Scalable Derivative-Free Optimization Algorithms with Low-Dimensional Subspace Techniques *

Zaikun Zhang †

January 8, 2025

Abstract

We re-introduce a derivative-free subspace optimization framework originating from Chapter 5 the Ph.D. thesis [Z. Zhang, *On Derivative-Free Optimization Methods, Ph.D. thesis, Chinese Academy of Sciences, Beijing, 2012*] of the author under the supervision of Ya-xiang Yuan. At each iteration, the framework defines a (low-dimensional) subspace based on an approximate gradient, and then solves a subproblem in this subspace to generate a new iterate. We sketch the global convergence and worst-case complexity analysis of the framework, elaborate on its implementation, and present some numerical results on solving problems with dimension as high as 10^4 using only inaccurate function values.

1 Introduction

Consider the unconstrained problem

$$\min\{f(x) : x \in \mathbb{R}^n\},$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth yet possibly nonconvex function.

We make the following assumption on f and will not repeat it in the sequel.

Assumption 1.1. *The function $f : \mathbb{R} \to \mathbb{R}$ is bounded from below and differentiable, and its gradient ∇f is Lipschitz continuous on $\mathbb{R}^n$ with a Lipschitz constant $L \in (0, \infty)$.*

*This is only a **draft containing quick notes on the main ideas** of the work. The final paper will be **very different**. The theory is only sketched without detailed proofs. The numerical results may change as the software package is still under development.

†Email: zaikunzhang@gmail.com.

We define $f_* = \inf\{f(x) : x \in \mathbb{R}^n\}$. For any sequence $\{x_k\} \subset \mathbb{R}^n$, we denote $f_k = f(x_k)$ and $g_k = \nabla f(x_k)$ for each k.

We focus on derivative-free optimization (DFO) algorithms for problem (1). Our target is to solve this problem with n as large as 10^4. We rely on low-dimensional subspace techniques to be detailed in the sequel.

2 A subspace framework for optimization

Algorithm 2.1 presents the iterated-subspace optimization framework by Conn et al. [4]. Step 1 of the algorithm chooses a subspace $\mathcal{S}_k$ to explore. When $\dim(\mathcal{S}_k) \equiv 1$, the algorithm reduces to a line search method. Step 2 sets x_{k+1} to an approximate solution to the subspace subproblem, which is generally much easier to tackle than the original full-space problem due to the low dimensionality. Despite writing "argmin", we do not require this subproblem to have a unique global minimizer.

Algorithm 2.1 **Optim**ization with **I**terated-**S**ubspace **T**echnique (**OptimIST**)

Input: $f : \mathbb{R}^n \to \mathbb{R}$, $x_0 \in \mathbb{R}^n$.

For $k = 0, 1, 2, \ldots$, iterate the following.

1. Choose a subspace $\mathcal{S}_k \subset \mathbb{R}^n$.
2. Calculate $x_{k+1} \approx \operatorname{argmin}\{f(x) : x \in x_k + \mathcal{S}_k\}$.

Algorithm 2.1 is conceptual. For its global convergence, we need to impose some conditions on the subspaces $\{\mathcal{S}_k\}$ and the subproblem solutions $\{x_{k+1}\}$. Proposition 2.1 provides a necessary and sufficient condition when the objective function is convex with bounded level sets.

Proposition 2.1. *Suppose that f is convex with bounded level sets. If Algorithm 2.1 ensures $f_{k+1} \le f_k$ for each $k \ge 0$, then $f_k \to f_*$ if and only if*

$$\operatorname{dist}(g_k, \mathcal{S}_k) \;\to\; 0 \quad \text{and} \quad f_{k+1} - \inf\{f(x) : x \in x_k + \mathcal{S}_k\} \;\to\; 0, \quad k \to \infty. \tag{2.1}$$

Proof. (a) The "if" part. Let P_k be the orthogonal projection onto $\mathcal{S}_k$. Then we have

$$\begin{aligned} \frac{1}{2L}\|P_k g_k\|^2 \;&\le\; f_k - \inf\{f(x) : x \in x_k + \mathcal{S}_k\} \\ &=\; (f_k - f_{k+1}) + [f_{k+1} - \inf\{f(x) : x \in x_k + \mathcal{S}_k\}] \;\to\; 0, \end{aligned}$$

where $f_k - f_{k+1} \to 0$ due to the monotonicity and boundedness of $\{f_k\}$. Therefore, $\|g_k\| \le \operatorname{dist}(g_k, \mathcal{S}_k) + \|P_k g_k\| \to 0$. This implies that $f_k \to f_*$ since f is convex with bounded level sets.

(b) The "only if" part. If $f_k \to f_*$, then $\|g_k\| \to 0$ by the Lipschitz continuity of ∇f. Hence $\operatorname{dist}(g_k, \mathcal{S}_k) \leq \operatorname{dist}(g_k, 0) \to 0$. Meanwhile,

$$f_{k+1} - \inf\{f(x) : x \in x_k + \mathcal{S}_k\} \;\leq\; f_{k+1} - f_* \to 0. \qquad \square$$

According to the "if" part of the above proof, even without the convexity or lower-boundedness of f, condition (2.1) still renders $\|g_k\| \to 0$; if this condition holds only for k in an infinite subset of $\mathbb{N}$, then we will have $\liminf_k \|g_k\| = 0$ instead.

Although Proposition 2.1 is mainly of theoretical interest, it suggests a general strategy to implement Algorithm 2.1:

(a) choose a subspace $\mathcal{S}_k$ that contains a vector $\hat{g}_k \approx g_k$;

(b) set x_{k+1} to a sufficiently accurate solution to $\min\{f(x) : x \in x_k + \mathcal{S}_k\}$.

The construction of $\mathcal{S}_k$ essentially demands an approximate gradient at x_k. The calculation of x_{k+1} needs more elaboration, especially if the problem is nonconvex. If we can implement this strategy without using derivatives, then we will get a DFO algorithm. This will be the focus of Section 3.

3 A subspace framework for derivative-free optimization

Algorithm 3.1 is a specialization of Algorithm 2.1. As we will see, it can be implemented without using derivatives, providing the framework for a class of subspace DFO algorithms.

Algorithm 3.1 Derivative-free OptimIST

Input: $f : \mathbb{R}^n \to \mathbb{R}$, $x_0 \in \mathbb{R}^n$, $\delta_0 \in (0, \infty)$, $\eta \in (0, \infty)$.

For $k = 0, 1, 2, \ldots$, iterate the following.

1. Generate an approximate gradient $\hat{g}_k$ for f at x_k.
2. Choose a subspace $\mathcal{S}_k \subset \mathbb{R}^n$ with $\hat{g}_k \in \mathcal{S}_k$.
3. Calculate $x_{k+1} \approx \operatorname{argmin}\{f(x) : x \in x_k + \mathcal{S}_k\}$ so that $f_{k+1} \leq f_k$ and

$$f_{k+1} \;\leq\; \mathbf{max}\left\{f_k - \eta\delta_k^2,\;\; f(x_k - \delta_k \hat{g}_k/\|\hat{g}_k\|)\right\}. \tag{3.1}$$

4. If $\|\hat{g}_k\| \geq \eta\delta_k$ and $f_{k+1} \leq f_k - \eta\delta_k^2$, then $\delta_{k+1} = 2\delta_k$; otherwise, $\delta_{k+1} = \delta_k/2$.

If $\hat{g}_k = 0$, then we define $\hat{g}_k/\|\hat{g}_k\| = 0$ in Step 3 of Algorithm 3.1. The adjustment of δ_k in Step 4 resembles that of the trust region radius in [1, 8]. It can be generalized to update δ_k in a more refined fashion, but this simple scheme is sufficient for our discussion.

The **key idea** of Algorithm 3.1 is simple: use the local information to discover a subspace and then explore it. In contrast, model-based trust-region methods use the local information to generate a single trial point in the full space and then move on, which may not exploit the information sufficiently.

3.1 Global convergence and worst-case complexity

We sketch the theoretical analysis of Algorithm 3.1. To this end, we propose the following assumption on the approximate gradients $\{\hat{g}_k\}$. We will elaborate on how to guarantee this assumption in Subsection 3.2

Assumption 3.1. *There exists a constant $\zeta > 0$ such that*

$$\|\hat{g}_k - g_k\| \;\le\; \zeta\delta_k \quad \textit{for each} \quad k \ge 0. \tag{3.2}$$

Now we explain why Algorithm 3.1 should be globally convergent with provable worst-case complexity bounds. We will use our proof techniques in [8, 9]. Define

$$\mathcal{K} \;=\; \left\{k \in \mathbb{N} : \|\hat{g}_k\| \ge \eta\delta_k \;\text{ and }\; f_{k+1} \le f_k - \eta\delta_k^2\right\}. \tag{3.3}$$

Let $\mu = 2/(L+2\eta+4\zeta)$. By the triangle inequality and Taylor expansion, we can establish

$$\begin{cases} \delta_k \le \mu\|g_k\|, \\ \|\hat{g}_k - g_k\| \le \zeta\delta_k \end{cases} \quad\Longrightarrow\quad \begin{cases} \|\hat{g}_k\| \ge \eta\delta_k, \\ f(x_k - \delta_k\hat{g}_k/\|\hat{g}_k\|) \le f_k - \eta\delta_k^2\,. \end{cases} \tag{3.4}$$

Consequently, whenever $\delta_k \le \mu\|g_k\|$, we will have $k \in \mathcal{K}$ according to (3.1) and (3.2), and hence $\delta_{k+1} = 2\delta_k$. Then, as in the analysis of trust-region methods, we can obtain

$$\delta_k \;\ge\; \nu\|\tilde{g}_k\| \quad \text{with} \quad \nu = \min\{\delta_0/\|g_0\|,\; \mu/2\} \quad \text{and} \quad \|\tilde{g}_k\| = \min_{0\le\ell\le k}\|g_\ell\|.$$

Hence we establish a lower bound for the reduction achieved by the iterations in $\mathcal{K}$, namely

$$f(x_k) - f_{k+1} \;\ge\; \eta\delta_k^2 \;\ge\; \eta\nu^2\|\tilde{g}_k\|^2 \quad \text{for each} \quad k \in \mathcal{K}. \tag{3.5}$$

Now let $K_\epsilon = \min\{k \in \mathbb{N} : \|g_k\| \le \epsilon\}$ for any given $\epsilon > 0$. Based on (3.5), we can demonstrate that $K_\epsilon = \mathcal{O}(\nu^{-2}\epsilon^{-2})$ in the general nonconvex case, $K_\epsilon = \mathcal{O}(\nu^{-2}\epsilon^{-1})$ if f is convex, and $K_\epsilon = \mathcal{O}(\nu^{-2}|\log\epsilon|)$ if f is strongly convex, using the techniques in [8, 9]. These bounds also imply the global convergence of Algorithm 3.1.

Note that the convergence of Algorithm 3.1 is essentially guaranteed by (3.1) and (3.2). The subspace $\mathcal{S}_k$ does not play a direct role in the convergence analysis. However, the choice of $\mathcal{S}_k$ is crucial for the practical performance of the algorithm.

3.2 Defining the subspace without using derivatives

How to generate the approximate gradient $\hat{g}_k$ and ensure (3.2)? We can construct a model $\hat{f}_k$ for f in $\mathbb{R}^n$ around x_k, and then take $\hat{g}_k = \nabla \hat{f}_k(x_k)$. Condition (3.2) holds if $\hat{f}_k$ is a fully linear model [3] of f in a ball centered at x_k with a radius proportional to δ_k. Such an $\hat{f}_k$ can be obtained by linear or (underdetermined) quadratic Lagrange interpolation of f on a well-poised interpolation set $\mathcal{Y}_k$ in this ball [3].

As an illustration, consider

$$\mathcal{Y}_k = \{x_k\} \cup \{x_k + \tau \delta_k e^i : i = 1, ..., n\}, \tag{3.6}$$

with e^i being the ith coordinate vector and τ being a positive constant. In this case, the linear interpolation of f on $\mathcal{Y}_k$ is equivalent to a forward finite difference, and it will provide a $\hat{g}_k$ satisfying (3.2) with $\zeta = \tau\sqrt{n}L/2$. Note that the function evaluations over $\mathcal{Y}_k$ can be done in parallel.

We may include some previously evaluated points into $\mathcal{Y}_k$ as long as they do not deteriorate the geometry of the interpolation set. However, if Algorithm 3.1 converges fast enough, it is affordable to define $\mathcal{Y}_k$ like in (3.6) without reusing previous points at all.

Algorithm 3.1 has the advantage that it converges regardless of the precise definition of $\mathcal{S}_k$ provided that $\hat{g}_k \in \mathcal{S}_k$. This flexibility allows us to explore and compare different possibilities of $\mathcal{S}_k$ under a unified framework. In general, we will choose $\mathcal{S}_k$ according to the following principles.

(a) $\mathcal{S}_k$ should include directions along which f is likely to decrease.

(b) $\mathcal{S}_k$ should have a dimension much lower than n.

As examples, a few possible configurations of $\mathcal{S}_k$ are listed below.

(a) Conjugate-gradient subspace:

$$\mathcal{S}_k = \text{span}\{\hat{g}_k,\, x_k - x_{k-1}\}. \tag{3.7}$$

This subspace is inspired by Yuan and Stoer's subspace perspective of conjugate-gradient methods [18]. It is also studied by [4] and very recently by [19] in the gradient-based case.

(b) Limited memory quasi-Newton subspace:

$$\mathcal{S}_k \;=\; \text{span}\{\hat{g}_k,\, y_{k-1},\, ...,\, y_{k-m},\, s_{k-1},\, ...,\, s_{k-m}\}, \tag{3.8}$$

where $y_\ell = \hat{g}_{\ell+1} - \hat{g}_\ell$, $s_\ell = x_{\ell+1} - x_\ell$, and $m \geq 1$ is an integer much less than n. When $\hat{g}_k = g_k$, this subspace is discussed by Yuan [17], and it is motivated by the fact that limited memory quasi-Newton methods [13] produce a step in this subspace.

Furthermore, we can augment the subspaces by adding directions reflecting second-order information of the problem if we can obtain an approximate Hessian $\hat{H}_k$ without using derivatives. For instance, we can take $\hat{H}_k = \nabla^2 \hat{f}_k(x_k)$, where $\hat{f}_k$ is the model that generates $\hat{g}_k$, provided that $\hat{f}_k$ is nonlinear. Alternatively, we can establish $\hat{H}_k$ or $\hat{H}_k^{-1}$ by quasi-Newton formulae using the vectors $\{y_\ell, s_\ell\}$ mentioned earlier. We can also explore the methods of Hessian approximation proposed by Hare et al. in [11]. Given an approximate Hessian $\hat{H}_k$, we can include the following directions into $\mathcal{S}_k$.

(a) Approximate Newton direction, namely a direction d such that $\hat{H}_k d \approx -\hat{g}_k$.

(b) Approximate negative-curvature directions. It is known that algorithms can benefit from exploring negative-curvature directions even based on inexact information [6, 12, 16]. If $\hat{H}_k$ is not positive semidefinite, we can take eigenvectors of $\hat{H}_k$ associated with negative eigenvalues as approximate negative-curvature directions.

Recall that the conjugate-gradient method is known to be inefficient for ill-conditioned problems. If we consider the conjugate-gradient subspace (3.7), it is particularly important to augment the space to include the approximate Newton direction. Otherwise, Algorithm 3.1 will converge slowly when the problem is ill-conditioned.

3.3 Solving the subspace subproblem without using derivatives

How to calculate $x_{k+1} \approx \operatorname{argmin}\{f(x) : x \in x_k + \mathcal{S}_k\}$ and ensure (3.1)? We propose the following strategy.

(a) Invoke a DFO solver to solve approximately the low-dimensional subproblem

$$\min\{f(x) : x \in x_k + \mathcal{S}_k\}, \tag{3.9}$$

obtaining an approximate solution x_k^{s}.

(b) If $f(x_k^{\mathrm{s}}) \leq f_k - \eta\delta_k^2$, then set $x_{k+1} = x_k^{\mathrm{s}}$; otherwise, evaluate f at $x_k^{\mathrm{g}} = x_k - \delta_k \hat{g}_k / \|\hat{g}_k\|$, and set x_{k+1} to the point with the smallest function value in $\{x_k, x_k^{\mathrm{s}}, x_k^{\mathrm{g}}\}$.

Note that our strategy does not impose any requirement on the quality of x_k^{s}. Indeed, the purpose of x_k^{s} is to explore the subspace $\mathcal{S}_k$, and that of x_k^{g} is to provide a

safeguard when the exploration fails. It is also worth mentioning that problem (3.9) is an unconstrained derivative-free optimization problem and can be handled by, for example, NEWUOA [15]. Take $\mathcal{S}_k = \text{span}\{\hat{g}_k, x_k - x_{k-1}\}$ as an example. Problem (3.9) is equivalent to

$$\min\{f(x_k + \alpha\hat{g}_k + \beta(x_k - x_{k-1}) : \alpha, \beta \in \mathbb{R}\},$$

a low-dimensional unconstrained problem.

3.4 Evaluation complexity and scalability

We now examine the complexity of Algorithm 3.1 in terms of function evaluations, paying particular attention to its dependence on n. Take the nonconvex case as an example, and suppose that the interpolation set $\mathcal{Y}_k$ is $\{x_k\} \cup \{x_k + \tau\delta_k e^i : i = 1, ..., n\}$. As speculated above, the worst-case iteration complexity is $K_\epsilon = \mathcal{O}(\nu^{-2}\epsilon^{-2})$. If $\tau = \mathcal{O}(n^{-p})$, then $\zeta = \mathcal{O}(n^{\frac{1}{2}-p})$, and hence $\nu^{-1} = \mathcal{O}(\mu^{-1}) = \mathcal{O}(L + 2\eta + 4\zeta) = \mathcal{O}(\max\{1, n^{\frac{1}{2}-p}\})$. Thus $K_\epsilon = \mathcal{O}(n^{1-2p}\epsilon^{-2})$ when $p \in [0, 1/2]$. Since (3.9) is low dimensional and we have no requirement on x_k^s, we can allocate $\mathcal{O}(1)$ function evaluations to Step 3 for solving (3.9). Then the complexity of function evaluations will be $K_\epsilon^f = \mathcal{O}(nK_\epsilon) = \mathcal{O}(n^{2-2p}\epsilon^{-2})$. In particular, $K_\epsilon^f = \mathcal{O}(n\epsilon^{-2})$ if we take $p = 1/2$. This linear dependence on n ensures the scalability of Algorithm 3.1 for the targeted problems, where n is in the order of $10^3 \sim 10^4$. In contrast, the evaluation complexity of standard full-space trust-region DFO methods is $\mathcal{O}(n^2\epsilon^{-2})$, which is also the case for the RSDFO method of Cartis and Roberts [2].

4 SPRIMA: A package based on OptimIST and PRIMA

4.1 Historical remarks

The thesis [20] implemented Algorithm 3.1 by solving the subproblem (3.9) using Powell's NEWUOA [15], leading to the NEWUOAs method. The MATLAB implementation of NEWUOAs was ported to Module-3 in 2016 by Dr. M. Nyström (Principle Engineer at the Intel Corporation) and made available in the open-source package CM3.[1] It has been used by Intel in the design of chips, including its flagship product Atom P5900. It also leads to the BBGP-sDFO method for high-dimensional analog circuit synthesis [10].

We will extend NEWUOAs to develop a new package named SPRIMA, where the subproblem (3.9) is solved by the solvers in PRIMA [21].

[1] https://github.com/modula3/cm3/blob/master/caltech-other/newuoa/src/NewUOAs.m3

4.2 Numerical experiments

Here we present some numerical experiments based on NEWUOAs.[2]

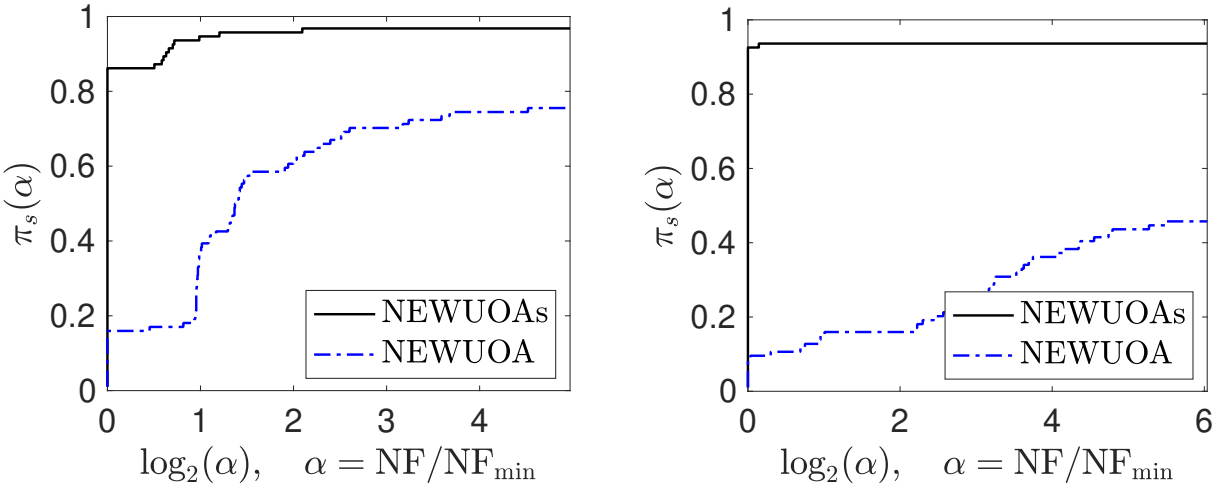
4.2.1 Comparing NEWUOAs and NEWUOA on moderate-dimensional problems

We tested NEWUOAs and NEWUOA on 98 unconstrained CUTEst [7] problems listed in Table 1. All the problems have changeable dimensions, which was to $n = 200$ in this experiment. We provided the solvers with only the first 3 significant digits of the function values, intending to test the robustness of the algorithms. Figure 1 shows the performance profiles [5, 14] generated with the tolerance of the convergence test set to 10^{-1} and 10^{-3}. It is evident that NEWUOAs outperformed NEWUOA in this experiment.

arglina	arglina4	arglinb	arglinc	argtrig	arwhead
bdqrtic	bdqrticp	bdvalue	biggsb1	brownal	broydn3d
broydn7d	brybnd	chainwoo	chebquad	chnrosnb	chpowellb
chpowells	chrosen	cosine	cragglvy	cube	curly10
curly20	curly30	dixmaane	dixmaanf	dixmaang	dixmaanh
dixmaani	dixmaanj	dixmaank	dixmaanl	dixmaanm	dixmaann
dixmaano	dixmaanp	dqrtic	edensch	eg2	engval1
errinros	expsum	extrosnb	exttet	firose	fletcbv2
fletcbv3	fletchcr	fminsrf2	freuroth	genbrown	genhumps
genrose	indef	integreq	liarwhd	lilifun3	lilifun4
morebv	morebvl	ncb20	ncb20b	noncvxu2	noncvxun
nondia	nondquar	penalty1	penalty2	penalty3	penalty3p
powellsg	power	rosenbrock	sbrybnd	sbrybndl	schmvett
scosine	scosinel	serose	sinquad	sparsine	sparsqur
sphrpts	spmsrtls	srosenbr	stmod	tointgss	tointtrig
tquartic	tridia	trigsabs	trigssqs	trirose1	trirose2
vardim	woods				

Table 1: 98 CUTEst problems with changeable dimensions

[2] https://github.com/newuoas/newuoas



(a) Tolerance of convergence test: 10^{-1} (b) Tolerance of convergence test: 10^{-3}

Horizontal: relative cost in $\log_2$-scale; vertical: percentage of problems solved

Figure 1: Performance Profiles of NEWUOAs and NEWUOA
(function values were truncated to 3 significant digits; dimension $n = 200$)

4.2.2 Comparing NEWUOAs and fminunc on large-dimensional problems

To illustrate the scalability of NEWUOAs, we compared it with fminunc on 12 problems with 10^4 variables. The latter is a solver provided by MATLAB's Optimization Toolbox, and it uses a BFGS method based on finite-difference gradients when derivatives are unavailable. We did not test NEWUOA in this case because it cannot scale to such high dimensions. In this experiment, we still provided the solvers with only the first 3 significant digits. The results are summarized in Table 2. Indeed, fminunc always stopped prematurely on these problems, while NEWUOAs provided reasonable solutions.

Although we have no intention to claim that NEWUOAs can solve 10^4-dimensional problems in general, the results in Table 2 demonstrate its potential for high-dimensional problems. This illustrates the power of the subspace strategy in scaling up DFO methods.

Table 2: Comparison between NEWUOAs and fminunc on 10^4-dimensional problems (function values were truncated to 3 significant digits)

		NEWUOAs		fminunc	
Problem	$f(x_0)$	$f(x_{\text{fin}})$	NF	$f(x_{\text{fin}})$	NF
arwhead	2.99E+04	0.00E+00	90331	2.99E+04	170017
brybnd	3.60E+05	4.50E−15	370895	1.61E+04	270027
chrosen	1.99E+05	8.80E−14	851736	1.99E+05	140014
cragglvy	5.49E+06	3.40E+03	110483	5.49E+06	140014
dixmaane	7.36E+04	1.02E+00	170658	1.23E+04	90018
engval1	5.89E+05	1.10E+04	230880	5.89E+05	140014
eg2	8.41E+03	−9.99E+03	110353	−9.62E+03	160016
liarwhd	5.85E+06	7.89E−14	130464	3.71E+05	240024
nondia	1.01E+08	1.97E+00	90242	1.01E+08	90009
power	3.33E+11	1.64E+06	270951	3.33E+11	20002
sparsqur	1.40E+07	1.12E−18	410989	3.98E+06	90009
woods	4.79E+07	1.97E+04	90339	4.79E+07	120012

x_0: starting point; x_{fin}: final iterate; NF: number of function evaluations

5 Conclusions and future work

TODO.

References

[1] A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM J. Optim.*, 24:1238–1264, 2014.

[2] C. Cartis and L. Roberts. Scalable subspace methods for derivative-free nonlinear least-squares optimization. *Math. Program.*, 199:461–524, 2023.

[3] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*, volume 8 of *MOS-SIAM Ser. Optim.* SIAM, Philadelphia, 2009.

[4] A. R. Conn, Ph. L. Toint, A. Sartenaer, and N. I. M. Gould. On iterated-subspace minimization methods for nonlinear optimization. In Adams L. and Nazareth J. L., editors,

Linear and Nonlinear Conjugate Gradient-Related Methods, pages 50–78. SIAM, Philadelphia, 1996.

[5] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91:201–213, 2002.

[6] G. Fasano and S. Lucidi. A nonmonotone truncated Newton-Krylov method exploiting negative curvature directions, for large scale unconstrained optimization. *Optim. Lett.*, 3:521–535, 2009.

[7] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.*, 60:545–557, 2015.

[8] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Complexity and global rates of trust-region methods based on probabilistic models. *IMA J. Numer. Anal.*, 38:1579–1597, 2018.

[9] S. Gratton, L. N. Vicente, and Z. Zhang. Optimization by space transformation and decomposition, 2021.

[10] T. Gu et al. BBGP-sDFO: Batch Bayesian and Gaussian process enhanced subspace derivative free optimization for high-dimensional analog circuit synthesis. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 43:417–430, 2024.

[11] W. Hare, G. Jarry-Bolduc, and C. Planiden. Hessian approximations. *arXiv:2011.02584*, 2020.

[12] W. Hare and C. W. Royer. Detecting negative eigenvalues of exact and approximate Hessian matrices in optimization. *arXiv:2206.05318*, 2022.

[13] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45:503–528, 1989.

[14] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 20:172–191, 2009.

[15] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In G. Di Pillo and M. Roma, editors, *Large-scale Nonlinear Optimization*, pages 255–297. Springer, Boston, 2006.

[16] C. W. Royer and S. J. Wright. Complexity analysis of second-order line-search algorithms for smooth nonconvex optimization. *SIAM J. Optim.*, 28:1448–1477, 2018.

[17] Y. Yuan. A review on subspace methods for nonlinear optimization. In S. Y. Jang, Y. R. Kim, Lee D.-W., and I. Yie, editors, *Proceedings of the International Congress of Mathematicians (2014)*, pages 807–827. Kyung Moon Sa Co. Ltd., Seoul, 2014.

[18] Y. Yuan and J. Stoer. A subspace study on conjugate gradient algorithms. *ZAMM Z. Angew. Math. Mech.*, 75:69–77, 1995.

[19] C. Zhang, D. Ge, B. Jiang, and Y. Ye. DRSOM: A dimension reduced second-order method and preliminary analyses. *arXiv:2208.00208*, 2022.

[20] Z. Zhang. *On Derivative-Free Optimization Methods* (in Chinese). PhD thesis, Chinese Academy of Sciences, Beijing, 2012.

[21] Z. Zhang. PRIMA: Reference Implementation for Powell's Methods with Modernization and Amelioration. https://www.libprima.net, 2023.