then note that $K + 1 = \#(\mathcal{A}) + \#(\mathcal{A}^C)$, and so

$$\mathbb{P}\left[\#(\mathcal{A}) \le \frac{1-\alpha}{2\alpha}[K + 1 - \#(\mathcal{A}) - 1]\right] \le e^{-\delta^2(1-\delta_S)K/4}, \tag{72}$$

$$\mathbb{P}\left[\#(\mathcal{A}) + \frac{1-\alpha}{1+\alpha} \le \frac{1-\alpha}{1+\alpha}(K+1)\right] \le e^{-\delta^2(1-\delta_S)K/4}, \tag{73}$$

$$\mathbb{P}\left[\#(\mathcal{A}) + 1 \le (1-\alpha)(K+1)\right] \le e^{-\delta^2(1-\delta_S)K/4}, \tag{74}$$

since $\alpha > 0$. □

Theorem 1 *Suppose Assumptions 1, 2, 3 and 4 hold, and we have $\beta_F \le c_2$, $\delta_S < 1/(1+C_4)$ for C_4 defined in Lemma 9, and $\gamma_{\text{inc}} > \min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-2}$. Then for any $\epsilon > 0$ and*

$$k \ge \frac{2(\psi(\epsilon)+1)}{1-\delta_S - C_4/(1+C_4)}, \tag{75}$$

we have

$$\mathbb{P}\left[\min_{j\le k} \|\nabla f(\boldsymbol{x}_j)\| \le \epsilon\right] \ge 1 - \exp\left(-k\frac{(1-\delta_S - C_4/(1+C_4))^2}{16(1-\delta_S)}\right). \tag{76}$$

Alternatively, if $K_\epsilon := \min\{k : \|\nabla f(\boldsymbol{x}_k)\| \le \epsilon\}$ for any $\epsilon > 0$, then

$$\mathbb{P}\left[K_\epsilon \le \left\lceil \frac{2(\psi(\epsilon)+1)}{1-\delta_S - C_4/(1+C_4)} \right\rceil\right] \ge 1 - \exp\left(-\frac{(\psi(\epsilon)+1)[1-\delta_S - C_4/(1+C_4)]}{8(1-\delta_S)}\right), \tag{77}$$

where $\psi(\epsilon)$ is defined in Lemma 9.

Proof First, fix some arbitrary $k \ge 0$. Let $\epsilon_k := \min_{j\le k} \|\nabla f(\boldsymbol{x}_j)\|$ and A_k be the number of well-aligned iterations in $\{0, \dots, k\}$. If $\epsilon_k > 0$, from Lemma 9, we have

$$A_k \le \psi(\epsilon_k) + \frac{C_4}{1+C_4}(k+1). \tag{78}$$

For any $\delta > 0$ such that

$$\delta < 1 - \frac{C_4}{(1+C_4)(1-\delta_S)}, \tag{79}$$

we have $(1-\delta_S)(1-\delta) > C_4/(1+C_4)$, and so we can compute

$$\begin{aligned}
\mathbb{P}\left[\psi(\epsilon_k) \leq \left[(1-\delta_S)(1-\delta) - \frac{C_4}{1+C_4}\right](k+1) - 1\right] & \\
\leq \mathbb{P}\left[A_k \leq (1-\delta_S)(1-\delta)(k+1)\right], & \qquad (80)\\
\leq e^{-\delta^2(1-\delta_S)k/4}, & \qquad (81)
\end{aligned}$$

using Lemma 10. Defining

$$\delta := \frac{1}{2}\left[1 - \frac{C_4}{(1+C_4)(1-\delta_S)}\right], \qquad (82)$$

we have

$$(1-\delta_S)(1-\delta) = \frac{1}{2}\left[1-\delta_S + \frac{C_4}{1+C_4}\right] > \frac{C_4}{1+C_4}, \qquad (83)$$

since $1-\delta_S > C_4/(1+C_4)$ from our assumption on δ_S. Hence we get

$$\mathbb{P}\left[\psi(\epsilon_k) \leq \frac{1}{2}\left(1-\delta_S - \frac{C_4}{1+C_4}\right)(k+1) - 1\right] \leq e^{-k[1-\delta_S-C_4/(1+C_4)]^2/[16(1-\delta_S)]}, \qquad (84)$$

and we note that this result is still holds if $\epsilon_k = 0$, as $\lim_{\epsilon\to 0}\psi(\epsilon) = \infty$.

Now we fix $\epsilon > 0$ and choose k satisfying (75). We use the fact that $\psi(\cdot)$ is non-increasing to get

$$\begin{aligned}
\mathbb{P}\left[\epsilon_k \geq \epsilon\right] \leq \mathbb{P}\left[\psi(\epsilon_k) \leq \psi(\epsilon)\right], & \qquad (85)\\
\leq \mathbb{P}\left[\psi(\epsilon_k) \leq \frac{1}{2}(1-\delta_S - C_4/(1+C_4))k - 1\right], & \qquad (86)\\
\leq \mathbb{P}\left[\psi(\epsilon_k) \leq \frac{1}{2}(1-\delta_S - C_4/(1+C_4))(k+1) - 1\right], & \qquad (87)
\end{aligned}$$

and (76) follows. Lastly, we fix

$$k = \left\lceil \frac{2(\psi(\epsilon)+1)}{1-\delta_S - C_4/(1+C_4)} \right\rceil, \qquad (88)$$

and we use (76) and the definition of K_ϵ to get

$$\begin{aligned}
\mathbb{P}\left[K_\epsilon \geq k\right] &= \mathbb{P}\left[\epsilon_k \geq \epsilon\right], & (89)\\
&\leq e^{-k[1-\delta_S-C_4/(1+C_4)]^2/[16(1-\delta_S)]}, & (90)\\
&\leq \exp\left(-\frac{(\psi(\epsilon)+1)[1-\delta_S - C_4/(1+C_4)]}{8(1-\delta_S)}\right), & (91)
\end{aligned}$$

and we get (77). □

Corollary 1 *Suppose the assumptions of Theorem 1 hold. Then for $k \geq k_0$ for some k_0, we have*

$$\mathbb{P}\left[\min_{j\leq k} \|\nabla f(\boldsymbol{x}_j)\| \leq \frac{C\kappa_H^{1/2}\kappa_d}{\alpha_Q\sqrt{k}}\right] \geq 1 - e^{-ck}, \tag{92}$$

for some constants $c, C > 0$, and where $\kappa_d := \max(\kappa_{\text{ef}}, \kappa_{\text{eg}})$. Alternatively, for $\epsilon \in (0, \epsilon_0)$ for some ϵ_0, we have

$$\mathbb{P}\left[K_\epsilon \leq \widetilde{C}\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2}\right] \geq 1 - e^{-\widetilde{c}\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2}}, \tag{93}$$

for constants $\widetilde{c}, \widetilde{C} > 0$.

Proof For ϵ sufficiently small, $\epsilon_g(\epsilon)$ and $\Delta_{\min}(\epsilon)$ are equal to a multiple of $\alpha_Q\epsilon/\kappa_d$ and $\alpha_Q\epsilon/(\kappa_H\kappa_d)$ respectively, and so $\psi(\epsilon) = \alpha_1\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2} + \alpha_2 = \Theta(\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2})$, for some constants $\alpha_1, \alpha_2 > 0$.

Therefore for k sufficiently large, the choice

$$\epsilon = \sqrt{\frac{2\alpha_1\kappa_H\kappa_d^2\alpha_Q^{-2}}{(1-\delta_S - C_4/(1+C_4))k - 2 - 2\alpha_2}} = \Theta(\kappa_H^{1/2}\kappa_d\alpha_Q^{-1}k^{-1/2}), \tag{94}$$

is sufficiently small that $\psi(\epsilon) = \alpha_1\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2} + \alpha_2$, and gives (75) with equality. The first result then follows from (76).

The second result follows immediately from $\psi(\epsilon) = \Theta(\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2})$ and (77). □

Remark 2 All the above analysis holds with minimal modifications if we replace the trust-region mechanisms in RSDFO with more standard trust-region updating mechanisms. This includes, for example, having no safety step (i.e. $\beta_F = 0$), and replacing (8) with

$$\boldsymbol{x}_{k+1} = \begin{cases} \boldsymbol{x}_k + \boldsymbol{s}_k, & \rho_k \geq \eta, \\ \boldsymbol{x}_k, & \rho_k < \eta, \end{cases} \quad \text{and} \quad \Delta_{k+1} = \begin{cases} \min(\gamma_{\text{inc}}\Delta_k, \Delta_{\max}), & \rho_k \geq \eta, \\ \gamma_{\text{dec}}\Delta_k, & \rho_k < \eta, \end{cases} \tag{95}$$

for some $\eta \in (0, 1)$. The corresponding requirement on the trust-region updating parameters to prove a version of Theorem 1 is simply $\gamma_{\text{inc}} > \gamma_{\text{dec}}^{-2}$ (provided we also set $\gamma_C = \gamma_{\text{dec}}$).

2.5 Remarks on complexity bound

Our final complexity bounds for RSDFO in Corollary 1 are comparable to probabilistic direct search [46, Corollary 4.9]. They also match—in their dependencies on ϵ, κ_H

and κ_d—the standard bounds for (full space) model-based DFO methods for general objective [37, 76] and nonlinear least-squares [21] problems.

Following [46], we may also derive complexity bounds on the expected first-order optimality measure (of $\mathcal{O}(k^{-1/2})$) and the expected worst-case complexity (of $\mathcal{O}(\epsilon^{-2})$ iterations) for RSDFO.

Theorem 2 *Suppose the assumptions of Theorem 1 hold. Then for $k \geq k_0$, the iterates of RSDFO satisfy*

$$\mathbb{E}\left[\min_{j\leq k} \|\nabla f(\boldsymbol{x}_j)\|\right] \leq C\kappa_H^{1/2}\kappa_d\alpha_Q^{-1}k^{-1/2} + \|\nabla f(\boldsymbol{x}_0)\|e^{-ck}, \tag{96}$$

for $c, C > 0$ and κ_d from (92), and for $\epsilon \in (0, \epsilon_0)$ we have

$$\mathbb{E}[K_\epsilon] \leq \widetilde{C}_1\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2} + \frac{1}{\widetilde{c}_1}, \tag{97}$$

for constants $\widetilde{c}_1, \widetilde{C}_1 > 0$. Here, k_0 and ϵ_0 are the same as in Corollary 1.

Proof First, for $k \geq k_0$ define the random variable H_k as

$$H_k := \begin{cases} C\kappa_H^{1/2}\kappa_d\alpha_Q^{-1}k^{-1/2}, & \text{if } \min_{j\leq k}\|\nabla f(\boldsymbol{x}_j)\| \leq C\kappa_H^{1/2}\kappa_d\alpha_Q^{-1}k^{-1/2}, \\ \|\nabla f(\boldsymbol{x}_0)\| & \text{otherwise.} \end{cases} \tag{98}$$

Then since $\min_{j\leq k}\|\nabla f(\boldsymbol{x}_j)\| \leq H_k$, we get

$$\begin{aligned} \mathbb{E}\left[\min_{j\leq k}\|\nabla f(\boldsymbol{x}_j)\|\right] &\leq \mathbb{E}[H_k] \\ &\leq C\kappa_H^{1/2}\kappa_d\alpha_Q^{-1}k^{-1/2} \\ &\quad + \|\nabla f(\boldsymbol{x}_0)\|\,\mathbb{P}\left[\min_{j\leq k}\|\nabla f(\boldsymbol{x}_j)\| > C\kappa_H^{1/2}\kappa_d\alpha_Q^{-1}k^{-1/2}\right], \end{aligned} \tag{99}$$

and we get the first result by applying Corollary 1.

Next, if $\epsilon \in (0, \epsilon_0)$ then

$$k \geq k_0(\epsilon) := \frac{2(\psi(\epsilon)+1)}{1-\delta_S - C_4/(1+C_4)} = \Theta(\kappa_H\kappa_d^2\alpha_Q^{-2}\epsilon^{-2}), \tag{100}$$

and so from Theorem 1 we have

$$\mathbb{P}[K_\epsilon \leq k] = \mathbb{P}\left[\min_{j\leq k}\|\nabla f(\boldsymbol{x}_j)\| \leq \epsilon\right] \geq 1 - e^{-\widetilde{c}_1 k}, \tag{101}$$

where $\widetilde{c}_1 := (1-\delta_S - C_4/(1+C_4))^2/[16(1-\delta_S)]$. We use the identity $\mathbb{E}[X] = \int_0^\infty \mathbb{P}[X>t]\,dt$ for non-negative random variables X (e.g. [73, eqn. (1.9)]) to get

$$\mathbb{E}[K_\epsilon] \le k_0(\epsilon) + \int_{k_0(\epsilon)}^\infty \mathbb{P}[K_\epsilon > t]\,dt \le k_0(\epsilon) + \sum_{k=k_0(\epsilon)}^\infty e^{-\widetilde{c}_1 k} = k_0(\epsilon) + \frac{e^{-\widetilde{c}_1 k_0(\epsilon)}}{1-e^{-\widetilde{c}_1}}, \tag{102}$$

where $\widetilde{C}_1$ comes from $k_0(\epsilon) = \Theta(\kappa_H \kappa_d^2 \alpha_Q^{-2} \epsilon^{-2})$, which concludes our proof. □

Furthermore, we also get almost-sure convergence of lim inf type, similar to [29, Theorem 5.8] or [30, Theorem 10.12] in the deterministic case.

Theorem 3 *Suppose the assumptions of Theorem 1 hold. Then the iterates of RSDFO satisfy* $\inf_{k\ge 0} \|\nabla f(\boldsymbol{x}_k)\| = 0$ *almost surely.*

Proof From Theorem 1, for any $\epsilon > 0$ we have

$$\lim_{k\to\infty} \mathbb{P}\left[\min_{j\le k} \|\nabla f(\boldsymbol{x}_j)\| > \epsilon\right] = 0. \tag{103}$$

However, $\mathbb{P}\left[\inf_{k\ge 0} \|\nabla f(\boldsymbol{x}_k)\| > \epsilon\right] \le \mathbb{P}\left[\min_{j\le k} \|\nabla f(\boldsymbol{x}_j)\| > \epsilon\right]$ for all k, and so

$$\mathbb{P}\left[\inf_{k\ge 0} \|\nabla f(\boldsymbol{x}_k)\| > \epsilon\right] = 0. \tag{104}$$

The result follows from the union bound applied to any sequence $\epsilon \to 0$, e.g. $\epsilon_k = k^{-1}$. □

In particular, if $\|\nabla f(\boldsymbol{x}_k)\| > 0$ for all k, then Theorem 3 implies $\liminf_{k\to\infty} |\nabla f(\boldsymbol{x}_k)\| = 0$ almost surely.

2.6 Selecting a subspace dimension

We now specify how to generate our subspaces Q_k to be probabilistically well-aligned and uniformly bounded (Assumption 4). These requirements are quite weak, and so there are several possible approaches for constructing Q_k. Of course the simplest case is to use no embedding, taking $Q_k = I_{n\times n}$, which gives us $p = n$ and $Q_{\max} = 1$ in Assumption 4, however our overall complexity can be reduced with alternative approaches.

One approach to achieve this is by using Johnson-Lindenstrauss transforms (JLTs) [80]. The application of these techniques to random subspace optimization algorithms follows [16, 17, 72].

Definition 3 A random matrix $S \in \mathbb{R}^{p \times n}$ is an (β, δ)-JLT if, for any point $\boldsymbol{v} \in \mathbb{R}^n$, we have

$$\mathbb{P}\left[(1-\beta)\|\boldsymbol{v}\|^2 \leq \|S\boldsymbol{v}\|^2 \leq (1+\beta)\|\boldsymbol{v}\|^2\right] \geq 1-\delta. \tag{105}$$

There have been many different approaches for constructing (β, δ)-JLT matrices proposed. Two common examples are:

- If S is a random Gaussian matrix with independent entries $S_{i,j} \sim N(0, 1/p)$ and $p = \Omega(\beta^{-2}|\log \delta|)$, then S is an (β, δ)-JLT (see [13, Theorem 2.13], for example).
- We say that S is an s-hashing matrix if it has exactly s nonzero entries per column (indices sampled independently), which take values $\pm 1/\sqrt{s}$ selected independently with probability 1/2. If S is an s-hashing matrix with $s = \Theta(\beta^{-1}|\log \delta|)$ and $p = \Omega(\beta^{-2}|\log \delta|)$, then S is an (β, δ)-JLT [52].

By taking $\boldsymbol{v} = \nabla f(\boldsymbol{x}_k)$ in iteration k, and noting $(1-\beta)^2 \leq 1-\beta$ for all $\beta \in (0, 1)$, we have that Assumption 4(a) holds if we take $Q_k = S^T$, where S is any $(1-\alpha_Q, \delta_S)$-JLT. That is, Assumption 4(a) is satisfied using either of the constructions above and $p = \Omega((1-\alpha_Q)^{-2}|\log \delta_S|)$. We note that we need $\alpha_Q < 1$ to use this construction.

Alternatively, following [54], we may take $Q_k = \sqrt{n/p}\, Z_{:,1:p}$, where $Z_{:,1:p}$ comprises the first p columns of $Z \in \mathbb{R}^{n \times n}$ sampled from the Haar distribution (i.e. a uniform distribution over $n \times n$ orthogonal matrices). In this construction, the columns of Q_k are orthogonal. From [54, Lemma 1], we have that Q_k satisfies Assumption 4(a) for any p and α_Q with failure probability

$$\delta_S = I_{\alpha_Q^2 p/n}(p/2, (n-p)/2), \tag{106}$$

where $I_q(\alpha, \beta)$ is the regularized incomplete beta function. Although this does not give us a simple criterion for choosing p in terms of α_Q and δ_S, [54, Figure 1] gives numerical evidence that p can be chosen independently of n.[7] We note that [77] considered a similar construction based on the Grassmann manifold.

Value of $Q_{\max}$ If S is chosen to be Gaussian, then [6, Corollary 3.11] gives the upper bound $Q_{\max} = \mathcal{O}(\sqrt{n/p})$ with high probability. Following a union bound argument, by generating Gaussian S and rejecting those with large norm, we can achieve Assumption 4 for this construction while maintaining $p = \mathcal{O}(1)$. If S is a hashing matrix, then we have $\|Q_k\| \leq \|Q_k\|_F = \sqrt{n}$, and so $Q_{\max} = \sqrt{n}$ suffices to achieve Assumption 4. Lastly, if Q_k is a subsampled Haar matrix, we simply get $Q_{\max} = \sqrt{n/p}$.

Thus, we have presented three different random ensembles from which Q_k may be generated, each allowing us to use subspace dimension $p = \mathcal{O}(1)$, but requiring $Q_{\max} = \mathcal{O}(\sqrt{n})$. We note that the RSDFO framework and complexity analysis allow for different ensembles and/or bounds on p or $Q_{\max}$, including any with improved dependencies on n, if possible.

[7] For example, with $\alpha_Q = \sqrt{0.8} \approx 0.89$ and $\delta_S = 0.2$, we may choose subspace dimension $p = 40$ for all ambient dimensions $n \leq 10^8$.

Remark 3 We conclude this section by noting that our analysis raises the question of whether scaling Q by a (small) constant factor would improve the performance and complexity of the algorithm (by decreasing both α_Q and $Q_{\max}$). This, and more broadly how to optimally design an embedding, is a diffcult and important question that we dedicate to future work.

2.7 Complexity for general linear interpolation

In the case of linear interpolation models for a general objective problem (for which RSDFO may be applied), reasoning similar to Lemma 11 and using the standard fully linear error bounds from [28] or [30, Theorems 2.11, 2.12, 3.14] gives $\kappa_{\text{ef}}, \kappa_{\text{eg}} = \mathcal{O}(Q_{\max}^2 p \Lambda)$. Since we may take $\kappa_H = 1$ for linear models and noting that these methods still require at most $p+1$ evaluations per iteration, this yields a high probability complexity of $\mathcal{O}(Q_{\max}^4 p^2 \epsilon^{-2})$ iterations or $\mathcal{O}(Q_{\max}^4 p^3 \epsilon^{-2})$ evaluations.

This means that RSDFO with a full-space model (i.e. $p = n$ and $Q_k = I$) requires $\mathcal{O}(n^2\epsilon^{-2})$ iterations and $\mathcal{O}(n^3\epsilon^{-2})$ evaluations. However, with careful subspace generation using the methods in Sect. 2.6, with $p = \mathcal{O}(1)$ and $Q_{\max} = \mathcal{O}(\sqrt{n})$, we again get $\mathcal{O}(n^2\epsilon^{-2})$ iterations but a strict improvement to only $\mathcal{O}(n^2\epsilon^{-2})$ evaluations. Our linear algebra cost also reduces from $\mathcal{O}(n^3)$ to $\mathcal{O}(n)$ flops per iteration, with a corresponding reduction in the overall linear algebra cost over the whole algorithm from $\mathcal{O}(n^5\epsilon^{-2})$ to $\mathcal{O}(n^3\epsilon^{-2})$ flops. A detailed summary of the linear algebra cost of RSDFO is given for the nonlinear least-squares case in Sect. 3.3; similar results apply here.

Instead of linear models, we may instead use (possibly underdetermined) quadratic interpolation to construct fully linear models. Details of these procedures may be found in [28, 30].

3 Random subspace nonlinear least-squares method

We now describe how RSDFO (Algorithm 1) can be specialized to the unconstrained nonlinear least-squares problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) := \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{x})\|^2 = \frac{1}{2}\sum_{i=1}^{m} r_i(\boldsymbol{x})^2, \tag{107}$$

where $\boldsymbol{r} : \mathbb{R}^n \to \mathbb{R}^m$ is given by $\boldsymbol{r}(\boldsymbol{x}) := [r_1(\boldsymbol{x}), \ldots, r_m(\boldsymbol{x})]^T$. We assume that $\boldsymbol{r}$ is differentiable, but that access to the Jacobian $J : \mathbb{R}^n \to \mathbb{R}^{m\times n}$ is not possible. In addition, we typically assume that $m \geq n$ (regression), but everything here also applies to the case $m < n$ (inverse problems). We now introduce the algorithm RSDFO-GN (Random Subspace DFO with Gauss–Newton), which is a random subspace version of a model-based DFO variant of the Gauss–Newton method [19].

Following the construction from [19], we assume that we have selected the p-dimensional search space $\mathcal{Y}_k$ defined by $Q_k \in \mathbb{R}^{n\times p}$ (as in RSDFO above). Then, we suppose that we have evaluated $\boldsymbol{r}$ at $p+1$ points $Y_k := \{\boldsymbol{x}_k, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_p\} \subset \mathcal{Y}_k$ (which typically are all close to $\boldsymbol{x}_k$ and not recycled from previous iterations). Since $\boldsymbol{y}_t \in \mathcal{Y}_k$ for each $t = 1, \dots, p$, from (3) we have $\boldsymbol{y}_t = \boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}}_t$ for some $\hat{\boldsymbol{s}}_t \in \mathbb{R}^p$.

Given this interpolation set, we first wish to construct a local subspace linear model for $\boldsymbol{r}$:

$$\boldsymbol{r}(\boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}}) \approx \hat{\boldsymbol{m}}_k(\hat{\boldsymbol{s}}) = \boldsymbol{r}(\boldsymbol{x}_k) + \hat{J}_k\hat{\boldsymbol{s}}. \tag{108}$$

To do this, we choose the approximate subspace Jacobian $\hat{J}_k \in \mathbb{R}^{m\times p}$ by requiring that $\hat{\boldsymbol{m}}_k$ interpolate $\boldsymbol{r}$ at our interpolation points Y_k. That is, we impose

$$\hat{\boldsymbol{m}}_k(\hat{\boldsymbol{s}}_t) = \boldsymbol{r}(\boldsymbol{y}_t), \qquad \forall t = 1, \dots, p, \tag{109}$$

which yields the $p \times p$ linear system (with m right-hand sides)

$$\hat{W}_k\hat{J}_k^T := \begin{bmatrix} \hat{\boldsymbol{s}}_1^T \\ \vdots \\ \hat{\boldsymbol{s}}_p^T \end{bmatrix} \hat{J}_k^T = \begin{bmatrix} (\boldsymbol{r}(\boldsymbol{y}_1) - \boldsymbol{r}(\boldsymbol{x}_k))^T \\ \vdots \\ (\boldsymbol{r}(\boldsymbol{y}_p) - \boldsymbol{r}(\boldsymbol{x}_k))^T \end{bmatrix}. \tag{110}$$

Our linear subspace model $\hat{\boldsymbol{m}}_k$ (108) naturally yields a local subspace quadratic model for f, as in the classical Gauss–Newton method, namely (c.f. (4)),

$$f(\boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}}) \approx \hat{m}_k(\hat{\boldsymbol{s}}) := \frac{1}{2}\|\hat{\boldsymbol{m}}_k(\hat{\boldsymbol{s}})\|^2 = f(\boldsymbol{x}_k) + \hat{\boldsymbol{g}}_k^T\hat{\boldsymbol{s}} + \frac{1}{2}\hat{\boldsymbol{s}}^T\hat{H}_k\hat{\boldsymbol{s}}, \tag{111}$$

where $\hat{\boldsymbol{g}}_k := \hat{J}_k^T\boldsymbol{r}(\boldsymbol{x}_k)$ and $\hat{H}_k := \hat{J}_k^T\hat{J}_k$.

3.1 Constructing Q_k-fully linear models

We now describe how we can achieve Q_k-fully linear models of the form (111) in RSDFO-GN.

As in [19], we will need to define the Lagrange polynomials and Λ-poisedness of an interpolation set. Given our interpolation set Y_k lies inside $\mathcal{Y}_k$, we consider the (low-dimensional) Lagrange polynomials associated with Y_k. These are the linear functions $\hat{\ell}_0, \dots, \hat{\ell}_p : \mathbb{R}^p \to \mathbb{R}$, defined by the interpolation conditions

$$\hat{\ell}_t(\hat{\boldsymbol{s}}_{t'}) = \delta_{t,t'}, \qquad \forall t, t' = 0, \dots, p, \tag{112}$$

with the convention $\hat{\boldsymbol{s}}_0 = \boldsymbol{0}$ corresponding to the interpolation point $\boldsymbol{x}_k$. The Lagrange polynomials exist and are unique whenever $\hat{W}_k$ (110) is invertible, which we typically ensure through judicious updating of Y_k at each iteration.

Definition 4 For any $\Lambda > 0$, the set Y_k is Λ-poised in the p-dimensional ball $B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$ if

$$\max_{t=0,\dots,p} \max_{\|\hat{\boldsymbol{s}}\|\le\Delta_k} |\hat{\ell}_t(\hat{\boldsymbol{s}})| \le \Lambda. \tag{113}$$

Note that since $\hat{\ell}_0(\mathbf{0}) = 1$, for the set Y_k to be Λ-poised we require $\Lambda \geq 1$. In general, a larger Λ indicates that Y_k has "worse" geometry, which leads to a less accurate approximation for f. This notion of Λ-poisedness (in a subspace) is sufficient to construct Q_k-fully linear models (111) for f.

Lemma 11 *Suppose Assumption* 4(b) *holds,* $J(\boldsymbol{x})$ *is Lipschitz continuous, and* $\boldsymbol{r}$ *and* J *are uniformly bounded above in* $\cup_{k\geq 0} B(\boldsymbol{x}_k, \Delta_{\max})$. *If* $Y_k \subset B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$ *and* Y_k *is* Λ*-poised in* $B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$*, then* $\hat{m}_k$ (111) *is a* Q_k*-fully linear model for* f*, with* $\kappa_{\text{ef}}, \kappa_{\text{eg}} = \mathcal{O}(Q_{\max}^4 p^2 \Lambda^2)$.

Proof Consider the low-dimensional functions $\hat{\boldsymbol{r}} : \mathbb{R}^p \to \mathbb{R}^m$ and $\hat{f} : \mathbb{R}^p \to \mathbb{R}$ given by $\hat{\boldsymbol{r}}_k(\hat{\boldsymbol{s}}) := \boldsymbol{r}(\boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}})$ and $\hat{f}(\hat{\boldsymbol{s}}) := \frac{1}{2}\|\hat{\boldsymbol{r}}(\hat{\boldsymbol{s}})\|^2$ respectively. We note that $\hat{\boldsymbol{r}}$ is continuously differentiable with Jacobian $\hat{J}(\hat{\boldsymbol{s}}) = J(\boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}})Q_k$. Then since $\|Q_k\| \leq Q_{\max}$ from Assumption 4(b), it is straightforward to show that both $\hat{\boldsymbol{r}}$ and $\hat{J}$ are uniformly bounded above and $\hat{J}$ is Lipschitz continuous (with a Lipschitz constant $Q_{\max}^2$ times larger than for $J(\boldsymbol{x})$).

We can then consider $\hat{\boldsymbol{m}}_k$ (108) and $\hat{m}_k$ (111) to be interpolation models for $\hat{r}$) and $\hat{f}$ in the low-dimensional ball $B(\mathbf{0}, \Delta_k) \subset \mathbb{R}^p$. From [19, Lemma 3.3], we conclude that $\hat{m}_k$ is a fully linear model for $\hat{f}$ with constants $\kappa_{\text{ef}}, \kappa_{\text{eg}} = \mathcal{O}(p^2\Lambda^2)$. The Q_k-fully linear property follows immediately from this, noting that $\nabla\hat{f}_k(\hat{\boldsymbol{s}}) = Q_k^T \nabla f(\boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}})$. The dependency on $Q_{\max}$ follows as $\kappa_{\text{ef}}, \kappa_{\text{eg}} = \mathcal{O}(L_{\hat{J}}^2)$, where $L_{\hat{J}}$ is the Lipschitz constant of $\hat{J}$ [19, Lemma 3.2]. $\square$

Given this result, the procedures in [28] or [30, Chapter 6] allow us to check and/or guarantee the Λ-poisedness of an interpolation set, and we have met all the requirements needed to fully specify RSDFO-GN.

Lastly, we note that underdetermined linear interpolation, where (110) is underdetermined and solved in a minimal norm sense, has been recently shown to yield a property similar to Q_k-full linearity [51, Theorem 3.6].

Complete RSDFO-GN algorithm A complete statement of RSDFO-GN is given in Algorithm 2. This exactly follows RSDFO (Algorithm 1), but where we ask that the interpolation set satisfies the conditions: $Y_k \subset B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$ and Y_k is Λ-poised in $B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$. From Lemma 11, this is sufficient to guarantee Q_k-full linearity of $\hat{m}_k$.

3.2 Complexity analysis for RSDFO-GN

We are now in a position to specialize our complexity analysis for RSDFO to RSDFO-GN. For this, we need to impose a smoothness assumption on $\boldsymbol{r}$.

Assumption 5 The level set $\mathcal{L} := \{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0)\}$ is bounded, $\boldsymbol{r}$ is continuously differentiable, and the Jacobian J is Lipschitz continuous on $\mathcal{L}$.

This smoothness requirement allows us to immediately apply the complexity analysis for RSDFO, yielding the following result.

Algorithm 2 RSDFO-GN (Random Subspace Derivative-Free Optimization with Gauss–Newton) for solving (107).

Input: Starting point $\boldsymbol{x}_0 \in \mathbb{R}^n$, initial trust region radius $\Delta_0 > 0$, and subspace dimension $p \in \{1, \dots, n\}$.

Parameters: maximum trust-region radius $\Delta_{\max} \geq \Delta_0$, trust-region radius scalings $0 < \gamma_{\text{dec}} < 1 < \gamma_{\text{inc}} \leq \overline{\gamma}_{\text{inc}}$, criticality constants $\epsilon_C, \mu > 0$ and trust-region scaling $0 < \gamma_C < 1$, safety step threshold $\beta_F > 0$ and trust-region scaling $0 < \gamma_F < 1$, acceptance thresholds $0 < \eta_1 \leq \eta_2 < 1$, and poisedness constant $\Lambda > 1$.

```
1: Set flag CHECK_MODEL=FALSE.
2: for $k = 0, 1, 2, \dots$ do
3:     if CHECK_MODEL=TRUE then
4:         Set $Q_k = Q_{k-1}$.
5:         Construct an interpolation set $Y_k \subset B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$ which is $Y_k$ is $\Lambda$-poised in $B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$.
6:         Build the reduced model $\hat{m}_k : \mathbb{R}^p \to \mathbb{R}$ (111) by solving (110).
7:     else
8:         Define a subspace by randomly sampling $Q_k \in \mathbb{R}^{n \times p}$.
9:         Construct a reduced model $\hat{m}_k : \mathbb{R}^p \to \mathbb{R}$ (111) by solving (110), where the interpolation points $Y_k \subset \mathcal{Y}_k$ need not be contained in $B(\boldsymbol{x}_k, \Delta_k)$ or be $\Lambda$-poised in $B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$.
10:    end if
11:    Follow lines 10 to 22 of RSDFO (Algorithm 1), but replace every instance of checking $Q_k$-full linearity of $\hat{m}_k$ in $B(\boldsymbol{x}_k, \Delta_k)$ with checking that $Y_k \subset B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$ and $Y_k$ is $\Lambda$-poised in $B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$.
12: end for
```

Corollary 2 *Suppose Assumptions 5, 2, 3 and 4 hold, and we have $\beta_F \leq c_2$, $\delta_S < 1/(1 + C_4)$ for C_4 defined in Lemma 9, and $\gamma_{\text{inc}} > \min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-2}$. Then for the iterates generated by RSDFO-GN and k sufficiently large,*

$$\mathbb{P}\left[\min_{j \leq k} \|\nabla f(\boldsymbol{x}_j)\| \leq \frac{C\kappa_H^{1/2} Q_{\max}^4 p^2}{\sqrt{k}}\right] \geq 1 - e^{-ck}, \tag{114}$$

for some constants $c, C > 0$. Alternatively, for $\epsilon \in (0, \epsilon_0)$ for some ϵ_0, we have

$$\mathbb{P}\left[K_\epsilon \leq \widetilde{C}\kappa_H Q_{\max}^8 p^4 \epsilon^{-2}\right] \geq 1 - e^{-\widetilde{c}\kappa_H Q_{\max}^8 p^4 \epsilon^{-2}}, \tag{115}$$

for constants $\widetilde{c}, \widetilde{C} > 0$.

Proof Assumption 5 implies that both $\boldsymbol{r}$ and J are uniformly bounded above on $\mathcal{L}$, which is sufficient for Lemma 11 to hold. Hence, whenever we check/ensure that $Y_k \subset B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$ and Y_k is Λ-poised in $B(\boldsymbol{x}_k, \Delta_k) \cap \mathcal{Y}_k$ we are checking/guaranteeing that $\hat{m}_k$ is Q_k-fully linear in $B(\boldsymbol{x}_k, \Delta_k)$. In addition, from [19, Lemma 3.2] and taking $f_{\text{low}} = 0$, we have that Assumption 1 is satisfied. Therefore the result follows directly from Corollary 1 and $\kappa_d = \mathcal{O}(Q_{\max}^4 p^2)$ from Lemma 11. □

We also note that it is reasonable to assume $\kappa_H = \mathcal{O}(\kappa_d)$ [19, Lemma 3.3], and so the overall iteration complexity of RSDFO-GN is $\mathcal{O}(Q_{\max}^{12} p^6 \epsilon^{-2})$ with high probability. Furthermore, each iteration of RSDFO or RSDFO-GN requires at most $p + 1$ objective evaluations (p to form the model $\hat{m}_k$, regardless of whether we need Q_k-full linearity or not, and one for $\boldsymbol{x}_k + \boldsymbol{s}_k$). Hence the evaluation complexity of RSDFO-GN is $\mathcal{O}(Q_{\max}^{12} p^7 \epsilon^{-2})$ with high probability.

If we use a full-space model (so $p = n$ and $Q_k = I$), then with $Q_{\max} = 1$ we get a high probability complexity of $\mathcal{O}(n^6 \epsilon^{-2})$ iterations and $\mathcal{O}(n^7 \epsilon^{-2})$ evaluations. However, if we apply one of the subspace generation techniques from Sect. 2.6, with $p = \mathcal{O}(1)$ and $Q_{\max} = \mathcal{O}(\sqrt{n})$, then we get the same complexity of $\mathcal{O}(n^6 \epsilon^{-2})$ iterations, but an improved evaluation complexity of $\mathcal{O}(n^6 \epsilon^{-2})$. Thus RSDFO-GN with careful subspace generation can give a strict improvement in the evaluation complexity compared to standard full-space methods.

In the next section we show that RSDFO-GN also improves on full-space methods in the linear algebra cost at each iteration, reducing from $\mathcal{O}(mn^2 + n^3)$ flops per iteration to $\mathcal{O}(m + n)$ flops per iteration. Hence in the standard least-squares setting where $m \ge n$, the linear algebra cost of achieving a ϵ first-order optimality reduces from $\mathcal{O}(mn^8 \epsilon^{-2})$ to $\mathcal{O}(mn^6 \epsilon^{-2})$ flops.

3.3 Linear algebra cost of RSDFO-GN

In RSDFO-GN, the interpolation linear system (110) is solved in two steps, namely: factorize the interpolation matrix $\hat{W}_k$, then back-solve for each right-hand side. Thus, the cost of the linear algebra is:

1. Model construction costs $\mathcal{O}(p^3)$ to compute the factorization of $\hat{W}_k$, and $\mathcal{O}(mp^2)$ for the back-substitution solves with m right-hand sides; and
2. Lagrange polynomial construction costs $\mathcal{O}(p^3)$ in total, due to one backsolve for each of the $p + 1$ polynomials (using the pre-existing factorization of $\hat{W}_k$).

By updating the factorization or $\hat{W}_k^{-1}$ directly (e.g. via the Sherman-Morrison formula), we can replace the $\mathcal{O}(p^3)$ factorization cost with a $\mathcal{O}(p^2)$ updating cost (c.f. [66]). However, the dominant $\mathcal{O}(mp^2)$ model construction cost remains, and in practice we have observed that the factorization needs to be recomputed from scratch to avoid the accumulation of rounding errors. We also have a cost, typically of $\mathcal{O}(np)$ or $\mathcal{O}(np^2)$ to construct Q_k using our randomized procedures from Sect. 2.6, and $\mathcal{O}(np)$ from projecting the computed step $\hat{\boldsymbol{s}}_k$ to the full space. Hence in our case where $p = \mathcal{O}(1)$, the linear algebra cost per iteration is $\mathcal{O}(m + n)$.

In the case of a full-space method where $p = n$ such as in [19], these costs becomes $\mathcal{O}(n^3)$ for the factorization (or $\mathcal{O}(n^2)$ if Sherman-Morrison is used) plus $\mathcal{O}(mn^2)$ for the back-solves. When n grows large, this linear algebra cost rapidly dominates the total runtime of these algorithms and limits the efficiency of full-space methods. This issue is discussed in more detail, with numerical results, in [69, Chapter 7.2]. This per-iteration cost is substantially higher than RSDFO-GN with random subspace generation.

In light of this discussion, we now turn our attention to building an implementation of RSDFO-GN that has both strong performance (in terms of objective evaluations) and low linear algebra cost.

4 DFBGN: an efficient implementation of RSDFO-GN

An important tenet of DFO is that objective evaluations are often expensive, and so algorithms should be efficient in reusing information, hence limiting the total objective evaluations required to achieve a given decrease. However because we require our model to sit within our active space $\mathcal{Y}_k$, we do not have a natural process by which to reuse evaluations between iterations, when the space changes. We dedicate this section to outlining an implementation of RSDFO-GN, which we call DFBGN (Derivative-Free Block Gauss–Newton). DFBGN is designed to be efficient in its objective queries while still only building low-dimensional models, and hence is also efficient in terms of linear algebra. Specifically, we design DFBGN to achieve two aims:

- *Low computational cost* we want our implementation to have a per-iteration linear algebra cost which is linear in the ambient dimension;
- *Efficient use of objective evaluations* our implementation should follow the principles of other DFO methods and make progress with few objective evaluations. In particular, we hope that, when run with ‘full-space models’ (i.e. $p = n$), our implementation should have (close to) state-of-the-art performance.

We will assess the second point in Sect. 5 by comparison with DFO-LS [15] an open-source model-based DFO Gauss–Newton solver which explores the full space (i.e. $p = n$).

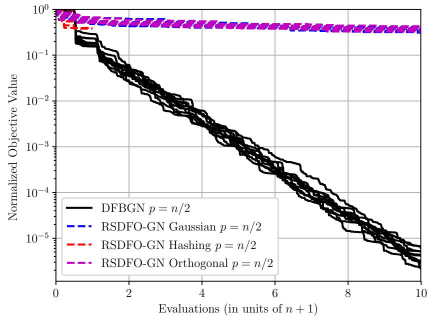
Remark 4 As discussed in [15], DFO-LS has a mechanism to build a model with fewer than $n + 1$ interpolation points. However, in that context we modify the model so that it varies over the whole space $\mathbb{R}^n$, which enables the interpolation set to grow to the usual $n + 1$ points and yield a full-dimensional model. There, the goal is to make progress with very few evaluations, but here our goal is scalability, so we keep our model low-dimensional throughout and instead change the subspace at each iteration.

4.1 Efficiency of DFBGN versus RSDFO-GN

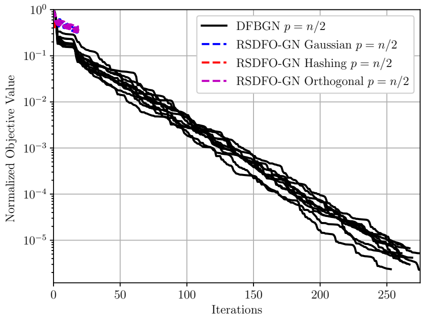
To motivate the utility of DFBGN, we begin by showing a comparison of DFBGN against a direct implementation of RSDFO-GN. We implement RSDFO-GN by constructing Q_k-fully linear interpolation models at each iteration by setting $\hat{\boldsymbol{s}}_t$ in (110) to be the t-th coordinate vector in $\mathbb{R}^p$, and generate Q_k using the different approaches described in Sect. 2.6. To ensure consistency between the two algorithms, we use identical trust-region management procedures, algorithm parameters and starting points for our comparison.

We test DFBGN and RSDFO-GN on several CUTEst problems [42] with dimension $n \approx 100$, drawn from the (CR) collection described in Sect. 5.1, with subspace dimensions $p \in \{n/10, n/4, n/2, n\}$. For brevity, we show results for ARWHDNE and $p = n/2$, but the results are similar for all problems and values of p.[8] All solvers were run for a maximum of $10(n + 1)$ evaluations or until $\Delta_k \leq 10^{-8}$, and because

[8] We tested (CR) collection problems ARGLALE, ARGTRIG, ARWHDNE, BROYDN3D, CHANDHEQ, FREURONE, INTEGREQ, and VARDIMNE.



(a) Normalized objective value vs. evaluations



(b) Normalized objective value vs. iterations

Fig. 1 Normalized objective value (versus evaluations and iterations) for 10 runs of RSDFO-GN DFBGN on CUTEst problem ARWHDNE with $p = n/2$. We also show different methods for generating Q_k for RSDFO-GN. These results use a budget of $10(n + 1)$ evaluations

both RSDFO-GN and DFBGN are random we perform 10 independent runs of each solver/problem combination.

In Fig. 1 we plot the objective decrease attained by each solver versus the number of objective evaluations[9] and iterations. We see that DFBGN significantly outperforms all variants of RSDFO-GN (i.e. all approaches for generating Q_k) for all choices of subspace dimension p tested when measured in terms of evaluations. The primary benefit of DFBGN in this context is that it reuses objective evaluations between iterations, rather than having to fully resample an interpolation set whenever the subspace Q_k is redrawn. This is most clearly seen by the relative performance of RSDFO-GN being better when measured on iterations than on evaluations (noting that DFBGN can perform many more iterations within a given evaluation budget).

4.2 Subspace interpolation models

Similar to Sect. 3, we assume that, at iteration k, our interpolation set has $p+1$ points $\{\boldsymbol{x}_k, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_p\} \subset \mathbb{R}^n$ with $1 \le p \le n$. However, we assume that these points are already given, and use them to determine the space $\mathcal{Y}_k$ (as defined by Q_k). That is, given

$$W_k := \begin{bmatrix} (\boldsymbol{y}_1 - \boldsymbol{x}_k)^T \\ \vdots \\ (\boldsymbol{y}_p - \boldsymbol{x}_k)^T \end{bmatrix} \in \mathbb{R}^{p\times n}, \tag{116}$$

we compute the QR factorization

$$W_k^T = Q_k R_k, \tag{117}$$

[9] Throughout we measure evaluation budgets in (simplex) gradients; that is, evaluations in units of $n + 1$.

where $Q_k \in \mathbb{R}^{n\times p}$ has orthonormal columns and $R_k \in \mathbb{R}^{p\times p}$ is upper triangular—and invertible provided W_k^T is full rank, which we guarantee by judicious replacement of interpolation points. This gives us the Q_k that defines $\mathcal{Y}_k$ via (3)—in this case Q_k has orthonormal columns—and in this way all our interpolation points are in $\mathcal{Y}_k$.

Since each $\boldsymbol{y}_t \in \mathcal{Y}_k$, from (117) we have $\boldsymbol{y}_t = \boldsymbol{x}_k + Q_k\hat{\boldsymbol{s}}_t$, where $\hat{\boldsymbol{s}}_t$ is the t-th column of R_k. Hence we have $\hat{W}_k = R_k^T$ in (110) and so $\hat{\boldsymbol{m}}_k$ (108) is given by solving

$$R_k^T \hat{J}_k^T = \begin{bmatrix} (\boldsymbol{r}(\boldsymbol{y}_1) - \boldsymbol{r}(\boldsymbol{x}_k))^T \\ \vdots \\ (\boldsymbol{r}(\boldsymbol{y}_p) - \boldsymbol{r}(\boldsymbol{x}_k))^T \end{bmatrix}, \tag{118}$$

via forward substitution, since R_k^T is lower triangular. This ultimately gives us our local model $\hat{m}_k$ via (111).

We reiterate that compared to RSDFO-GN, we have used the interpolation set Y_k to determine both Q_k and $\hat{m}_k$, rather than first sampling Q_k, then finding interpolation points $Y_k \subset \mathcal{Y}_k$ with which to construct $\hat{m}_k$. This difference is crucial in allowing the reuse of interpolation points between iterations, and hence lowering the objective evaluation requirements of model construction.

Remark 5 As discussed in [69, Chapter 7.3], we can equivalently recover this construction by asking for a full-space model $\boldsymbol{m}_k : \mathbb{R}^n \to \mathbb{R}^m$ given by $\boldsymbol{m}_k(\boldsymbol{s}) = \boldsymbol{r}(\boldsymbol{x}_k) + J_k\boldsymbol{s}$ such that the interpolation conditions $\boldsymbol{m}_k(\boldsymbol{y}_t - \boldsymbol{x}_k) = \boldsymbol{r}(\boldsymbol{y}_t)$ are satisfied and J_k has minimal Frobenius norm.

4.3 Complete DFBGN algorithm

A complete statement of DFBGN is given in Algorithm 3. Compared to RSDFO-GN, we include specific steps to manage the interpolation set, which in turn dictates the choice of subspace $\mathcal{Y}_k$. Specifically, one issue with our approach is that our new iterate $\boldsymbol{x}_k + \boldsymbol{s}_k$ is in $\mathcal{Y}_k$, so if we were to simply add $\boldsymbol{x}_k + \boldsymbol{s}_k$ into the interpolation set, $\mathcal{Y}_k$ would not change across iterations, and we will never explore the whole space. On the other hand, unlike RSDFO and RSDFO-GN we do not want to completely resample Q_k as this would require too many objective evaluations. Instead, in DFBGN we delete a subset of points from the interpolation set and add new directions orthogonal to the existing directions, which ensures that $Q_{k+1} \neq Q_k$ in every iteration.[10]

We also note that DFBGN does not include some important algorithmic features present in RSDFO-GN, DFO-LS or other model-based DFO methods, and hence is quite simple to state. These features are not necessary for a variety of reasons, which we now outline.

[10] By contrast, the optional growing mechanism in DFO-LS (Remark 4) is designed such that $\boldsymbol{x}_k + \boldsymbol{s}_k$ is not in $\mathcal{Y}_k$, and so the search space is automatically expanded at every iteration. However, this requires an expensive SVD of $J_k \in \mathbb{R}^{m\times n}$ at every iteration, and so is not suitable for our large-scale setting.

Algorithm 3 DFBGN: Derivative-Free Block Gauss–Newton for solving (107).

```
Input: Starting point $x_0 \in \mathbb{R}^n$, initial trust region radius $\Delta_0 > 0$, subspace dimension $p \in \{1, \ldots, n\}$, and
    number of points to drop at each iteration $p_{\text{drop}} \in \{1, \ldots, p\}$.
    Parameters: maximum and minimum trust-region radii $\Delta_{\max} \geq \Delta_0 > \Delta_{\text{end}} > 0$, trust-region radius
    scalings $0 < \gamma_{\text{dec}} < 1 < \gamma_{\text{inc}} \leq \overline{\gamma}_{\text{inc}}$, and acceptance thresholds $0 < \eta_1 \leq \eta_2 < 1$.
1:  Select random orthonormal directions $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_p \in \mathbb{R}^n$ using Algorithm 5, and build initial interpolation
    set $Y_0 := \{\boldsymbol{x}_0, \boldsymbol{x}_0 + \Delta_0\boldsymbol{d}_1, \ldots, \boldsymbol{x}_0 + \Delta_0\boldsymbol{d}_p\}$.
2:  for $k = 0, 1, 2, \ldots$ do
3:      Given $\boldsymbol{x}_k$ and $Y_k$, solve (118) to build subspace models $\hat{\boldsymbol{m}}_k : \mathbb{R}^p \to \mathbb{R}^m$ (108) and $\hat{m}_k : \mathbb{R}^p \to \mathbb{R}$
        (111).
4:      Approximately solve the subspace trust-region subproblem in $\mathbb{R}^p$ (5) and calculate the step $\boldsymbol{s}_k =$
        $Q_k\hat{\boldsymbol{s}}_k \in \mathbb{R}^n$.
5:      Evaluate $\boldsymbol{r}(\boldsymbol{x}_k + \boldsymbol{s}_k)$ and calculate ratio $\rho_k$ (7).
6:      Accept/reject step and update trust region radius: set $\boldsymbol{x}_{k+1}$ and $\Delta_{k+1}$ as per (8).
7:      if $\Delta_{k+1} \leq \Delta_{\text{end}}$, terminate.
8:      if $p < n$ then
9:          Set $Y_{k+1}^{\text{init}} = Y_k \cup \{\boldsymbol{x}_k + \boldsymbol{s}_k\}$.
10:         Remove $\min(\max(p_{\text{drop}}, 2), p)$ points from $Y_{k+1}^{\text{init}}$ (without removing $\boldsymbol{x}_{k+1}$) using Algorithm 4.
11:     else
12:         Set $Y_{k+1}^{\text{init}} = Y_k \cup \{\boldsymbol{x}_k + \boldsymbol{s}_k\} \setminus \{\boldsymbol{y}\}$ for some $\boldsymbol{y} \in Y_k \setminus \{\boldsymbol{x}_{k+1}\}$.
13:         Remove $\min(\max(p_{\text{drop}}, 1), p)$ points from $Y_{k+1}^{\text{init}}$ (without removing $\boldsymbol{x}_{k+1}$) using Algorithm 4.
14:     end if
15:     Let $q := p + 1 - |Y_{k+1}^{\text{init}}|$, and generate random orthonormal vectors $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_q\}$ that are also
        orthogonal to $\{\boldsymbol{y} - \boldsymbol{x}_{k+1} : \boldsymbol{y} \in Y_{k+1}^{\text{init}} \setminus \{\boldsymbol{x}_{k+1}\}\}$, using Algorithm 5.
16:     Set $Y_{k+1} = Y_{k+1}^{\text{init}} \cup \{\boldsymbol{x}_{k+1} + \Delta_{k+1}\boldsymbol{d}_1, \ldots, \boldsymbol{x}_{k+1} + \Delta_{k+1}\boldsymbol{d}_q\}$.
17: end for
```

No criticality and safety steps Compared to RSDFO-GN, the implementation of DFBGN does not include criticality (which is also the case in DFO-LS) or safety steps. These steps ultimately function to ensure we do not have $\hat{\boldsymbol{g}}_k \ll \Delta_k$. In DFBGN, we ensure Δ_k does not get too large compared to $\|\boldsymbol{s}_k\|$ through (8), while $\|\boldsymbol{s}_k\|$ is linked to $\|\hat{\boldsymbol{g}}_k\|$ through Lemma 1. If $\|\boldsymbol{s}_k\|$ is much smaller than Δ_k and our step produces a poor objective decrease, then we will set $\Delta_{k+1} \leftarrow \|\boldsymbol{s}_k\|$ for the next iteration. Although Lemma 1 allows $\|\boldsymbol{s}_k\|$ to be large even if $\|\boldsymbol{g}_k\|$ is small, in practice we do not observe $\Delta_k \gg \|\boldsymbol{g}_k\|$ without DFBGN setting $\Delta_{k+1} \leftarrow \|\boldsymbol{s}_k\|$ after a small number of iterations.

No model-improving steps An important feature of model-based DFO methods are model-improving procedures, which change the interpolation set to ensure Λ-poisedness (Definition 4), or equivalently ensure that the local model for f is fully linear. In RSDFO-GN for instance, model-improvement is performed when `CHECK_MODEL=TRUE`, whereas in [29, Algorithm 4.1] or [30, Algorithm 10.1] there are dedicated model-improvement phases.

Instead, DFBGN ensures accurate interpolation models via a geometry-aware (i.e. Λ-poisedness aware) process for deleting interpolation points at each iteration, where they are replaced by new points in directions (from $\boldsymbol{x}_{k+1}$) which are orthogonal to Q_k and selected at random. The process for deleting interpolation points—and choosing a suitable number of points to remove, p_{drop}—at each iteration are considered in Sects. 4.4 and 4.5 respectively. The process for generating new interpolation points, Algorithm 5, is outlined in Sect. 4.4.

A downside of our approach is that the new orthogonal directions are not chosen by minimizing a model for the objective (i.e. not attempting to reduce the objective), as we have no information about how the objective varies outside $\mathcal{Y}_k$. This is the fundamental trade-off between a subspace approach and standard methods (such as DFO-LS); we can reduce the linear algebra cost, but must spend objective evaluations to change the search space between iterations.

Linear algebra cost of DFBGN As in Sect. 3.3, our approach in DFBGN yields substantial reductions in the required linear algebra costs compared to DFO-LS:

- Model construction costs $\mathcal{O}(np^2)$ for the factorization (117) and $\mathcal{O}(mp^2)$ for back-substitution solves (118), rather than $\mathcal{O}(n^3)$ and $\mathcal{O}(mn^2)$ respectively for DFO-LS; and
- Lagrange polynomial construction costs $\mathcal{O}(p^3)$ rather than $\mathcal{O}(n^3)$.[11]

As well as these reductions, we also get a smaller trust-region subproblem (5)—in $\mathbb{R}^p$ rather than $\mathbb{R}^n$—and smaller memory requirements for storing the model Jacobian: we only store $\hat{J}_k$ and Q_k, requiring $\mathcal{O}((m+n)p)$ memory rather than $\mathcal{O}(mn)$ for storing the full $m \times n$ Jacobian. However, in (5), we do have the extra cost of projecting $\hat{\boldsymbol{s}}_k \in \mathbb{R}^p$ into the full space $\mathbb{R}^n$, which requires a multiplication by Q_k, costing $\mathcal{O}(np)$. In addition to the reduced linear algebra costs, the smaller interpolation set means we have a lower evaluation cost to construct the initial model of $p+1$ evaluations (rather than $n+1$).

No particular choice of p is needed for this method, and anything from $p=1$ (i.e. coordinate search) to $p=n$ (i.e. full space search) is allowed. However, unsurprisingly, we shall see that larger values of p give better performance in terms of evaluations, except for the very low-budget phase, where smaller values of p benefit from a lower initialization cost. Hence, we expect that our approach with small p is useful when the $\mathcal{O}(mn^2+n^3)$ per-iteration linear algebra cost of DFO-LS is too great, and reducing the linear algebra cost is worth (possibly) needing more objective evaluations to achieve a given accuracy. As a result, p should in general be set as large as possible, given the linear algebra costs the user is willing to bear.

In Table 1, we compare the linear algebra costs of DFO-LS and DFBGN. The overall per-iteration cost of DFO-LS is $\mathcal{O}(mn^2+n^3)$ and the cost of DFBGN is $\mathcal{O}(mp^2+np^2+p^3)$, depending on the choice of $p \in \{1,\dots,n\}$. The key benefit is that our dependency on the underlying problem dimension n decreases from cubic in DFO-LS to linear in DFBGN (provided $p \ll n$). We also note that both methods have linear cost in the number of residuals m, but with a factor that is significantly smaller in DFBGN than in DFO-LS—$\mathcal{O}(p^2)$ compared to $\mathcal{O}(n^2)$.

Remark 6 In every iteration we must compute the QR factorization (117). However, we note, similar to [19, Section 4.2], that adding, removing and changing interpolation points all induce simple changes to $\hat{W}_k^T$ (adding or removing columns, and low-rank updates). This means that (117) can be computed with cost $\mathcal{O}(np)$ per iteration using

[11] Here, we use the existing factorization (117) and solve with $p+1$ right-hand sides, as in Sect. 3.3.

Table 1 Comparison of per-iteration linear algebra costs of DFO-LS and DFBGN (Algorithm 3) with subspace dimension $p \in \{1, \dots, n\}$

Algorithm phase	DFO-LS	DFBGN	Comment
Form $\hat{\boldsymbol{m}}_k$ (108)	$\mathcal{O}(n^3 + mn^2)$	$\mathcal{O}(np^2 + mp^2)$	Factorization plus linear solves
Form $\hat{m}_k$ (111)	$\mathcal{O}(mn^2)$	$\mathcal{O}(mp^2)$	Form $\hat{J}_k^T \hat{J}_k$
Trust-region subproblem	$\mathcal{O}(n^2)$–$\mathcal{O}(n^3)$	$\mathcal{O}(p^2)$–$\mathcal{O}(p^3)$	Depending on # CG iterations*
Calculate $\boldsymbol{s}_k = Q_k \hat{\boldsymbol{s}}_k$	–	$\mathcal{O}(np)$	
Form new step $\boldsymbol{x}_k + \boldsymbol{s}_k$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	
Choose point to replace	$\mathcal{O}(n^3)$	–	Compute Lagrange polynomials
Model improvement	$\mathcal{O}(n^3)$	–	Recompute Lagrange polynomials
Choose points to remove	–	$\mathcal{O}(p^3 + np)$	See Algorithm 4
Generate new directions	–	$\mathcal{O}(np^2)$	See Algorithm 5
Total	$\mathcal{O}(mn^2 + n^3)$	$\mathcal{O}(mp^2 + np^2 + p^3)$	

*Note that the trust-region subproblem is solved using a truncated CG method [27, Chapter 7.5.1] originally from [67] in both DFO-LS and DFBGN

the updating methods in [41, Section 12.5]. In our implementation, however, we do not do this, as we find that these updates introduce errors[12] that accumulate at every iteration and reduce the accuracy of the resulting interpolation models. To maintain the numerical performance of our method, we need to recompute (117) from scratch regularly (e.g. every 10 iterations), and so would not see the $\mathcal{O}(np)$ per-iteration cost, on average.

Remark 7 The default parameter choices for DFBGN are the same as DFO-LS, namely: $\Delta_{\max} = 10^{10}$, $\Delta_{\text{end}} = 10^{-8}$, $\gamma_{\text{dec}} = 0.5$, $\gamma_{\text{inc}} = 2$, $\overline{\gamma}_{\text{inc}} = 4$, $\eta_1 = 0.1$, and $\eta_2 = 0.7$. DFBGN also uses the same default choice $\Delta_0 = 0.1 \max(\|\boldsymbol{x}_0\|_\infty, 1)$. The default choice of p_{drop} is discussed in Sect. 4.5.

Adaptive choice of p One approach that we have considered is to allow p to vary between iterations of DFBGN, rather than being constant throughout. Instead of adding p_{drop} new points at the end of each iteration (line 15), we implement a variable p by adding at least one new point to the interpolation set, continuing until some criterion is met. This criterion is designed to allow p small when such a p allows us to make reasonable progress, but to grow p up to $p \approx n$ when necessary.

We have tested several possible criteria—comparing some combination of model gradient and Hessian, trust-region radius, trust-region step length, and predicted decrease from the trust-region step—and found the most effective to be comparing the model gradient and Hessian with the trust-region radius. Specifically, we continue adding new directions until (c.f. Lemma 3 and [19, Lemma 3.22])

$$\frac{\|\boldsymbol{g}_k\|}{\max(\|H_k\|, 1)} \geq \alpha \Delta_k, \tag{119}$$

[12] Leading to $\hat{W}_k^T \neq Q_k R_k$, not relating to Q_k orthogonal or R_k upper-triangular.

Algorithm 4 Mechanism for removing points from the interpolation set in DFBGN.

```
Input: Interpolation set $\{x_{k+1}, y_1, \ldots, y_p\}$ with current iterate $x_{k+1}$, trust-region radius $\Delta_{k+1} > 0$ and
    number of points to remove $p_{\text{drop}} \in \{1, \ldots, p\}$.
1: Compute the (linear) Lagrange polynomials for $\{x_{k+1}, y_1, \ldots, y_p\}$ in the same way as (118).
2: For $t = 1, \ldots, p$ (i.e. all interpolation points except $x_{k+1}$), compute
       $$\theta_t := \max_{x \in B(x_{k+1}, \Delta_{k+1})} |\ell_t(x)| \cdot \max\left(\frac{\|y_t - x_{k+1}\|^4}{\Delta_{k+1}^4}, 1\right). \quad (120)$$
3: Remove the $p_{\text{drop}}$ interpolation points with the largest values of $\theta_t$.
```

for some $\alpha > 0$ (we use $\alpha = 0.2(n-p)/n$ for an interpolation set with $p+1$ points). However, our numerical testing has shown that DFBGN with p fixed outperforms this approach for all budget and accuracy levels, on both medium- and large-scale problems, and so we do not consider it further here. We delegate further study of this approach to future work, to see if alternative adaptive choices for p can be beneficial.

4.4 Interpolation set management

We now provide more details about how we manage the interpolation set in DFBGN. Specifically, we discuss how points are chosen for removal from Y_k, and how new interpolations points are calculated.

4.4.1 Geometry management

In the description of DFBGN, there are no explicit mechanisms to ensure that the interpolation set is well-poised. DFBGN ensures that the interpolation set has good geometry through two mechanisms:

- We use a geometry-aware mechanism for removing points, based on [19, 67], which requires the computation of Lagrange polynomials. This mechanism is given in Algorithm 4, and is called in lines 10 and 13 of DFBGN, as well as to select a point to replace in line 12; and
- Adding new directions that are orthogonal to existing directions, and of length Δ_k, means adding these new points never causes the interpolation set to have poor poisedness.

Together, these two mechanisms mean that any points causing poor poisedness are quickly removed, and replaced by high-quality interpolation points (orthogonal to existing directions, and within distance Δ_k of the current iterate). We note that the simpler approach of removing points based on distance to the current iterate alone does not perform as well as this method (see Appendix B.1 for details).

The linear algebra cost of Algorithm 4 is $\mathcal{O}(p^3)$ to compute p Lagrange polynomials with cost $\mathcal{O}(p^2)$ each (since we already have a factorization of $\hat{W}_k^T$). Then for each t we must evaluate θ_t (120), with cost $\mathcal{O}(p)$ to maximize $\ell_t(x)$ (since ℓ_t is linear and

Algorithm 5 Mechanism for generating new directions in DFBGN.

```
Input: Orthonormal basis for current subspace $Q \in \mathbb{R}^{n \times p_1}$ (optional), number of new directions $q \le n - p_1$.
1: Generate $A \in \mathbb{R}^{n \times q}$ with i.i.d. standard normal entries.
2: If $Q$ is specified, calculate $\widetilde{A} = A - QQ^T A$, otherwise set $\widetilde{A} = A$.
3: Perform the QR factorization $\widetilde{Q}\widetilde{R} = \widetilde{A}$ and return $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_q$ as the columns of $\widetilde{Q}$.
```

varies only in directions $\mathcal{Y}_k$), and $\mathcal{O}(n)$ to calculate $\|\boldsymbol{y}_t - \boldsymbol{x}_{k+1}\|$. This gives a total cost of $\mathcal{O}(p^3 + np)$.[13]

4.4.2 Generation of new directions

We now detail how new directions $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_q$ are created in line 15 of DFBGN (Algorithm 3). The same approach is suitable for generating the initial directions $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_p$ in line 1 of DFBGN, using $\widetilde{A} = A$ below (i.e. no Q required).

Suppose our current subspace is defined by the orthonormal columns of $Q \in \mathbb{R}^{n \times p_1}$, and we wish to generate q new orthonormal vectors that are also orthogonal to the columns of Q (with $p_1 + q \le n$). When called in line 15 of DFBGN, we will have $p_1 = p - p_{\text{drop}}$ and $q = p_{\text{drop}}$. We use the approach in Algorithm 5. From the QR factorization, the columns of $\widetilde{Q}$ are orthonormal, and if $\widetilde{A}$ is full rank (which occurs with probability 1; see Lemma 12 below) then we also have $\text{col}(\widetilde{Q}) = \text{col}(\widetilde{A})$. So, to confirm the columns of $\widetilde{Q}$ are orthogonal to Q, we only need to check that the columns of $\widetilde{A}$ are orthogonal to Q. Let $\widetilde{\boldsymbol{a}}_i$ be the i-th column of $\widetilde{A}$ and $\boldsymbol{q}_j$ be the j-th column of Q. Then, if $\boldsymbol{a}_i$ is the i-th column of A, we have

$$\widetilde{\boldsymbol{a}}_i^T \boldsymbol{q}_j = \boldsymbol{a}_i^T (I - QQ^T)\boldsymbol{q}_j = \boldsymbol{a}_i^T (\boldsymbol{q}_j - Q\boldsymbol{e}_j) = 0, \tag{121}$$

as required.

The cost of Algorithm 5 is $\mathcal{O}(nq)$ to generate A, $\mathcal{O}(np_1q)$ to form $\widetilde{A}$ and $\mathcal{O}(nq^2)$ for the QR factorization. Since $p_1, q \le p$ (since p_1 is the number of directions remaining in the interpolation set and q is the number of new directions to be added), the whole process has cost at most $\mathcal{O}(np^2)$. This bound is tight, up to constant factors, as we could take $p_1 = q = p/2$, for instance.

Lemma 12 *The matrix $\widetilde{A}$ has full column rank with probability 1.*

Proof Let $\boldsymbol{a}_i$ and $\widetilde{\boldsymbol{a}}_i$ be the i-th columns of A and $\widetilde{A}$ respectively. From [33, Proposition 7.1], A has full column rank with probability 1, and each $\boldsymbol{a}_i \notin \text{col}(Q)$ with probability 1. Now suppose we have constants $c_1, \ldots, c_q$ so that $\sum_{i=1}^q c_i \widetilde{\boldsymbol{a}}_i = \boldsymbol{0}$. Then since $\widetilde{\boldsymbol{a}}_i = \boldsymbol{a}_i - QQ^T\boldsymbol{a}_i$, we have

$$\sum_{i=1}^q c_i \boldsymbol{a}_i = \sum_{i=1}^q c_i QQ^T \boldsymbol{a}_i. \tag{122}$$

[13] We could instead compute $\|\boldsymbol{y}_t - \boldsymbol{x}_{k+1}\|$ by taking the norm of the t-th column of R_k, provided we have the factorization (117), for cost $\mathcal{O}(p)$ for each t. This does not affect the overall conclusion of Table 1.

The right-hand side is in $\mathrm{col}(Q)$, so since $\boldsymbol{a}_i \notin \mathrm{col}(Q)$, we must have $\sum_{i=1}^{q} c_i \boldsymbol{a}_i = \boldsymbol{0}$. Thus $c_1 = \cdots = c_q = 0$ since A has full column rank, and so $\widetilde{A}$ has full column rank. □

4.5 Selecting an appropriate value of p_{drop}

An important component of DFBGN that we have not yet specified is how many points to remove from the interpolation set at each iteration, $p_{\text{drop}} \in \{1, \ldots, p\}$.

On one hand, a large p_{drop} enables us to change the subspace by a large amount between iterations, ensuring we explore the whole of $\mathbb{R}^n$ quickly, rather than searching in unproductive subspaces for many iterations. However, a small p_{drop} means we require few objective evaluations per iteration, and so are more likely to use our evaluation budget efficiently.

In DFBGN we use a compromise choice as the default mechanism: $p_{\text{drop}} = 1$ on successful iterations and $p_{\text{drop}} = p/10$ on unsuccessful iterations. Our careful and extensive testing show that this is a successful choice in practice, because it ensures that the trust-region radius Δ_k does not decrease too quickly compared to the first-order optimality measure $\|\hat{\boldsymbol{g}}_k\|$. We detail our choices and approach in Appendix B.2.

5 Numerical results

In this section we compare the performance of DFBGN (Algorithm 3) to that of DFO-LS. We note that that DFO-LS has been shown to have state-of-the-art performance compared to other solvers in [15]. As described in Sect. 4.3, the implementation of DFBGN is based on the decision to reduce the linear algebra cost of the algorithm at the expense of more objective evaluations per iteration. However, we still maintain the goal of DFBGN achieving (close to) state-of-the-art performance when it is run as a 'full space' method (i.e. $p = n$). Here, we will investigate this tradeoff in practice.

5.1 Testing framework

In our testing, we will compare a Python implementation of DFBGN (Algorithm 3) against DFO-LS version 1.0.2 (also implemented in Python). The implementation of DFBGN is available on Github.[14] We will consider both the standard version of DFO-LS, and one where we use a reduced initialization cost of $n/100$ evaluations (c.f. Remark 4). This will allow us to compare both the overall performance of DFBGN and its performance with small budgets (since DFBGN also has a reduced initialization cost of $p + 1$ evaluations). We compare these against DFBGN with the choices $p \in \{n/100, n/10, n/2, n\}$ and the adaptive choice of $p_{\text{drop}} \in \{1, p/10\}$ (Sect. 4.5). All default settings are used for both solvers, and since both are randomized (DFO-LS uses random initial directions only, and DFBGN is randomized through Algorithm 5), we run 10 instances of each problem under all solver configurations.

[14] See https://github.com/numericalalgorithmsgroup/dfbgn. Results here use version 0.1.

Test problems We will consider two collections of nonlinear least-squares test problems, both taken from the CUTEst collection [42]. The first, denoted (CR), is a collection of 60 medium-scale problems (with $25 \leq n \leq 110$ and $n \leq m \leq 400$). Full details of the (CR) collection may be found in [19, Table 3]. The second, denoted (CR-large), is a collection of 28 large-scale problems (with $1000 \leq n \leq 5000$ and $n \leq m \leq 9998$). This collection is a subset of problems from (CR), with their dimension increased substantially. Full details of the (CR-large) collection are given in Appendix C. Note that the 12 h runtime limit was only relevant for (CR-large) in all cases.

Measuring solver performance For every problem, we allow all solvers a budget of at most $100(n+1)$ objective evaluations (i.e. evaluations of the full vector $\boldsymbol{r}(\boldsymbol{x})$). This dimension-dependent choice may be understood as equivalent to 100 evaluations of $\boldsymbol{r}(\boldsymbol{x})$ and the Jacobian $J(\boldsymbol{x})$ via finite differencing. However, given the importance of linear algebra cost to our comparisons, we allow each solver a maximum runtime of 12 h for each instance of each problem.[15] For each solver S, each problem instance P, and accuracy level $\tau \in (0, 1)$, we calculate

$$N(S, P, \tau) := \#\text{ evaluations of } \boldsymbol{r}(\boldsymbol{x}) \text{ required to find a point } \boldsymbol{x} \text{ with} \\ f(\boldsymbol{x}) \leq f(\boldsymbol{x}^*) + \tau(f(\boldsymbol{x}_0) - f(\boldsymbol{x}^*)), \tag{123}$$

where $f(\boldsymbol{x}^*)$ is an estimate of the minimum of f as listed in [19, Table 3] for (CR) and Appendix C for (CR-large). If this objective decrease is not achieved by a solver before its budget or runtime limit is hit, we set $N(S, P, \tau) = \infty$. We then compare solver performances on a problem collection $\mathcal{P}$ by plotting either data profiles [59]

$$d_{S,\tau}(\alpha) := \frac{1}{|\mathcal{P}|} \left|\{P \in \mathcal{P} : N(S, P, \tau) \leq \alpha(n_P + 1)\}\right|, \tag{124}$$

where n_P is the dimension of problem instance P and $\alpha \in [0, 100]$ is an evaluation budget (in "gradients", or multiples of $n + 1$), or performance profiles [32]

$$\pi_{S,\tau}(\alpha) := \frac{1}{|\mathcal{P}|} \left|\{P \in \mathcal{P} : N(S, P, \tau) \leq \alpha N_{\min}(P, \tau)\}\right|, \tag{125}$$

where $N_{\min}(P, \tau)$ is the minimum value of $N(S, P, \tau)$ for any solver S, and $\alpha \geq 1$ is a performance ratio. In some instances, we will plot profiles based on runtime rather than objective evaluations. For this, we simply replace "number of evaluations of $\boldsymbol{r}(\boldsymbol{x})$" with "runtime" in (123).

When we plot the objective reduction achieved by a given solver, we normalize the objective value to be in [0, 1] by plotting

$$\frac{f(\boldsymbol{x}) - f(\boldsymbol{x}^*)}{f(\boldsymbol{x}_0) - f(\boldsymbol{x}^*)}, \tag{126}$$

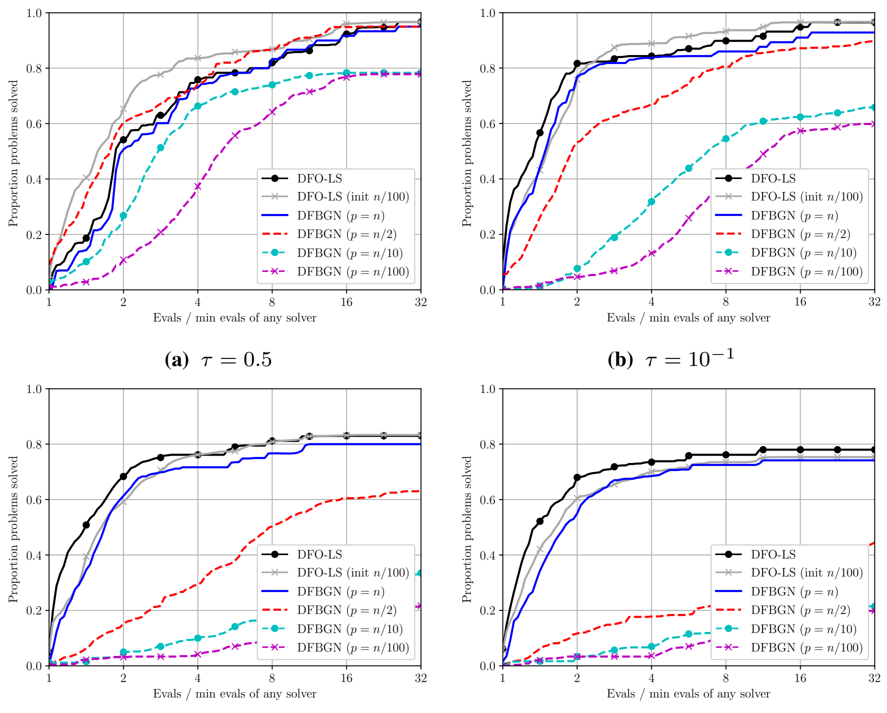
[15] Since all problems are implemented in Fortran via CUTEst, the cost of objective evaluations for this testing is minimal.

which corresponds to the best τ achieved in (123) after a given number of evaluations (again measured in "gradients") or runtime.

5.2 Results based on evaluations

We begin our comparisons by considering the performance of DFO-LS and DFBGN when measured in terms of evaluations.

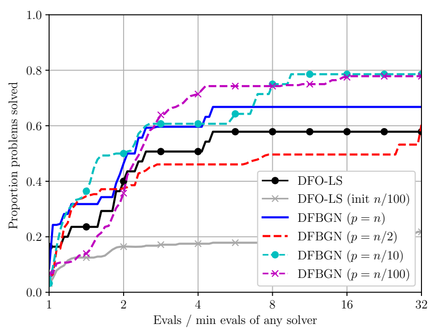
Medium-scale problems (CR) First, in Fig. 2, we show the results for different accuracy levels for the (CR) problem collection (with $n \approx 100$). For the lowest accuracy level $\tau = 0.5$, DFO-LS with reduced initialization cost is the best-performing solver, followed by DFBGN with $p = n/2$. These correspond to methods with lower initialization costs than DFO-LS and DFBGN with $p = n$, so this is likely a large driver behind their performance. DFBGN with full space size $p = n$ performs similarly to DFO-LS, and DFBGN with $p = n/10$ and $p = n/100$ perform worst (as they are optimizing in a very small subspace at each iteration).



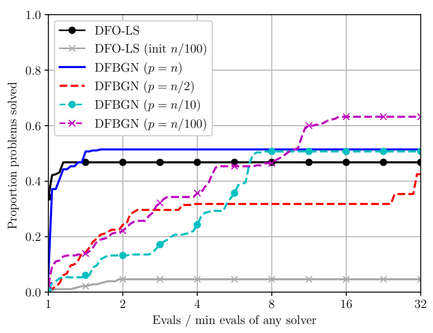
(a) $\tau = 0.5$ **(b)** $\tau = 10^{-1}$

(c) $\tau = 10^{-3}$ **(d)** $\tau = 10^{-5}$

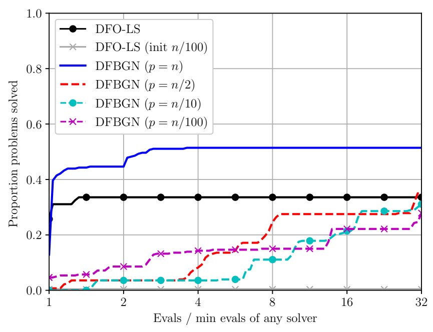
Fig. 2 Performance profiles (in evaluations) comparing DFO-LS (with and without reduced initialization cost) with DFBGN (various p choices) for different accuracy levels. Results are an average of 10 runs for each problem, with a budget of $100(n+1)$ evaluations and a 12 h runtime limit per instance. The problem collection is (CR)



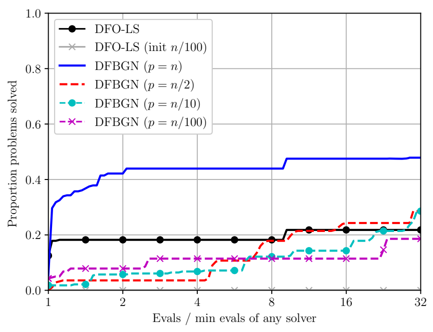
(a) $\tau = 0.5$



(b) $\tau = 10^{-1}$



(c) $\tau = 10^{-3}$



(d) $\tau = 10^{-5}$

Fig. 3 Performance profiles (in evaluations) comparing DFO-LS (with and without reduced initialization cost) with DFBGN (various p choices) for different accuracy levels. Results are an average of 10 runs for each problem, with a budget of $100(n+1)$ evaluations and a 12 h runtime limit per instance. The problem collection is (CR-large)

However, as we look at higher accuracy levels, we see that DFO-LS (with and without reduced initialization cost) performs best, and the DFBGN methods perform worse. The performance gap is more noticeable for small values of p. As expected, this means that DFBGN requires more evaluations to achieve these levels of accuracy, and benefits from being allowed to use a larger p. Notably, DFBGN with $p = n$ has only a slight performance loss compared to DFO-LS, even though it uses $p/10$ evaluations on unsuccessful iterations (rather than 1–2 for DFO-LS); this indicates that our choice of p_{drop} provides a suitable compromise between solver robustness and evaluation efficiency.

Large-scale problems (CR-large) Next, in Fig. 3, we show the same plots but for the (CR-large) problem collection, with $n \approx 1000$. Compared to Fig. 2, the situation is quite different.

At the lowest accuracy level, $\tau = 0.5$, DFBGN with small subspaces ($p = n/10$ and $p = n/100$) gives the best-performing solvers, followed by the full-space solvers (DFO-LS and DFBGN with $p = n$). For higher accuracy levels, the performance of DFBGN with small p deteriorates compared with the full-space methods. DFBGN with $p = n/2$ is the worst-performing DFBGN variant at low accuracy levels, and performs similar to DFBGN with small p at high accuracy levels. DFO-LS with reduced initialization cost is the worst-performing solver for this dataset.

Unlike the medium-scale results above, we no longer have a clear trend in the performance of DFBGN as we vary p. Instead, we have a combination of two factors coming into play, which have opposite impacts on the performance of DFBGN as we vary p. On one hand, we have the number of evaluations required for DFBGN (with a given p) to reach the desired accuracy level. On the other hand, we have the number of iterations that DFBGN can perform before reaching the 12 h runtime limit.

DFBGN with small p requires more evaluations to reach a given level of accuracy (as seen with the medium-scale results), but can perform many evaluations before timing out due to its low per-iteration linear algebra cost. This is reflected in it solving many problems to low accuracy, but few problems to high accuracy. By contrast, DFBGN with $p = n$ is allowed to perform fewer iterations before timing out (and hence see fewer evaluations), but requires many fewer evaluations to solve problems, particularly for high accuracy. This manifests in its good performance for low and high accuracy levels. The middle ground, DFBGN with $p = n/2$, has its performance negatively impacted by both issues: it requires many fewer evaluations to solve problems than $p = n$ (especially for high accuracy), but also has a relatively high per-iteration linear algebra cost and times out compared to small p.

Both variants of DFO-LS show worse performance here than for the medium-scale problems. This is because, as suggested by the analysis in Table 1, they are both affected by the runtime limit. DFO-LS with reduced initialization cost is particularly affected, because of the high cost of the SVD (of the full $m \times n$ Jacobian) at each iteration for these problems. We note that this cost is only noticeable for these large-scale problems, and this variant of DFO-LS is still useful for small- and medium-scale problems, as discussed in [15].

We can verify the impact of the timeout on DFO-LS and DFBGN by considering the proportion of problem instances for (CR-large) for which the solver terminated because of the timeout. These results are presented in Table 2. DFO-LS reaches the 12 h maximum much more frequently than DFBGN: over 90% rather than 35% for DFBGN with $p = n/100$ or 66% for DFBGN with $p = n$ (see Remark 8 below). For DFBGN with different values of p, we see the same behaviour as in Fig. 3. That is, DFBGN with small p times out the least frequently, as its low per-iteration runtime means it performs enough iterations to terminate naturally. For DFBGN with $p = n$,

Table 2 Proportion of problem instances from (CR-large) for which each solver terminated on the maximum 12 h runtime

Solver	% timeout
DFO-LS	92.5
DFO-LS (init $n/100$)	97.9
DFBGN ($p = n/100$)	34.6
DFBGN ($p = n/10$)	73.9
DFBGN ($p = n/2$)	81.8
DFBGN ($p = n$)	66.4

we time out more frequently (due to the high per-iteration runtime), but not as often as with $p = n/2$, as the its superior performance in terms of evaluations for high accuracy levels means it fully solves more problems, even with comparatively fewer iterations. We note that Table 2 does not measure what accuracy level was achieved before the timeout, which is better captured in the performance profiles Fig. 3.

Remark 8 DFBGN with $p = n$ has a similar per-iteration linear algebra cost to DFO-LS. Hence it can perform a similar number of iterations before reaching the runtime limit. However, DFBGN performs more objective evaluations per iteration, because of the choice of p_{drop}. Since DFBGN with $p = n$ has a similar performance to DFO-LS when measured by evaluations (as seen in Fig. 2), this means that it has a superior performance when measured by runtime. Additionally, if multiple objective evaluations can be run in parallel, then DFBGN would also be able to benefit from this, unlike DFO-LS.

Remark 9 For completeness, the technical report associated with this work [20, Appendix A] compares DFBGN with DFO-LS on the low-dimensional collection of test problems from Moré and Wild [59]. We do not include this discussion here as these problems are low-dimensional, which is not the main use case for DFBGN.

5.3 Results based on runtime

We have seen above that DFBGN performs well compared to DFO-LS on the (CR-large) problem collection, as the 12 h timeout causes DFO-LS to terminate after relatively few objective evaluations. In Fig. 4, we show the same comparison for (CR-large) as in Fig. 3, but showing data profiles of problems solved versus runtime (rather than evaluations). Here, all DFBGN variants perform similar to or better than DFO-LS for low accuracy levels, since DFBGN has a lower per-iteration runtime than DFO-LS, and this is the regime where DFBGN performs best (on evaluations). For high accuracy levels, DFBGN with $p = n$ is the best-performing solver, as it uses large enough subspaces to solve many problems to high accuracy. By contrast, both DFBGN with small p and DFO-LS perform similarly at high accuracy levels—the impact of the timeout on DFO-LS roughly matches the reduced robustness of DFBGN with small p at these accuracy levels. Again, as we observed above, DFO-LS with reduced initialization cost is the worst-performing solver, due to the high cost of the SVD at each iteration.

To further see the impact of this issue, we now consider how the solvers perform for a variable-dimension test problem, as we increase the underlying dimension. We run each solver, with the same settings as above, on the CUTEst problem ARWHDNE for different choices of problem dimension n.[16] In Fig. 5 we plot the objective reduction for each solver against objective evaluations and runtime for DFO-LS and DFBGN, showing $n = 100$, $n = 1000$ and $n = 2000$.

We see that, when measured on evaluations, both DFO-LS variants achieve the fastest objective reduction, and that DFBGN with small p achieves the slowest objective reduction. This is in line with our results from Sect. 5.2. However, when we

[16] This problem appears in the collections (CR) and (CR-large), with $n = 100$ and $n = 1000$ respectively.

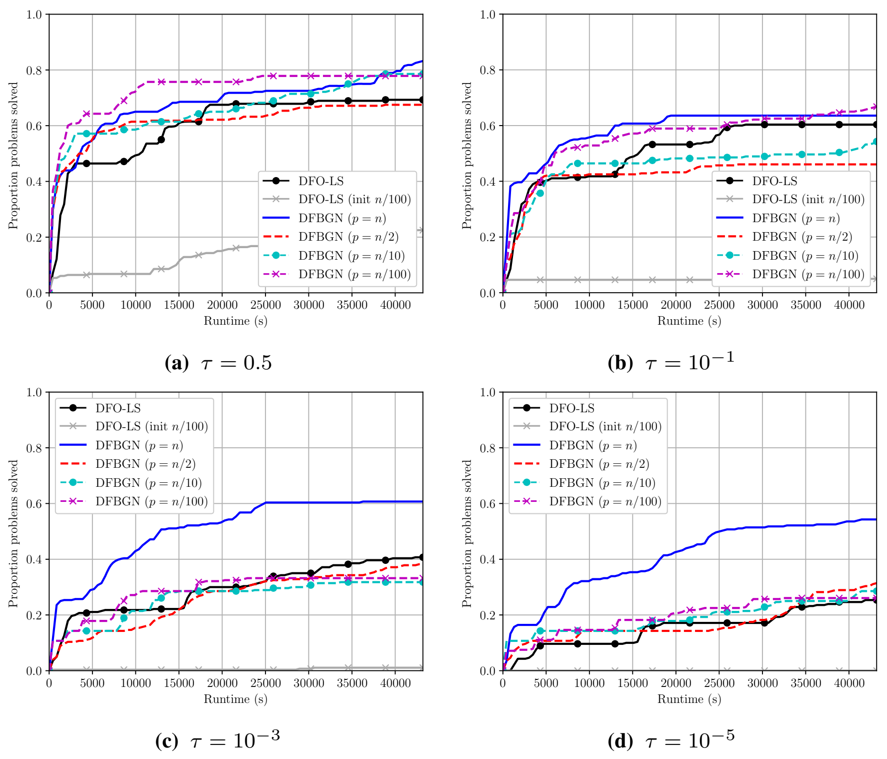


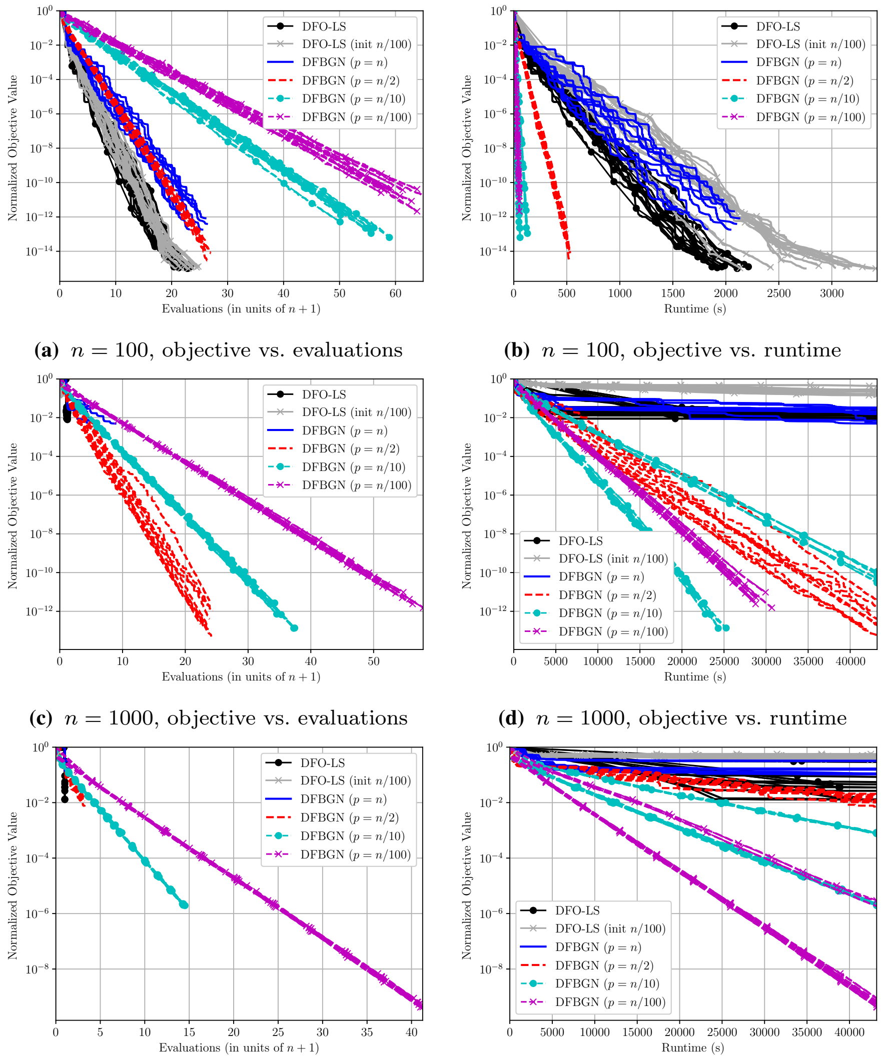
Fig. 4 Data profiles comparing the runtime of DFO-LS (with and without reduced initialization cost) with DFBGN (various p choices) for different accuracy levels. Results are an average of 10 runs for each problem, with a budget of $100(n + 1)$ evaluations and a 12 h runtime limit per instance. The problem collection is (CR-large)

instead consider objective decrease against runtime, we see that DFBGN with small p gives the fastest decrease—the larger number of iterations needed by these variants (as seen by the larger number of evaluations) is offset by the substantially reduced per-iteration linear algebra cost. When viewed against runtime, both DFO-LS variants can only achieve a small objective decrease in the allowed 12 h, even though they are showing fast decrease against number of evaluations, and would achieve higher accuracy than DFBGN if the linear algebra cost were negligible.

5.4 Results for small budgets

Another benefit of DFBGN is that it has a small initialization cost of $p+1$ evaluations. When n is large, it is more likely for a user to be limited by a budget of fewer than n evaluations. Here, we examine how DFBGN compares for small budgets to DFO-LS with reduced initialization cost.

We recall from Remark 4 that DFO-LS with reduced initialization cost progressively increases the dimension of the subspace of its interpolation model, until it reaches the



(a) $n = 100$, objective vs. evaluations **(b)** $n = 100$, objective vs. runtime

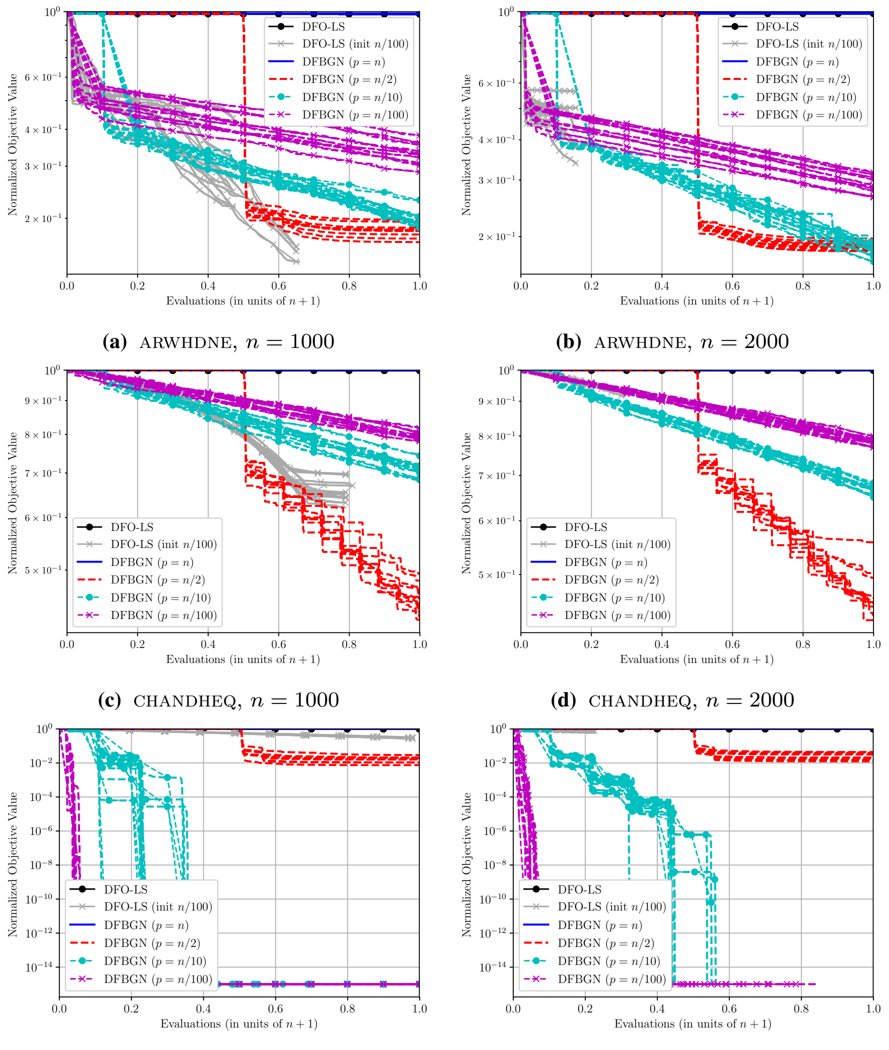
(c) $n = 1000$, objective vs. evaluations **(d)** $n = 1000$, objective vs. runtime

(e) $n = 2000$, objective vs. evaluations **(f)** $n = 2000$, objective vs. runtime

Fig. 5 Normalized objective value (versus evaluations and runtime) for 10 runs of DFO-LS and DFBGN on CUTEst problem ARWHDNE. These results use a budget of $100(n + 1)$ evaluations and a 12 h runtime limit per instance

whole space $\mathbb{R}^n$ (after approximately $n + 1$ evaluations), while in DFBGN we restrict the dimension at all iterations.

In Fig. 6 we consider three variable-dimensional CUTEst problems from (CR) and (CR-large), all using $n = 1000$ and $n = 2000$. We show the objective decrease against number of evaluations for 10 runs of each solver, restricted to a maximum of $n + 1$ evaluations. We see that the smaller p used in DFBGN, the faster DFBGN is able to make progress (due to the lower number of initial evaluations). However, this is



(a) ARWHDNE, $n = 1000$ **(b)** ARWHDNE, $n = 2000$

(c) CHANDHEQ, $n = 1000$ **(d)** CHANDHEQ, $n = 2000$

(e) VARDIMNE, $n = 1000$ **(f)** VARDIMNE, $n = 2000$

Fig. 6 Normalized objective value (versus evaluations) for 10 runs of DFO-LS and DFBGN on different CUTEst problems (all with $n = 1000$ and $n = 2000$). These results use a budget of $n + 1$ evaluations and a 12 h runtime limit per instance

offset by the faster objective decrease achieved by larger p values (after the higher initialization cost)—if the user can afford a larger p, both in terms of linear algebra and initial evaluations, then this is usually a better option. An exception to this is the problem VARDIMNE, where its simple structure means DFBGN with small p solves the problem to very high accuracy with very few evaluations, substantially outperforming both DFBGN with larger p, and DFO-LS with reduced initialization cost.

In Fig. 6 we also show the decrease for DFO-LS with full initialization cost and DFBGN with $p = n$, but they use the full budget on initialization, and so make no

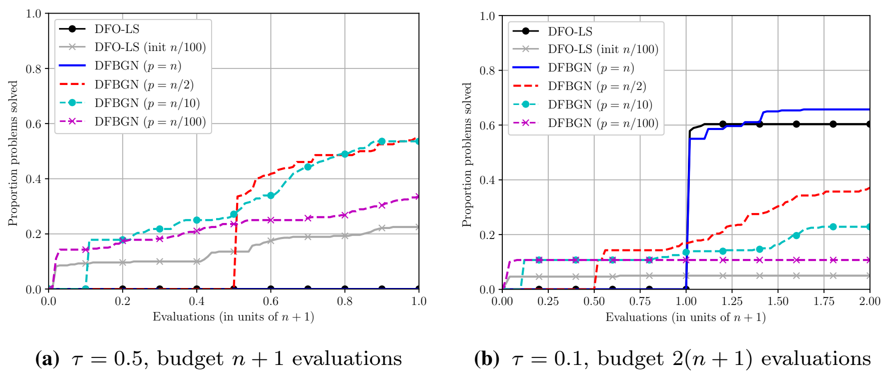
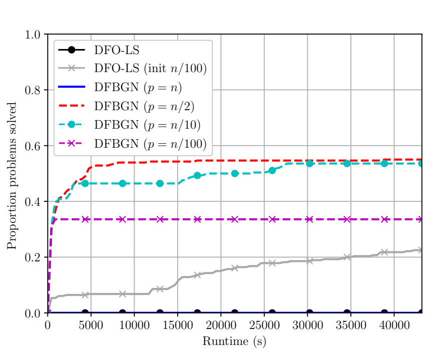


Fig. 7 Data profiles (in evaluations) comparing DFO-LS (with and without reduced initialization cost) with DFBGN (various p choices) for different accuracy levels and budgets. Results are an average of 10 runs for each problem, with a budget of $n + 1$ or $2(n + 1)$ evaluations and a 12 h runtime limit per instance. The problem collection is (CR-large)

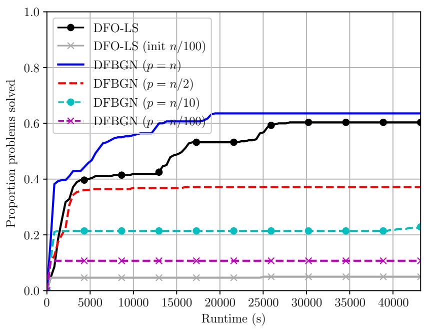
progress. However, in addition, we show DFO-LS with a reduced initialization cost of $n/100$ evaluations. This variant performs well, in most cases matching the decrease of DFBGN with $p = n/100$ initially, but achieving a faster objective reduction against number of evaluations—this matches with our previous observations. However, the extra cost of the linear algebra means that DFO-LS with reduced initialization does not end up using the full budget, instead hitting the 12 h timeout. This is most clear when comparing the results for $n = 1000$ with $n = 2000$, where DFO-LS with reduced initialization cost begins by achieving a similar decrease in both cases, but hits the timeout more quickly with $n = 2000$, and so terminates after fewer objective evaluations (with a corresponding smaller objective decrease).

We analyze this more systematically in Fig. 7, where we show data profiles (measured on number of evaluations) of DFBGN and DFO-LS on the (CR-large) problem collection, for low accuracy and small budgets. These results verify our conclusions: DFBGN with small p can make progress on many problems with a very short budget (fewer than $n + 1$ evaluations), and outperform DFO-LS with reduced initialization cost due to its slow runtime. However, once we reach a budget of more than $n+1$ evaluations, then DFO-LS and DFBGN with $p = n$ become the best-performing solvers (when measuring on evaluations only). They are also able to achieve a higher level of accuracy compared to DFBGN with small p.

Lastly, in Fig. 8 we show the same results as Fig. 7, but showing profiles measured on runtime. We note that we are only measuring the linear algebra costs, as the cost of objective evaluation for our problems is negligible. Here, the benefits of DFBGN with small p are not seen. This is because the problems that can be solved by DFBGN with small p using very few evaluations are likely easier, and so can likely be solved by DFBGN with large p in few iterations. Thus, the runtime requirements for DFBGN with large p to solve the problem are not large—even though they have a higher per-iteration cost, the number of iterations is small. In this setting, therefore, the benefit of DFBGN with small p is not lower linear algebra costs, but fewer evaluations—which is likely to be the more relevant issue in this small-budget regime.



(a) $\tau = 0.5$, $n + 1$ evaluations (vs. runtime)



(b) $\tau = 0.1$, $2(n + 1)$ evaluations (vs. runtime)

Fig. 8 Data profiles (in runtime) comparing DFO-LS (with and without reduced initialization cost) with DFBGN (various p choices) for different accuracy levels and budgets. Results are an average of 10 runs for each problem, with a budget of $n + 1$ or $2(n + 1)$ evaluations and a 12 h runtime limit per instance. The problem collection is (CR-large)

6 Conclusions and future work

The development of scalable derivative-free optimization algorithms is an active area of research with many applications. In model-based DFO, the high per-iteration linear algebra cost associated (primarily) with interpolation model creation and point management is a barrier to its utility for large-scale problems. To address this, we introduce three model-based DFO algorithms for large-scale problems.

First, RSDFO is a general framework for model-based DFO based on model construction and minimization in random subspaces, and is suitable for general smooth nonconvex objectives. This is specialized to nonlinear least-squares problems in RSDFO-GN, a version of RSDFO based on Gauss–Newton interpolation models built in subspaces. Lastly, we introduce DFBGN, a practical implementation of RSDFO-GN. In all cases, the scalability of these methods arises from the construction and minimization of models in p-dimensional subspaces of the ambient space $\mathbb{R}^n$. The subspace dimension can be specified by the user to reflect the computational resources available for linear algebra calculations.

We prove high-probability worst-case complexity bounds for RSDFO, and show that RSDFO-GN inherits the same bounds with an oracle and flop complexity having an improved dependency on the ambient dimension compared to full space methods. In terms of selecting the subspace dimension, we show that by using matrices based on Johnson-Lindenstrauss transformations, we can choose p to be independent of the ambient dimension n. Our analysis extends to DFO the techniques in [16, 17, 72], and yields similar results to probabilistic direct search [46] and standard model-based DFO [19, 37]. Our results also imply almost-sure global convergence to first-order stationary points.

Our practical implementation of RSDFO-GN, DFBGN, has very low computational requirements: asymptotically, linear in the ambient dimension rather than cubic for standard model-based DFO. After extensive algorithm development described here, our implementation is simple and combines several techniques for modifying the interpolation set which allows it to still make progress with few objective evaluations (an important consideration for DFO techniques). A Python version of DFBGN is available on Github.[17]

For medium-scale problems, DFBGN operating in the full ambient space ($p = n$) has similar performance to DFO-LS [15] when measured by objective evaluations, validating the techniques introduced in the practical implementation. However, DFBGN (with any choice of subspace dimension) has substantially faster runtime, which means it is much more effective than DFO-LS at solving large-scale problems from CUTEst, even when working in a very low-dimensional subspace. Further, in the case of expensive objective evaluations, working a subspace means that DFBGN can make progress with very few evaluations, many fewer than the $n + 1$ needed for standard methods to build their initial model. Overall, the implementation of DFBGN is suitable for large-scale problems both when objective evaluations are cheap (and linear algebra costs dominate) or when evaluations are expensive (and the initialization cost of standard methods is impractical).

Future work will focus on extending the ideas from the implementation DFBGN to the case of general objectives with quadratic models. This will bring the available software in line with the theoretical guarantees for RSDFO. We note that model-based DFO for nonlinear least-squares problems has been adapted to include sketching methods, which use randomization to reduce the number of residuals considered at each iteration [14]. We also delegate to future work the development of techniques for nonlinear least-squares problems which combine sketching (i.e. dimensionality reduction in the observation space) with our subspace approach (i.e. dimensionality reduction in variable space), and further study of methods for adaptively selecting a subspace dimension (c.f. Sect. 4.3).

Acknowledgements The authors would like to acknowledge Zhen Shao for useful discussions on the complexity analysis for RSDFO, two anonymous referees for their valuable feedback, and the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work. (http://dx.doi.org/10.5281/zenodo.22558).

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

A Proofs of technical results

Here we include proofs omitted from the main text.

[17] https://github.com/numericalalgorithmsgroup/dfbgn.

A.1 Proof of Lemma 2

Since $\hat{m}_k$ is Q_k-fully linear and $\Delta_k \le \mu \|\hat{\boldsymbol{g}}_k\|$, the criticality step is not called. From Lemma 1 and $\Delta_k \le \|\hat{\boldsymbol{g}}_k\|/\kappa_H$, we have $\|\hat{\boldsymbol{s}}_k\| \ge c_2 \Delta_k \ge \beta_F \Delta_k$ and so the safety step is not called.

From Assumptions 2 and 3, we have

$$\hat{m}_k(\mathbf{0}) - \hat{m}_k(\hat{\boldsymbol{s}}_k) \ge c_1 \|\hat{\boldsymbol{g}}_k\| \min\left(\Delta_k, \frac{\|\hat{\boldsymbol{g}}_k\|}{\kappa_H}\right) = c_1 \|\hat{\boldsymbol{g}}_k\| \Delta_k, \tag{127}$$

since $\Delta_k \le \|\hat{\boldsymbol{g}}_k\|/\kappa_H$ by assumption. Next, since $\hat{m}_k$ is Q_k-fully linear, from (6a) we have

$$|f(\boldsymbol{x}_k) - \hat{m}_k(\mathbf{0})| = |f(\boldsymbol{x}_k + Q_k \mathbf{0}) - \hat{m}_k(\mathbf{0})| \le \kappa_{\text{ef}} \Delta_k^2, \tag{128}$$

$$|f(\boldsymbol{x}_k + \boldsymbol{s}_k) - \hat{m}_k(\hat{\boldsymbol{s}}_k)| = |f(\boldsymbol{x}_k + Q_k \hat{\boldsymbol{s}}_k) - \hat{m}_k(\hat{\boldsymbol{s}}_k)| \le \kappa_{\text{ef}} \Delta_k^2. \tag{129}$$

Hence we have

$$|\rho_k - 1| \le \frac{|f(\boldsymbol{x}_k) - \hat{m}_k(\mathbf{0})|}{|\hat{m}_k(\mathbf{0}) - \hat{m}_k(\hat{\boldsymbol{s}}_k)|} + \frac{|f(\boldsymbol{x}_k + \boldsymbol{s}_k) - \hat{m}_k(\hat{\boldsymbol{s}}_k)|}{|\hat{m}_k(\mathbf{0}) - \hat{m}_k(\hat{\boldsymbol{s}}_k)|} \le \frac{2\kappa_{\text{ef}} \Delta_k^2}{c_1 \|\hat{\boldsymbol{g}}_k\| \Delta_k} \le 1 - \eta_2, \tag{130}$$

since $\Delta_k \le c_0 \|\hat{\boldsymbol{g}}_k\| \le c_1(1-\eta_2)\|\hat{\boldsymbol{g}}_k\|/(2\kappa_{\text{ef}})$. Thus $\rho_k \ge \eta_2$, which is the claim of the lemma. □

A.2 Proof of Lemma 3

The first part follows immediately from the entry condition of the criticality step. To prove (13), suppose the criticality step is not called in iteration k and $\|\hat{\boldsymbol{g}}_k\| < \epsilon_C$. Then we have $\Delta_k \le \mu \|\hat{\boldsymbol{g}}_k\|$ and $\hat{m}_k$ is Q_k-fully linear, and so from (6b) we have

$$\|Q_k^T \nabla f(\boldsymbol{x}_k)\| \le \|Q_k^T \nabla f(\boldsymbol{x}_k) - \hat{\boldsymbol{g}}_k\| + \|\hat{\boldsymbol{g}}_k\| \le \kappa_{\text{eg}} \Delta_k + \|\hat{\boldsymbol{g}}_k\| \le (\kappa_{\text{eg}} \mu + 1)\|\hat{\boldsymbol{g}}_k\|. \tag{131}$$

Since Q_k is well-aligned, we conclude from (12) and (131) that

$$\alpha_Q \|\nabla f(\boldsymbol{x}_k)\| \le \|Q_k^T \nabla f(\boldsymbol{x}_k)\| \le (\kappa_{\text{eg}} \mu + 1)\|\hat{\boldsymbol{g}}_k\|, \tag{132}$$

and we are done, since $\|\nabla f(\boldsymbol{x}_k)\| \ge \epsilon$. □

A.3 Proof of Lemma 4

Since $\|\nabla f(\boldsymbol{x}_k)\| \geq \epsilon$, from Lemma 3 we have $\|\hat{\boldsymbol{g}}_k\| \geq \epsilon_g(\epsilon)$ for all $k \in \mathcal{A} \cap \mathcal{S}$ (noting that $k \in \mathcal{S}$ implies $k \in \mathcal{C}^C$). Then since $\rho_k \geq \eta_1$, from Assumptions 2 and 3 we get

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq \eta_1[\hat{m}_k(\boldsymbol{0}) - \hat{m}_k(\hat{\boldsymbol{s}}_k)], \tag{133}$$

$$\geq \eta_1 c_1 \|\hat{\boldsymbol{g}}_k\| \min\left(\Delta_k, \frac{\|\hat{\boldsymbol{g}}_k\|}{\kappa_H}\right), \tag{134}$$

$$\geq \eta_1 c_1 \epsilon_g(\epsilon) \min\left(\Delta, \frac{\epsilon_g(\epsilon)}{\kappa_H}\right), \tag{135}$$

where the last line follows from $\|\hat{\boldsymbol{g}}_k\| \geq \epsilon_g(\epsilon)$ and $\Delta_k \geq \Delta$ (from $k \in \mathcal{D}(\Delta)$). Since our step acceptance guarantees our algorithm is monotone (i.e. $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$ for all k), we get

$$f(\boldsymbol{x}_0) - f_{\text{low}} \geq \sum_{k \in \mathcal{A} \cap \mathcal{D}(\Delta) \cap \mathcal{S}} [f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1})]$$

$$\geq \left[\eta_1 c_1 \epsilon_g(\epsilon) \min\left(\Delta, \frac{\epsilon_g(\epsilon)}{\kappa_H}\right)\right] \cdot \#(\mathcal{A} \cap \mathcal{D}(\Delta) \cap \mathcal{S}), \tag{136}$$

from which the result follows. □

A.4 Proof of Lemma 5

To find a contradiction, first suppose $k \in \mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L} \cap \mathcal{C}^C \backslash \mathcal{VS}$. Then since $k \in \mathcal{A} \cap \mathcal{C}^C$ and $\|\nabla f(\boldsymbol{x}_k)\| \geq \epsilon$ by assumption, we have $\|\hat{\boldsymbol{g}}_k\| \geq \epsilon_g(\epsilon)$ from Lemma 3. Since $k \in \mathcal{D}^C(\Delta)$, we have $\Delta_k < \Delta \leq \Delta^*(\epsilon) \leq \min(\mu, 1/\kappa_H)\epsilon_g(\epsilon) \leq \min(\mu, 1/\kappa_H)\|\hat{\boldsymbol{g}}_k\|$ by definition of $\Delta^*(\epsilon)$. Also, since $k \in \mathcal{A} \cap \mathcal{L} \cap \mathcal{D}^C(\Delta)$, we have

$$\alpha_Q \epsilon \leq \|Q_k^T \nabla f(\boldsymbol{x}_k) - \hat{\boldsymbol{g}}_k\| + \|\hat{\boldsymbol{g}}_k\| \leq \kappa_{\text{eg}} \Delta^*(\epsilon) + \|\hat{\boldsymbol{g}}_k\|. \tag{137}$$

If $\Delta^*(\epsilon) > c_1(1 - \eta_2)\|\hat{\boldsymbol{g}}_k\|/(2\kappa_{\text{ef}})$ were to hold, then this would give $\alpha_Q \epsilon \leq \left(\kappa_{\text{eg}} + \frac{2\kappa_{\text{ef}}}{c_1(1-\eta_2)}\right)\Delta^*(\epsilon)$, contradicting the definition of $\Delta^*(\epsilon)$. Hence we must have $\Delta^*(\epsilon) \leq c_1(1 - \eta_2)\|\hat{\boldsymbol{g}}_k\|/(2\kappa_{\text{ef}})$. All together, since $k \in \mathcal{D}^C(\Delta)$, we have $\Delta_k < \Delta \leq \Delta^*(\epsilon) \leq c_0\|\hat{\boldsymbol{g}}_k\|$ by definition of $\Delta^*(\epsilon)$. From this and $k \in \mathcal{L}$, the assumptions of Lemma 2 are met, so $k \notin \mathcal{F}$ and $\rho_k \geq \eta_2$; that is, $k \in \mathcal{VS}$, a contradiction. Hence we have $\#(\mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L} \cap \mathcal{C}^C \backslash \mathcal{VS}) = 0$.

Next, we suppose $k \in \mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L} \cap \mathcal{C}$ and again look for a contradiction. In this case, we have $\Delta_k < \Delta \leq \Delta^*(\epsilon) \leq \alpha_Q \epsilon/(\kappa_{\text{eg}} + \mu^{-1})$, and so from $k \in \mathcal{A} \cap \mathcal{L}$ and $\|\nabla f(\boldsymbol{x}_k)\| \geq \epsilon$ we have

$$\|\hat{\boldsymbol{g}}_k\| \geq \|Q_k^T \nabla f(\boldsymbol{x}_k)\| - \|Q_k^T \nabla f(\boldsymbol{x}_k) - \hat{\boldsymbol{g}}_k\| \geq \alpha_Q \epsilon - \kappa_{\text{eg}} \Delta_k > \mu^{-1}\Delta_k. \tag{138}$$

This means we have $\|\hat{\boldsymbol{g}}_k\| > \mu^{-1}\Delta_k$ and $k \in \mathcal{L}$, so the criticality step is not entered; i.e. $k \in \mathcal{C}^C$, a contradiction. Hence we have $\#(\mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L} \cap \mathcal{C}) = 0$ and we are done. □

A.5 Proof of Lemma 6

If $k \in \mathcal{S}$, we always have $\Delta_{k+1} \le \overline{\gamma}_{\text{inc}}\Delta_k$. On the other hand, if $k \in \mathcal{U}$, we have

$$\Delta_{k+1} = \min(\gamma_{\text{dec}}\Delta_k, \|\hat{\boldsymbol{s}}_k\|) \le \gamma_{\text{dec}}\Delta_k, \tag{139}$$

Hence,

$$\Delta_{k+1} \le \max(\gamma_C, \gamma_F, \gamma_{\text{dec}})\Delta_k, \qquad \text{for all } k \in \mathcal{C} \cup \mathcal{F} \cup \mathcal{U}. \tag{140}$$

We now consider the value of $\log(\Delta_k)$ for $k = 0, \dots, K$, so at each iteration we have an additive change:

- Since $\Delta \le \Delta_0$, the threshold value $\log(\Delta)$ is $\log(\Delta_0/\Delta)$ below the starting value $\log(\Delta_0)$.
- If $k \in \mathcal{S}$, then $\log(\Delta_k)$ increases by at most $\log(\overline{\gamma}_{\text{inc}})$. In particular, $\Delta_{k+1} \ge \Delta$ is only possible if $\Delta_k \ge \overline{\gamma}_{\text{inc}}^{-1}\Delta$.
- If $k \notin \mathcal{S} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{U}$, then $\log(\Delta_k)$ decreases by at least $|\log(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}}))| = \log(1/\max(\gamma_C, \gamma_F, \gamma_{\text{dec}}))$.

Now, any decrease in Δ_k coming from $k \in \mathcal{D}(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\Delta)\backslash\mathcal{S}$ yields $\Delta_{k+1} \ge \Delta$. Hence the total decrease in $\log(\Delta_k)$ must be fully matched by the initial gap $\log(\Delta_0/\Delta)$ plus the maximum possible amount that $\log(\Delta_k)$ can be increased above $\log(\Delta)$. That is, we must have

$$\begin{aligned}\log(1/\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})) \cdot \#(\mathcal{D}(\max(\gamma_C, \gamma_F, \gamma_{\text{dec}})^{-1}\Delta)\backslash\mathcal{S}) \\ \le \log(\Delta_0/\Delta) + \log(\overline{\gamma}_{\text{inc}}) \cdot \#(\mathcal{D}(\overline{\gamma}_{\text{inc}}^{-1}\Delta) \cap \mathcal{S}),\end{aligned} \tag{141}$$

which gives us (18). □

A.6 Proof of Lemma 7

We follow a similar reasoning to the proof of Lemma 6. For every iteration $k \in \mathcal{VS} \cap \mathcal{D}^C(\Delta)$, we increase Δ_k by a factor of at least γ_{inc}, since $\Delta_k < \Delta \le \gamma_{\text{inc}}^{-1}\Delta_{\max}$. Equivalently, we increase $\log(\Delta_k)$ by at least $\log(\gamma_{\text{inc}})$. In particular, if $\Delta_k < \gamma_{\text{inc}}^{-1}\Delta$, then $\Delta_{k+1} < \Delta$.

Alternatively, if $k \in \mathcal{S}\backslash\mathcal{VS}$, we set

$$\Delta_{k+1} = \max(\gamma_{\text{dec}}\Delta_k, \|\hat{\boldsymbol{s}}_k\|) \ge \gamma_{\text{dec}}\Delta_k. \tag{142}$$

If $k \in \mathcal{U}$ we set

$$\Delta_{k+1} = \min(\gamma_{\text{dec}}\Delta_k, \|\hat{\boldsymbol{s}}_k\|) \ge \min(\gamma_{\text{dec}}, \beta_F)\Delta_k, \tag{143}$$

since $\|\hat{\boldsymbol{s}}_k\| \geq \beta_F \Delta_k$ from $k \notin \mathcal{F}$. Hence, for every iteration $k \notin \mathcal{VS}$, we decrease Δ_k by a factor of at most $\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)$, or equivalently we decrease $\log(\Delta_k)$ by at most the amount $|\log(\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F))| = \log(1/\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F))$. Then, to have $\Delta_{k+1} < \Delta$ we require $\Delta_k < \min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-1}\Delta$.

Therefore, since $\Delta_0 \geq \Delta$, the total increase in $\log(\Delta_k)$ from $k \in \mathcal{VS} \cap \mathcal{D}^C(\gamma_{\text{inc}}^{-1}\Delta)$ must be fully matched by the total decrease in $\log(\Delta_k)$ from $k \in \mathcal{D}^C(\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-1}\Delta) \setminus \mathcal{VS}$. That is,

$$\begin{aligned}&\log(\gamma_{\text{inc}})\#(\mathcal{VS} \cap \mathcal{D}^C(\gamma_{\text{inc}}^{-1}\Delta)) \\ &\quad \leq \log(1/\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F))\#(\mathcal{D}^C(\min(\gamma_C, \gamma_F, \gamma_{\text{dec}}, \beta_F)^{-1}\Delta) \setminus \mathcal{VS}),\end{aligned} \tag{144}$$

and we are done. □

A.7 Proof of Lemma 8

After every iteration k where $\hat{m}_k$ is not Q_k-fully linear and either the criticality step is called or $\rho_k < \eta_2$, we always set $\Delta_{k+1} \leq \Delta_k$ and `CHECK_MODEL=TRUE`. This means that $Q_{k+1} = Q_k$, so Q_{k+1} is well-aligned if and only if Q_k is well-aligned. Hence if $k \in \mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L}^C \setminus \mathcal{VS}$ then either $k = K$ or $k + 1 \in \mathcal{A} \cap \mathcal{D}^C(\Delta) \cap \mathcal{L}$, and we are done. □

B Supplementary analysis of DFBGN implementation

In this section we include supplementary analysis and motivation of the DFBGN method, omitted from the main text for brevity.

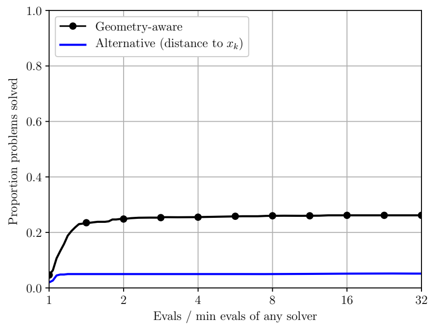
B.1 Alternative point removal mechanism

Instead of Algorithm 4, we could have used a simpler mechanism for removing points, such as removing the points furthest from the current iterate (with total cost[18] $\mathcal{O}(np)$). However, this leads to a substantial performance penalty. In Fig. 9, we compare these two approaches for selecting points to be removed, namely Algorithm 4 and distance to $\boldsymbol{x}_{k+1}$, on the (CR) test set with $p = n/10$ and $p = n$ (using the default value of p_{drop}, as detailed in Sect. 4.5). For more details on the numerical testing framework, see Sect. 5.1 below. We see that the geometry-aware criterion (120) gives substantially better performance than the cheaper criterion.

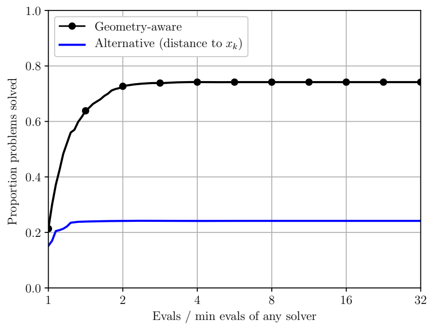
B.2 Choice of p_{drop}: further numerical studies

We note in the main text that there is a trade-off between wanting p_{drop} to be large (to bring in new information) and to be small (to avoid unnecessary objective evaluations).

[18] As above, if we have (117), we could calculate all distances to $\boldsymbol{x}_{k+1}$ using columns of R_k, with total cost $\mathcal{O}(p^2)$.

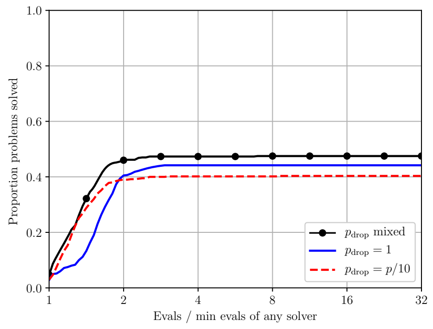


(a) DFBGN with $p = n/10$

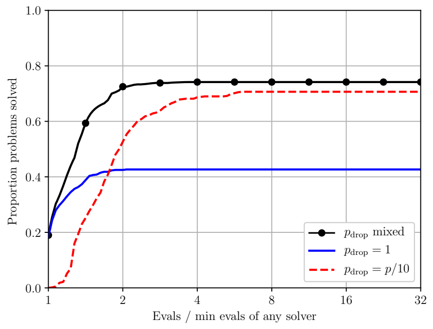


(b) DFBGN with $p = n$

Fig. 9 Performance profiles (in evaluations) for DFBGN when $p = n/10$ and $p = n$, comparing removing points with the geometry-aware Algorithm 4 and without Lagrange polynomials (by distance to the current iterate). We use accuracy level $\tau = 10^{-5}$, and results are an average of 10 runs, each with a budget of $100(n + 1)$ evaluations. The problem collection is (CR). See Sect. 5.1 for details on the testing framework



(a) DFBGN with $p = n/2$



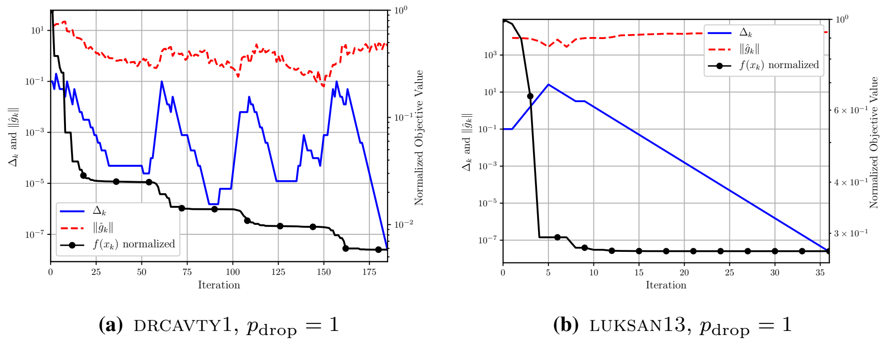
(b) DFBGN with $p = n$

Fig. 10 Performance profiles (in evaluations) comparing different choices of p_{drop}, for DFBGN with $p = n/2$ and $p = n$, with accuracy level $\tau = 10^{-5}$. The choice 'p_{drop} mixed' uses $p_{\text{drop}} = 1$ for successful iterations and $p_{\text{drop}} = p/10$ for unsuccessful iterations. Results an average of 10 runs, each with a budget of $100(n + 1)$ evaluations. The problem collection is (CR). See Sect. 5.1 for details on the testing framework

We consider two choices of p_{drop}, aimed at each of these possible benefits: $p_{\text{drop}} = p/10$ to change subspaces quickly, and $p_{\text{drop}} = 1$ (the minimum possible value) to use few objective evaluations.

Another approach that we consider is a compromise between these two choices. We note that having $p_{\text{drop}} = 1$ is useful to make progress with few evaluations, so we use this value while we are making progress—we consider this to occur when we have a successful iteration (i.e. $\rho_k \geq \eta_1$). When we are not progressing (i.e. unsuccessful steps with $\rho_k < \eta_1$), we use the larger value $p_{\text{drop}} = p/10$.

We compare these three approaches on the (CR) problem collection with a budget of $100(n + 1)$ evaluations in Fig. 10. Since these different choices of p_{drop} are similar when p is small, we show results for subspace dimensions $p = n/2$ and $p = n$. We



(a) DRCAVTY1, $p_{\text{drop}} = 1$ **(b)** LUKSAN13, $p_{\text{drop}} = 1$

Fig. 11 Comparison of Δ_k, $\|\hat{\boldsymbol{g}}_k\|$ (left y-axis) and normalized objective value (right y-axis) for DFBGN with $p = n$ and $p_{\text{drop}} = 1$. The two problems are taken from the (CR) collection. See Sect. 5.1 for details on the testing framework

first see that, even with $p = n/2$, the three approaches all perform similarly. However, for $p = n$ the compromise choice $p_{\text{drop}} \in \{1, p/10\}$ performs better than the two constant-p approaches. In addition, $p_{\text{drop}} = 1$ outperforms $p_{\text{drop}} = p/10$ for small performance ratios, but is less robust and solves fewer problems overall.

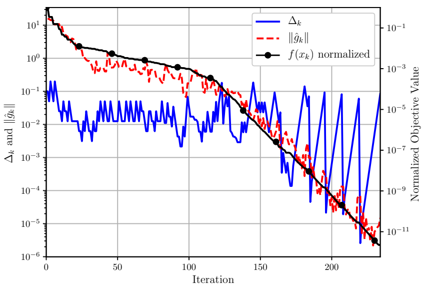
Given these results, in DFBGN we use the compromise choice as the default mechanism: $p_{\text{drop}} = 1$ on successful iterations and $p_{\text{drop}} = p/10$ on unsuccessful iterations.

Relationship to model-improvement phases The `CHECK_MODEL` flag in RSDFO-GN is important for ensuring we do not reduce Δ_k too quickly without first ensuring the quality of the interpolation model.[19] For a similar purpose, DFO-LS incorporates a second trust-region radius which also is involved with ensuring Δ_k does not decrease too quickly [15]. In DFBGN, as described in Sect. 4.4.1, we maintain the geometry of the interpolation set by replacing poorly-located points with orthogonal directions around the current iterate; in practice this ensures the quality of the interpolation set. However, the choice of p_{drop} has a large impact on causing Δ_k to shrink too quickly.

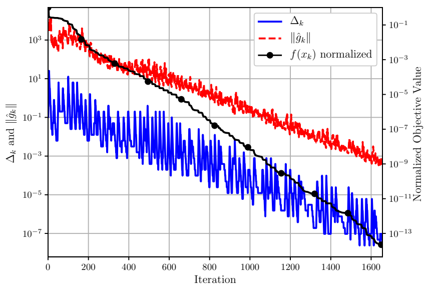
In many cases, DFBGN may reach a point where its model is not accurate and we start to have unsuccessful iterations. To fix this (and continue making progress), we need to introduce several new interpolation points to produce a high-quality model. If p_{drop} is small, this may take many unsuccessful iterations, causing Δ_k to decrease quickly.

The result of having p_{drop} small is seen in Fig. 11. Here, we show Δ_k, $\|\hat{\boldsymbol{g}}_k\|$ and $f(\boldsymbol{x}_k)$ for DFBGN with $p = n$ and $p_{\text{drop}} = 1$ for two problems from the (CR) collection. Both problems show that Δ_k can quickly shrink to be much smaller than $\|\hat{\boldsymbol{g}}_k\|$ before reaching optimality. In the case of DRCAVTY1, we see multiple instances where, after several unsuccessful iterations, we recover a high-quality model and continue making progress (causing Δ_k to increase again); this manifests itself as large oscillations in Δ_k with comparatively little change in $\|\hat{\boldsymbol{g}}_k\|$. Ultimately, as we terminate on $\Delta_k \le \Delta_{\text{end}} = 10^{-8}$, DFBGN quits without solving the problem (reaching accuracy $\tau \approx 6 \times 10^{-3}$). A more extreme version of this behaviour is seen for problem

[19] This is related to ensuring Δ_k does not get too small compared to $\|\hat{\boldsymbol{g}}_k\|$ via Lemma 2.



(a) DRCAVTY1, p_{drop} mixed



(b) LUKSAN13, p_{drop} mixed

Fig. 12 Comparison of Δ_k, $\|\hat{\boldsymbol{g}}_k\|$ (left y-axis) and normalized objective value (right y-axis) for DFBGN with $p = n$ and the default $p_{\text{drop}} \in \{1, p/10\}$. The two problems are taken from the (CR) collection. See Sect. 5.1 for details on the testing framework

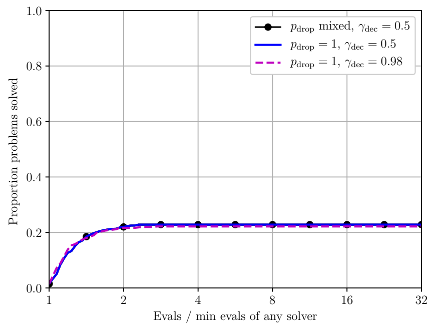
LUKSAN13, where we terminate on small Δ_k in the first sequence of unsuccessful iterations—DFBGN does not allow enough time to recover a high-quality model and terminates after achieving accuracy $\tau \approx 0.3$.

This effect is mitigated by our default choice of $p_{\text{drop}} \in \{1, p/10\}$. By using a larger p_{drop} on unsuccessful iterations, when our model is performing poorly, our interpolation set is changed quickly. This results in DFBGN recovering a high-quality model after a smaller decrease in Δ_k. To demonstrate this, in Fig. 12 we show the results of DFBGN with this p_{drop} for the same problems as Fig. 11 above. In both cases, we still see oscillations in Δ_k, but their magnitude is substantially reduced—it takes fewer iterations to get successful steps, and Δ_k stays well above Δ_{end}. This leads to both problems being solved to high accuracy.

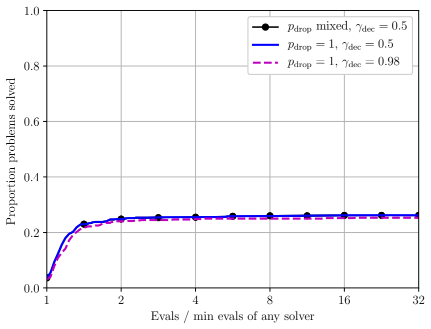
In Fig. 12, we also see that, as we approach the solution, $\|\hat{\boldsymbol{g}}_k\|$ and Δ_k decrease at the same rate, as we would hope. For DRCAVTY1 after iteration 150, we also see the phenomenon described above, where Δ_k can become much larger than $\|\hat{\boldsymbol{g}}_k\|$ due to many successful iterations, before an unsuccessful iteration with $\|\boldsymbol{s}_k\|$ small means that Δ_k returns to the level of $\|\hat{\boldsymbol{g}}_k\|$ regularly.

Alternative Mechanism for Recovering High-Quality Models An alternative approach for avoiding unnecessary decreases in Δ_k while the interpolation model quality is improved is to simply decrease Δ_k more slowly on unsuccessful iterations. This corresponds to setting γ_{dec} to be closer to 1, which is the default choice of DFO-LS for noisy problems (see [15, Section 3.1]), and aligns with our theoretical requirements on the trust-region parameters (Theorem 1).

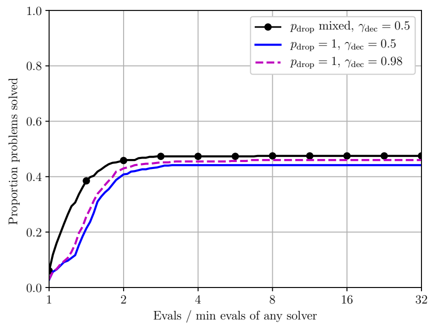
In Fig. 13, we compare the DFBGN default choices, of $p_{\text{drop}} \in \{1, p/10\}$ and $\gamma_{\text{dec}} = 0.5$, with $p_{\text{drop}} = 1$ and $\gamma_{\text{dec}} \in \{0.5, 0.98\}$ on the (CR) problem collection. For small values of p (where the different choices of p_{drop} are essentially identical), the choice of γ_{dec} has almost no impact on the performance of DFBGN. For larger values of p, using $\gamma_{\text{dec}} = 0.98$ with $p_{\text{drop}} = 1$ performs comparably well to the DFBGN default ($\gamma_{\text{dec}} = 0.5$ with $p_{\text{drop}} \in \{1, p/10\}$). However, we opt for keeping $\gamma_{\text{dec}} = 0.5$, to allow us to use the larger value for noisy problems (just as in DFO-LS), and to reduce the risk of overfitting our trust-region parameters to a particular problem collection.



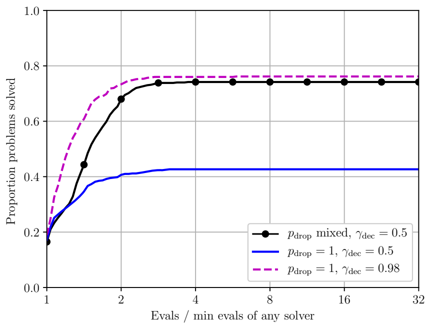
(a) DFBGN with $p = n/100$



(b) DFBGN with $p = n/10$



(c) DFBGN with $p = n/2$



(d) DFBGN with $p = n$

Fig. 13 Performance profiles (in evaluations) comparing different choices of p_{drop} and γ_{dec} for DFBGN, at accuracy level $\tau = 10^{-5}$. The choice 'p_{drop} mixed' uses $p_{\text{drop}} = 1$ for successful iterations and $p_{\text{drop}} = p/10$ for unsuccessful iterations. Results an average of 10 runs, each with a budget of $100(n+1)$ evaluations. The problem collection is (CR). See Sect. 5.1 for details on the testing framework

C Large-scale test problems (CR-large)

See Table 3.

Table 3 Details of large-scale test problems from the CUTEst test set (showing $2f(\boldsymbol{x}_0)$ and $2f(\boldsymbol{x}^*)$ as the implementations of DFO-LS and DFBGN do not have the 1/2 constant factor in (107))

#	Problem	n	m	$2f(\boldsymbol{x}_0)$	$2f(\boldsymbol{x}^*)$	Parameters
1	ARGLALE	2000	4000	10,000	2000	$(N, M) = (2000, 4000)$
2	ARGLBLE	2000	4000	8.545072×10^{22}	999.6250	$(N, M) = (2000, 4000)$
3	ARGTRIG	1000	1000	333.0006	0	$N = 1000$
4	ARTIF	5000	5000	1827.355	0	$N = 5000$
5	ARWHDNE	5000	9998	24,995	1396.793	$N = 5000$
6	BDVALUES	1000	1000	1.996774×10^4	0	$NDP = 1002$
7	BRATU2D	4900	4900	3.085195×10^{-3}	0	$P = 72$
8	BRATU2DT	4900	4900	8.937521×10^{-3}	7.078014×10^{-11}	$P = 72$
9	BRATU3D	3375	3375	2.386977	0	$P = 17$
10	BROWNALE	1000	1000	2.502498×10^8	0	$N = 1000$
11	BROYDN3D	1000	1000	1011	0	$N = 1000$
12	BROYDNBD	5000	5000	124,904	0	$N = 5000$
13	CBRATU2D	2888	2888	1.560446×10^{-2}	0	$P = 40$
14	CHANDHEQ	1000	1000	69.41682	0	$N = 1000$
15	EIGENB	2550	2550	99	0	$N = 50$

Table 3 continued

#	Problem	n	m	$2f(\boldsymbol{x}_0)$	$2f(\boldsymbol{x}^*)$	Parameters
16	FREURONE	5000	9998	5.0485565×10^6	6.081592×10^5	$N = 5000$
17	INTEGREQ	1000	1000	5.678349	0	$N = 1000$
18	MOREBVNE	1000	1000	3.961509×10^{-6}	0	$N = 1000$
19	MSQRTA	4900	4900	7.975592×10^4	0	$P = 70$
20	MSQRTB	1024	1024	7926.444	0	$P = 32$
21	OSCIGRNE	1000	1000	6.120720×10^8	0	$N = 1000$
22	PENLT1NE	1000	1001	1.114448×10^{17}	9.686272×10^{-8}	$N = 1000$
23	POWELLSE	1000	1000	418,750	0	$N = 1000$
24	SEMICN2U	1000	1000	1.960620×10^4	0	$(N, LN) = (1000, 900)$
25	SPMSQRT	1000	1664	797.0033	0	$M = 334$
26	VARDIMNE	1000	1002	1.241994×10^{22}	0	$N = 1000$
27	YATP1SQ	2600	2600	5.184111×10^7	0	$N = 50$
28	YATP2SQ	2600	2600	2.246192×10^7	0	$N = 50$

The set of problems are taken from those in [19, Table 3]; the relevant parameters yielding the given (n, m) are provided. The value of n shown excludes fixed variables

References

1. Alarie, S., Audet, C., Gheribi, A.E., Kokkolaras, M., Le Digabel, S.: Two decades of blackbox optimization applications. EURO J. Comput. Optim. **9**, 100011 (2021)
2. Alzantot, M., Sharma, Y., Chakraborty, S., Zhang, H., Hsieh, C.J., Srivastava, M.B.: GenAttack: practical black-box attacks with gradient-free optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1111–1119. ACM, Prague, Czech Republic (2019)
3. Arter, W., Osojnik, A., Cartis, C., Madho, G., Jones, C., Tobias, S.: Data assimilation approach to analysing systems of ordinary differential equations. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2018)
4. Bandeira, A.S., Scheinberg, K., Vicente, L.N.: Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization. Math. Program. **134**(1), 223–257 (2012)
5. Bandeira, A.S., Scheinberg, K., Vicente, L.N.: Convergence of trust-region methods based on probabilistic models. SIAM J. Optim. **24**(3), 1238–1264 (2014)
6. Bandeira, A.S., van Handel, R.: Sharp nonasymptotic bounds on the norm of random matrices with independent entries. Ann. Probab. **44**(4), 2479–2506 (2016)
7. Berahas, A.S., Bollapragada, R., Nocedal, J.: An investigation of Newton–Sketch and subsampled Newton methods. Optim. Methods Softw. **35**, 661–680 (2020)
8. Berahas, A.S., Cao, L., Choromanski, K., Scheinberg, K.: Linear interpolation gives better gradients than Gaussian smoothing in derivative-free optimization (2019). arXiv:1905.13043
9. Berahas, A.S., Cao, L., Choromanski, K., Scheinberg, K.: A theoretical and empirical comparison of gradient approximations in derivative-free optimization. Found. Comput. Math. **22**, 507–560 (2022)
10. Bergou, E., Gratton, S., Vicente, L.N.: Levenberg–Marquardt methods based on probabilistic gradient models and inexact subproblem solution, with application to data assimilation. SIAM/ASA J. Uncertain. Quantif. **4**(1), 924–951 (2016)
11. Bergou, E.H., Gorbunov, E., Richtárik, P.: Stochastic three points method for unconstrained smooth minimization. SIAM J. Optim. **30**, 2726–2749 (2020)
12. Blanchet, J., Cartis, C., Menickelly, M., Scheinberg, K.: Convergence rate analysis of a stochastic trust region method for nonconvex optimization. INFORMS J. Optim. **1**(2), 92–119 (2019)
13. Boucheron, S., Lugosi, G., Massart, P.: Concentration Inequalities: A Nonasymptotic Theory of Independence. Clarendon Press, Oxford (2012)
14. Cartis, C., Ferguson, T., Roberts, L.: Scalable derivative-free optimization for nonlinear least-squares problems. In: Workshop on “Beyond First-Order Methods in ML Systems” at the 37th International Conference on Machine Learning (2020)
15. Cartis, C., Fiala, J., Marteau, B., Roberts, L.: Improving the flexibility and robustness of model-based derivative-free optimization solvers. ACM Trans. Math. Softw. **45**(3), 32:1-32:41 (2019)
16. Cartis, C., Fowkes, J., Shao, Z.: A randomised subspace Gauss–Newton method for nonlinear least-squares. In: Workshop on “Beyond First-Order Methods in ML Systems” at the 37th International Conference on Machine Learning. Vienna, Austria (2020)
17. Cartis, C., Fowkes, J., Shao, Z.: Randomised subspace methods for non-convex optimization, with applications to nonlinear least-squares. Technical report, University of Oxford (2022)
18. Cartis, C., Massart, E., Otemissov, A.: Constrained global optimization of functions with low effective dimensionality using multiple random embeddings (2020). arXiv:2009.10446
19. Cartis, C., Roberts, L.: A derivative-free Gauss–Newton method. Math. Program. Comput. **11**(4), 631–674 (2019)
20. Cartis, C., Roberts, L.: Scalable subspace methods for derivative-free nonlinear least-squares optimization (2021). arXiv:2102.12016
21. Cartis, C., Roberts, L., Sheridan-Methven, O.: Escaping local minima with local derivative-free methods: a numerical investigation. Optimization (2021)
22. Cartis, C., Scheinberg, K.: Global convergence rate analysis of unconstrained optimization methods based on probabilistic models. Math. Program. **169**(2), 337–375 (2018)
23. Chen, R., Menickelly, M., Scheinberg, K.: Stochastic optimization using a trust-region method and random models. Math. Program. **169**(2), 447–487 (2018)
24. Chen, X., Liu, S., Xu, K., Li, X., Lin, X., Hong, M., Cox, D.: ZO-AdaMM: zeroth-order adaptive momentum method for black-box optimization. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc. (2019)

25. Chung, F., Lu, L.: Connected components in random graphs with given expected degree sequences. Ann. Comb. **6**(2), 125–145 (2002)
26. Colson, B., Toint, P.L.: Optimizing partially separable functions without derivatives. Optim. Methods Softw. **20**(4–5), 493–508 (2005)
27. Conn, A.R., Gould, N.I.M., Toint, P.L.: Trust-Region Methods, MPS-SIAM Series on Optimization, vol. 1. MPS/SIAM, Philadelphia (2000)
28. Conn, A.R., Scheinberg, K., Vicente, L.N.: Geometry of interpolation sets in derivative free optimization. Math. Program. **111**(1–2), 141–172 (2007)
29. Conn, A.R., Scheinberg, K., Vicente, L.N.: Global convergence of general derivative-free trust-region algorithms to first- and second-order critical points. SIAM J. Optim. **20**(1), 387–415 (2009)
30. Conn, A.R., Scheinberg, K., Vicente, L.N.: Introduction to Derivative-Free Optimization, MPS-SIAM Series on Optimization, vol. 8. MPS/SIAM, Philadelphia (2009)
31. Cristofari, A., Rinaldi, F.: A derivative-free method for structured optimization problems. SIAM J. Optim. **31**(2), 1079–1107 (2021)
32. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
33. Eaton, M.L.: Multivariate Statistics: A Vector Space Approach, Lecture Notes-Monograph Series, vol. 53. Institute of Mathematical Statistics, Beachwood (2007)
34. Ehrhardt, M.J., Roberts, L.: Inexact derivative-free optimization for bilevel learning. J. Math. Imaging Vis. **63**(5), 580–600 (2020)
35. Ergen, T., Candès, E., Pilanci, M.: Random projections for learning non-convex models. In: 33rd Conference on Neural Information Processing Systems (2019)
36. Facchinei, F., Scutari, G., Sagratella, S.: Parallel selective algorithms for nonconvex big data optimization. IEEE Trans. Signal Process. **63**(7), 1874–1889 (2015)
37. Garmanjani, R., Júdice, D., Vicente, L.N.: Trust-region methods without using derivatives: worst case complexity and the nonsmooth case. SIAM J. Optim. **26**(4), 1987–2011 (2016)
38. Ghadimi, S., Lan, G.: Stochastic first- and zeroth-order methods for nonconvex stochastic programming. SIAM J. Optim. **23**(4), 2341–2368 (2013)
39. Ghanbari, H., Scheinberg, K.: Black-box optimization in machine learning with trust region based derivative free algorithm (2017). arXiv:1703.06925
40. Golovin, D., Karro, J., Kochanski, G., Lee, C., Song, X., Zhang, Q.: Gradientless descent: high-dimensional zeroth-order optimization (2019). arXiv:1911.06317
41. Golub, G.H., Van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
42. Gould, N.I.M., Orban, D., Toint, P.L.: CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Comput. Optim. Appl. **60**(3), 545–557 (2015)
43. Gower, R., Goldfarb, D., Richtárik, P.: Stochastic block BFGS: squeezing more curvature out of data. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 48, pp. 1869–1878. PMLR, New York (2016)
44. Gower, R.M., Kovalev, D., Lieder, F., Richtárik, P.: RSN: randomized subspace Newton. In: 33rd Conference on Neural Information Processing Systems (2019)
45. Gower, R.M., Richtárik, P., Bach, F.: Stochastic quasi-gradient methods: variance reduction via Jacobian sketching. Math. Program. **188**, 135–192 (2020)
46. Gratton, S., Royer, C.W., Vicente, L.N., Zhang, Z.: Direct search based on probabilistic descent. SIAM J. Optim. **25**(3), 1515–1541 (2015)
47. Gratton, S., Royer, C.W., Vicente, L.N., Zhang, Z.: Complexity and global rates of trust-region methods based on probabilistic models. IMA J. Numer. Anal. **38**(3), 1579–1597 (2017)
48. Gratton, S., Royer, C.W., Vicente, L.N., Zhang, Z.: Direct search based on probabilistic feasible descent for bound and linearly constrained problems. Comput. Optim. Appl. **72**(3), 525–559 (2019)
49. Gross, J.C., Parks, G.T.: Optimization by moving ridge functions: derivative-free optimization for computationally intensive functions. Eng. Optim. **54**, 553–575 (2021)
50. Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. **53**(2), 217–288 (2011)
51. Hare, W., Jarry-Bolduc, G., Planiden, C.: Error bounds for overdetermined and underdetermined generalized centred simplex gradients. IMA J. Numer. Anal. **42**(1), 744–770 (2022)
52. Kane, D.M., Nelson, J.: Sparser Johnson–Lindenstrauss transforms. J. ACM **61**(1), 4:1-4:23 (2014)

53. Kelley, C.T.: Detection and remediation of stagnation in the Nelder–Mead algorithm using a sufficient decrease condition. SIAM J. Optim. **10**(1), 43–55 (1999)
54. Kozak, D., Becker, S., Doostan, A., Tenorio, L.: A stochastic subspace approach to gradient-free optimization in high dimensions. Comput. Optim. Appl. **79**(2), 339–368 (2021)
55. Larson, J.W., Menickelly, M., Wild, S.M.: Derivative-free optimization methods. Acta Numer. **28**, 287–404 (2019)
56. Liu, S., Kailkhura, B., Chen, P.Y., Ting, P., Chang, S., Amini, L.: Zeroth-order stochastic variance reduction for nonconvex optimization (2018). arXiv:1805.10367
57. Lu, Z., Xiao, L.: A randomized nonmonotone block proximal gradient method for a class of structured nonlinear programming. SIAM J. Numer. Anal. **55**(6), 2930–2955 (2017)
58. Mahoney, M.W.: Randomized algorithms for matrices and data. Found. Trends Mach. Learn. **3**(2), 123–224 (2011)
59. Moré, J.J., Wild, S.M.: Benchmarking derivative-free optimization algorithms. SIAM J. Optim. **20**(1), 172–191 (2009)
60. Nesterov, Y., Spokoiny, V.: Random gradient-free minimization of convex functions. Found. Comput. Math. **17**(2), 527–566 (2017)
61. Neumaier, A., Fendl, H., Schilly, H., Leitner, T.: VXQR: derivative-free unconstrained optimization based on QR factorizations. Soft Comput. **15**(11), 2287–2298 (2011)
62. Patrascu, A., Necoara, I.: Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. J. Glob. Optim. **61**(1), 19–46 (2015)
63. Pilanci, M., Wainwright, M.J.: Newton sketch: a linear-time optimization algorithm with linear-quadratic convergence. SIAM J. Optim. **27**(1), 205–245 (2017)
64. Porcelli, M., Toint, P.L.: Global and local information in structured derivative free optimization with BFO (2020). arXiv:2001.04801
65. Powell, M.J.D.: On trust region methods for unconstrained minimization without derivatives. Math. Program. **97**(3), 605–623 (2003)
66. Powell, M.J.D.: Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. Math. Program. **100**(1), 183–215 (2004)
67. Powell, M.J.D.: The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report DAMTP 2009/NA06, University of Cambridge (2009)
68. Qian, H., Hu, Y.Q., Yu, Y.: Derivative-free optimization of high-dimensional non-convex functions by sequential random embeddings. In: Kambhampati, S. (ed.) Proceedings of the 25th International Joint Conference on Artificial Intelligence, pp. 1946–1952. AAAI Press, New York (2016)
69. Roberts, L.: Derivative-free algorithms for nonlinear optimisation problems. Ph.D. thesis, University of Oxford (2019)
70. Roosta-Khorasani, F., Mahoney, M.W.: Sub-sampled Newton methods. Math. Program. **174**(1–2), 293–326 (2019)
71. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning (2017). arXiv:1703.03864
72. Shao, Z.: On random embeddings and their applications to optimization. Ph.D. thesis, University of Oxford (2022)
73. Tao, T.: Topics in Random Matrix Theory, Graduate Studies in Mathematics, vol. 132. American Mathematical Society, Providence (2012)
74. Tett, S.F.B., Yamazaki, K., Mineter, M.J., Cartis, C., Eizenberg, N.: Calibrating climate models using inverse methods: case studies with HadAM3, HadAM3P and HadCM3. Geosci. Model Dev. **10**, 3567–3589 (2017)
75. Ughi, G., Abrol, V., Tanner, J.: A model-based derivative-free approach to black-box adversarial examples: Bobyqa. In: Workshop on “Beyond First-Order Methods in ML” at the 32nd Conference on Advances in Neural Information Processing Systems (2019)
76. Vicente, L.N.: Worst case complexity of direct search. EURO J. Comput. Optim. **1**(1–2), 143–153 (2013)
77. Vicente, L.N.: Direct search based on probabilistic descent. Seminar slides provided by private communication (2014)
78. Wang, Z., Hutter, F., Zoghi, M., Matheson, D., de Freitas, N.: Bayesian optimization in a billion dimensions via random embeddings. J. Artif. Intell. Res. **55**(1), 361–387 (2016)
79. Wild, S.M.: POUNDERS in TAO: solving derivative-free nonlinear least-squares problems with POUNDERS. In: Terlaky, T., Anjos, M.F., Ahmed, S. (eds.) Advances and Trends in Optimization

with Engineering Applications, MOS-SIAM Book Series on Optimization, vol. 24, pp. 529–539. MOS/SIAM, Philadelphia (2017)
80. Woodruff, D.P.: Sketching as a tool for numerical linear algebra. Found. Trends Theoret. Comput. Sci. **10**(1–2), 1–157 (2014)
81. Wright, S.J.: Coordinate descent algorithms. Math. Program. **151**(1), 3–34 (2015)
82. Xu, Y., Yin, W.: Block stochastic gradient iteration for convex and nonconvex optimization. SIAM J. Optim. **25**(3), 1686–1716 (2015)
83. Xu, Y., Yin, W.: A globally convergent algorithm for nonconvex optimization based on block coordinate update. J. Sci. Comput. **72**(2), 700–734 (2017)
84. Yang, Y., Pesavento, M., Luo, Z.Q., Ottersten, B.: Inexact block coordinate descent algorithms for nonsmooth nonconvex optimization. IEEE Trans. Signal Process. **68**, 947–961 (2020)
85. Zhang, H., Conn, A.R., Scheinberg, K.: A derivative-free algorithm for least-squares minimization. SIAM J. Optim. **20**(6), 3555–3576 (2010)
86. Zhang, Z.: A subspace decomposition framework for nonlinear optimization: global convergence and global rate (2013). https://www.zhangzk.net/docs/talks/20130912-icnonla-subdcp.pdf. Accessed 26 Oct 2021