2019

A derivative-free VU-algorithm for convex finite-max problems

Warren L. Hare
University of British Columbia, Okanagan, warren.hare@ubc.ca

Chayne Planiden
University of Wollongong, chayne@uow.edu.au

Claudia Sagastizabal
IMECC - UNICAMP

Recommended Citation

Hare, Warren L.; Planiden, Chayne; and Sagastizabal, Claudia, "A derivative-free VU-algorithm for convex finite-max problems" (2019). *Faculty of Engineering and Information Sciences - Papers: Part B*. 3339.
https://ro.uow.edu.au/eispapers1/3339

A derivative-free VU-algorithm for convex finite-max problems

Abstract

The *VU*-algorithm is a superlinearly convergent method for minimizing nonsmooth, convex functions. At each iteration, the algorithm works with a certain *V*-space and its orthogonal u-space, such that the nonsmoothness of the objective function is concentrated on its projection onto the *V*-space, and on the *U*-space the projection is smooth. This structure allows for an alternation between a Newton-like step where the function is smooth, and a proximal-point step that is used to find iterates with promising *VU*-decompositions. We establish a derivative-free variant of the *VU*-algorithm for convex finite-max objective functions. We show global convergence and provide numerical results from a proof-of-concept implementation, which demonstrates the feasibility and practical value of the approach. We also carry out some tests using nonconvex functions and discuss the results.

Disciplines

Engineering | Science and Technology Studies

Publication Details

Hare, W., Planiden, C. & Sagastizabal, C. (2019). A derivative-free VU-algorithm for convex finite-max problems. Optimization Methods and Software, Online First 1-39.

A derivative-free $\mathcal{VU}$-algorithm for convex finite-max problems

Warren Hare[*] Chayne Planiden[†] Claudia Sagastizábal[‡]

November 14, 2019

Abstract

The $\mathcal{VU}$-algorithm is a superlinearly convergent method for minimizing nonsmooth, convex functions. At each iteration, the algorithm works with a certain $\mathcal{V}$-space and its orthogonal $\mathcal{U}$-space, such that the nonsmoothness of the objective function is concentrated on its projection onto the $\mathcal{V}$-space, and on the $\mathcal{U}$-space the projection is smooth. This structure allows for an alternation between a Newton-like step where the function is smooth, and a proximal-point step that is used to find iterates with promising $\mathcal{VU}$-decompositions. We establish a derivative-free variant of the $\mathcal{VU}$-algorithm for convex finite-max objective functions. We show global convergence and provide numerical results from a proof-of-concept implementation, which demonstrates the feasibility and practical value of the approach. We also carry out some tests using nonconvex functions and discuss the results.

Keywords: convex minimization, derivative-free optimization, finite-max function, proximal-point mapping, $\mathcal{U}$-gradient, $\mathcal{U}$-Hessian, $\mathcal{VU}$-algorithm, $\mathcal{VU}$-decomposition

[*]Department of Mathematics, University of British Columbia, Okanagan Campus, Kelowna, B.C. V1V 1V7, Canada. Research partially supported by Natural Sciences and Engineering Research Council (NSERC) of Canada Discovery Grant 355571-2013. warren.hare@ubc.ca, ORCID 0000-0002-4240-3903

[†]School of Mathematics and Applied Statistics, University of Wollongong, Wollongong, NSW, 2500, Australia. Research supported by UBC UGF and by NSERC of Canada. chayne@uow.edu.au, ORCID 0000-0002-0412-8445, Tel: +61 (02) 4221 4564

[‡]Adjunct Researcher IMECC - UNICAMP, 13083-859, Campinas, SP, Brazil. Research partially supported by CNPq Grant 303905/2015-8, CEMEAI, and FAPERJ. sagastiz@unicamp.br, ORCID 0000-0002-9363-9297

AMS Subject Classification: Primary 49M30, 90C56 ; Secondary 65K10, 90C20.

1 Introduction

In this paper, we consider the finite-dimensional, unconstrained minimization problem

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

with f a nonsmooth, proper, *convex finite-max function*,

$$f(x) = \max_{i=1,\dots,m} f_i(x),$$

where for $i \in \{1, 2, \dots, m\}$ the subfunctions $f_i : \mathbb{R}^n \to \mathbb{R}$ are of class $\mathcal{C}^{2+}$. We work under the assumption that there is an oracle delivering only function values. We refer to the oracle as a "grey-box" because, for a given $x \in \mathbb{R}^n$ the oracle informs not only $f(x)$, but also the values of each subfunction $f_i(x)$. Our goal is to exploit the grey-box information in a derivative-free optimization setting, blended with a suitable variant of bundle methods.

We remark that this finite-max grey-box structure is a natural outcome of any optimization problem where multiple simulations are employed and then a worst-case outcome must be optimized. For example, see [11], where multiple simulations were used to analyze structural quality of buildings after seismic retrofitting was applied. In that case, the goal was to minimize the worst-case damage.

While we acknowledge that convexity of the objective in a finite-max grey-box function would be difficult to confirm, we note that many nonsmooth optimization problems take a convex finite-max structure (e.g., maximum eigenvalue optimization [13, Ex 3.10]). Regardless, we view this step as a necessary starting point for research in this direction. If a $\mathcal{VU}$-style algorithm cannot be designed and proven to converge for convex finite-max grey-box functions, then designing such algorithm for a more general setting would be intractable. The convergence theory in this paper is proved for the convex case only, but our numerical testing includes a set of nonconvex trials as well. Those results are discussed further in Section 5.

Derivative-free optimization (DFO) studies algorithms that use only function values to minimize the objective [9, 16]. Due to its broad applicability, particularly for optimizing simulations, DFO has seen many successful applications in the past decade (see [5, 25] and references therein). DFO algorithms can be split broadly into two categories, direct search methods and model-based methods. Model-based methods approximate the objective function with a model function, then use the model to guide the optimization algorithm [9, Part 4].

While model-based methods were originally designed for optimization of smooth objective functions (see, for example, [15, 18, 22, 56]), recent research has moved away from this assumption [23, 24, 40]. In [23, 24], it is assumed that the true objective function takes the form

$$f = \max\{f_i : i = 1, 2, \ldots, m\},$$

where each f_i is given by a blackbox that provides only function values. In [40], it is assumed that the objective function takes the form $f = \sum_{i=1}^{m} |f_i|$, where each f_i is given by a blackbox that provides only function values. In each case, it is shown that such information allows for the creation of a convergent model-based DFO algorithm for nonsmooth optimization.

Bundle methods proceed by collecting information (function values and subgradient vectors) along iterations, then using that information to build a model of the objective function and seek a new incumbent solution (called a *serious point* in bundle method literature) [12, 54]. Bundle methods have been widely established as the most robust and effective technique for nonsmooth optimization [3, 12, 27, 34, 38, 52, 55]. They are also well-known for their ability to work with the structure of a given problem. Specialized bundle methods have been developed considering eigenvalue optimization [29, 30], sum functions [14, 19], chance-constrained problems [2], composite functions [43, 59] and difference-convex functions [20, 32, 53].

Of particular interest to this paper is the $\mathcal{VU}$-algorithm for convex minimization [50]. The $\mathcal{VU}$-algorithm alternates between a proximal-point step and a '$\mathcal{U}$-Newton' step (see Step 4 of the Conceptual $\mathcal{VU}$-algorithm in Subsection 2.3) to achieve superlinear convergence in the minimization of nonsmooth convex functions [50]. The $\mathcal{VU}$-algorithm has proven effective in dealing with the challenges that arise in the minimization of nonsmooth convex functions [21, 44, 46, 49, 50]. It continues to be a method of interest in the optimization community, having been expanded to use on convex functions with primal-dual gradient structure [45, 46, 48] and even some nonconvex functions [49]. The basic tenet is to separate the space into two orthogonal subspaces, called the $\mathcal{V}$-space and the $\mathcal{U}$-space, such that near the current iteration point the nonsmoothness of f is captured in the $\mathcal{V}$-space and the smoothness of f is captured in the $\mathcal{U}$-space. This procedure is known as $\mathcal{VU}$-decomposition. Once this is accomplished, one takes a proximal-point step ($\mathcal{V}$-step) parallel to the $\mathcal{V}$-space, in order to find incumbent solutions with favourable $\mathcal{VU}$-decompositions, then a Newton-like step ($\mathcal{U}$-step) parallel to the $\mathcal{U}$-space. This process is repeated iteratively and converges to a minimizer of f. In fact, the $\mathcal{VU}$-algorithm has been proved superlinearly convergent under reasonable conditions [50]. Further details on the $\mathcal{VU}$-algorithm can be found in Section 2 of the present and proof of convergence (for oracles delivering subgradient information) is given in [50]. Techniques used in the implementation of the $\mathcal{VU}$-algorithm are also currently being used in gradient sampling methods [60, 61, 62], see Section 6 for details. To date, no one has developed a DFO $\mathcal{VU}$-algorithm.

In order to apply the $\mathcal{VU}$-algorithm, at each iteration it is necessary to do the $\mathcal{VU}$-decomposition, compute the proximal point to apply the $\mathcal{V}$-step, then compute the $\mathcal{U}$-gradient and $\mathcal{U}$-Hessian to apply the $\mathcal{U}$-step (each of these computations is formally defined in Section 2). In our grey-box optimization setting, none of these objects is directly available. However, in [21] it was shown that the $\mathcal{VU}$-decomposition, $\mathcal{U}$-gradient, and $\mathcal{U}$-Hessian can be approximated numerically with controlled precision for finite-max functions. Moreover, in [26] a derivative-free algorithm for computing proximal points of convex functions that only requires approximate subgradients was developed. Finally, in [24] it was shown how to approximate subgradients for convex finite-max functions using only function values. Combined, these three papers provide a sufficient foundation to develop a derivative-free $\mathcal{VU}$-algorithm suitable for our grey-box optimization setting. We show that at each iteration, one can approximate subgradients of the objective function as closely as one wishes and use the inexact first-order information to obtain approximations of all the necessary components of the algorithm. We prove that the results of global convergence in [50] can be extended to the framework of inexact gradients and Hessians.

The remainder of this paper is organized as follows. We finish the present section with notation and assumptions on the objective function. Section 2 contains the initial basic definitions and provides a brief primer on the $\mathcal{VU}$-algorithm. Section 3 presents details on the simplex gradient and Frobenius norm, which are tools needed for the DFO version of the algorithm, and establishes the DFO $\mathcal{VU}$-algorithm. Section 3 includes the DFO $\mathcal{VU}$-algorithm pseudo-code and provides some comments comparing our algorithm to other established DFO methods. In Section 4, we examine the convergence properties of the algorithm. In Section 5, we showcase numerical results obtained for some established testing functions and some randomly generated max-of-quadratic functions. The numerical behaviour of the method on nonconvex functions is also explored, resulting in insight on its good performance (not yet backed up by a convergence analysis). Section 6 summarizes this work and discusses future possibilities of this field of research, in particular regarding recent variants of gradient sampling methods [60, 61, 62].

1.1 Notation

We work in the finite-dimensional space $\mathbb{R}^n$, with inner product $x^\top y = \sum_{i=1}^n x_i y_i$ and induced norm $\|x\| = \sqrt{x^\top x}$. We use standard notation and concepts from convex analysis found in [58]. The identity matrix is denoted by $\mathtt{I}$. We denote by B_δ the open ball of radius δ about the origin. Given a set S, we denote its interior, closure and relative interior by $\operatorname{int}(S)$, $\operatorname{cl}(S)$ and $\operatorname{ri}(S)$, respectively. We denote the smallest convex set containing S, i.e. the convex hull of S, by $\operatorname{conv} S$. The set of all linear combinations of the vectors in T is denoted by $\operatorname{span} T$.

As the objective function f is convex and finite-valued, the *subdifferential* of f at a point $\bar{x}$, defined by the set

$$\partial f(\bar{x}) = \{g \in \mathbb{R}^n : f(x) \geq f(\bar{x}) + g^\top (x - \bar{x}) \text{ for all } x \in \mathbb{R}^n\},$$

is well-defined and never empty. An element $g \in \partial f(\bar{x})$ is called a *subgradient* of f at $\bar{x}$. The *ε-subdifferential* of f at x is denoted $\partial_\varepsilon f(x)$ (with $g \in \partial_\varepsilon f(x)$ called an *ε-subgradient*) and is defined by

$$\partial_\varepsilon f(\bar{x}) = \{g \in \mathbb{R}^n : f(x) \geq f(\bar{x}) + g^\top (x - \bar{x}) - \varepsilon \text{ for all } x \in \mathbb{R}^n\}.$$

Given a finite-max function, the set of active indices provides an alternate manner of constructing the subdifferential:

$$\partial f(\bar{x}) = \operatorname{conv}\{\nabla f_i(\bar{x}) : i \in A(\bar{x})\}, \tag{2}$$

where the active set $A(\bar{x}) = \{i : f_i(\bar{x}) = f(\bar{x})\}$.

Definition 1.1. *A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is a $\mathcal{C}^k$ ($\mathcal{C}^{k+}$)* function *if all partial derivatives of f of degree* 0 *to k exist and are (locally Lipschitz) continuous.*

Definition 1.2. *Given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$, the* Jacobian *of f, written J_f, is the matrix of all partial derivatives of f:*

$$J_f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$

Definition 1.3. *Given a point $\bar{x}$ and a* proximal parameter $r > 0$, *the* proximal mapping, *denoted* $\text{Prox}_f^r(\bar{x})$, *is defined by*

$$\text{Prox}_f^r(\bar{x}) = \operatorname*{argmin}_y \left\{ f(y) + \frac{r}{2}\|y - \bar{x}\|^2 \right\}.$$

1.2 Assumptions

Throughout this paper, we assume the following for problem (1).

***Assumption* 1.4.** *The objective function $f : \mathbb{R}^n \to \mathbb{R}$ is convex and defined through the maximum of a finite number of subfunctions,*

$$f(x) = \max_{i=1,\ldots,m} f_i(x),$$

where each $f_i \in \mathcal{C}^{2+}$. Furthermore, at each given point $\bar{x}$ the grey-box returns the individual function values $f_i(x)$ and as such also provides indices of active subfunctions, i.e.,

$$A(\bar{x}) = \{i : f_i(\bar{x}) = f(\bar{x})\}.$$

***Assumption* 1.5.** *The objective function f has compact* lower level sets, *that is, the set*

$$S_\beta = \{x : f(x) \le \beta\}$$

is compact for any choice of $\beta \in \mathbb{R}$.

***Assumption* 1.6.** *For any fixed $\bar{x} \in \mathbb{R}^n$, the set of* active gradients

$$\{\nabla f_i(\bar{x}) : i \in A(\bar{x})\}$$

is affinely independent. That is, the only scalars λ_i that satisfy

$$\sum_{i \in A(\bar{x})} \lambda_i \nabla f_i(\bar{x}) = 0, \quad \sum_{i \in A(\bar{x})} \lambda_i = 0$$

are $\lambda_i = 0$ for all $i \in A(\bar{x})$.

2 Background and $\mathcal{VU}$-theory

At any point $x \in \mathbb{R}^n$, the space can be split into two orthogonal subspaces called the $\mathcal{U}$-space and the $\mathcal{V}$-space, such that the nonsmoothness of f is captured entirely in the $\mathcal{V}$-space, while on the $\mathcal{U}$-space f behaves smoothly. The $\mathcal{VU}$-method tracks a smooth trajectory of f along which a Newton-like update can be done, even though the function is not differentiable everywhere. The smooth trajectory is special in the sense that its $\mathcal{VU}$-decomposition has a $\mathcal{V}$-component that converges faster than its $\mathcal{U}$-component. Along the smooth trajectory, the rate of convergence is driven by the speed of the $\mathcal{U}$-component, which is updated using a (fast) Newton step. This explains the superlinear speed of convergence of conceptual $\mathcal{VU}$-methods under certain assumptions (such as having perfect knowledge of the full subdifferential of f at a minimizer and of the matrices involved in a second-order expansion of the smooth trajectory); see Section 2.3.

The algorithmic identification of the smooth trajectories is possible thanks to two useful relations established in [17] and [47]. Specifically, the first work shows that a bundle mechanism gives asymptotically the exact value of the proximal point operator at a given point, for oracles delivering subgradient information. The second work, see Theorem 2.2, relates proximal points with the smooth trajectory. A sound combination of these elements gives an implementable form to the conceptual $\mathcal{VU}$-algorithm in Section 2.3. Our contribution extends the relations above to the grey-box oracle in Assumption 1.4, by suitably coupling those bundle-method results with DFO techniques to derive the implementable DFO $\mathcal{VU}$-algorithm in Section 3.1.

The main relations and formal definitions of the $\mathcal{VU}$-decomposition, the $\mathcal{U}$-Lagrangian that yields the smooth trajectories, and the proximal point mapping are recalled below.

Definition 2.1. *Fix $\bar{x} \in \mathbb{R}^n$ and let $g \in \operatorname{ri}(\partial f(\bar{x}))$. The $\mathcal{VU}$-*decomposition *of $\mathbb{R}^n$ for f at $\bar{x}$ is the separation of $\mathbb{R}^n$ into the following two orthogonal subspaces:*

$$\mathcal{V}(\bar{x}) = \operatorname{span}\{\partial f(\bar{x}) - g\} \qquad \textit{and} \qquad U(\bar{x}) = [\mathcal{V}(\bar{x})]^{\perp}.$$

This decomposition is independent of the choice of $g \in \operatorname{ri} \partial f(\bar{x})$ [42, Proposition 2.2]. With $\mathrm{V} \in \mathbb{R}^{n\times \dim \mathcal{V}}$ a basis matrix for the $\mathcal{V}$-space (not necessarily orthonormal) and $\mathrm{U} \in \mathbb{R}^{n\times \dim \mathcal{U}}$ an orthonormal basis matrix for the

$\mathcal{U}$-space, every $x \in \mathbb{R}^n$ can be decomposed into components $x_\mathcal{U} \in \mathbb{R}^{\dim \mathcal{U}}$ and $x_\mathcal{V} \in \mathbb{R}^{\dim \mathcal{V}}$ [42, Section 2]. Defining

$$x_\mathcal{U} = \mathrm{U}^\top x \qquad \text{and} \qquad x_\mathcal{V} = \left(\mathrm{V}^\top \mathrm{V}\right)^{-1} \mathrm{V}^\top x,$$

we write

$$x = \mathrm{U}\, x_\mathcal{U} + \mathrm{V}\, x_\mathcal{V}.$$

(rather than the traditional Minkowski sum). Henceforth, the notation $\mathbb{R}^{|\mathcal{U}|}$ and $\mathbb{R}^{|\mathcal{V}|}$ represents $\mathbb{R}^{\dim \mathcal{U}}$ and $\mathbb{R}^{\dim \mathcal{V}}$, respectively.

Given a subgradient $g \in \partial f(\bar{x})$ with $\mathcal{V}$-component $g_\mathcal{V}$, the *$\mathcal{U}$-Lagrangian* of f, $L_\mathcal{U}(u; g_\mathcal{V}) : \mathbb{R}^{|\mathcal{U}|} \to \mathbb{R}$ is defined [50, Section 2.1] as follows:

$$L_\mathcal{U}(u; g_\mathcal{V}) = \min_{v \in \mathbb{R}^{|\mathcal{V}|}} \left\{ f\left(\bar{x} + \mathrm{U}\, u + \mathrm{V}\, v\right) - g^\top \mathrm{V}\, v \right\}.$$

The associated set of $\mathcal{V}$-space minimizers is

$$\begin{array}{rcl} W(u; g_\mathcal{V}) & = & \operatorname{argmin}_{v \in \mathbb{R}^{|\mathcal{V}|}} \left\{ f\left(\bar{x} + \mathrm{U}\, u + \mathrm{V}\, v\right) - g^\top \mathrm{V}\, v \right\} \\ & = & \{ \mathrm{V}\, v : L_\mathcal{U}(u; g_\mathcal{V}) = f(\bar{x} + \mathrm{U}\, u + \mathrm{V}\, v) - g^\top \mathrm{V}\, v \}. \end{array}$$

The $\mathcal{U}$-gradient $\nabla L_\mathcal{U}(0; g_\mathcal{V})$ and the $\mathcal{U}$-Hessian $\nabla^2 L_\mathcal{U}(0; g_\mathcal{V})$ are then defined as the gradient and Hessian, respectively, of the $\mathcal{U}$-Lagrangian. For f convex, each $\mathcal{U}$-Lagrangian is a convex function that is differentiable at $u = 0$, with

$$\nabla L_\mathcal{U}(0; g_\mathcal{V}) = g_\mathcal{U} = \mathrm{U}^\top g = \mathrm{U}^\top \tilde{g} \text{ for all } \tilde{g} \in \partial f(x) \qquad [50, \text{Section } 2.1].$$

If $L_\mathcal{U}(u; g_\mathcal{V})$ has a Hessian at $u = 0$, then the second-order expansion of $L_\mathcal{U}$ also provides a second-order expansion of f in the $\mathcal{U}$-space, which thereby allows for a so-called $\mathcal{U}$-Newton step. General conditions for existence of the $\mathcal{U}$-Hessian are found in [50]. However, for the purpose of this paper, we note that Assumptions 1.4 and 1.6, combined with $g \in \operatorname{ri} \partial f(x)$, imply the existence of a $\mathcal{U}$-Hessian at the origin [21, Lemma 2.6, Lemma 3.5]. When $\bar{x}$ minimizes f, we have $0 \in \partial f(\bar{x})$ [50, Section 2.1]. This gives

$$\nabla L_\mathcal{U}(0; g_\mathcal{V}) = 0 \text{ for all } g \in \partial f(\bar{x}),$$

and $L_\mathcal{U}$ is minimized at $u = 0$, which subsequently yields $L_\mathcal{U}(0; 0) = f(\bar{x})$.

2.1 Relation with the proximal point operator

The second-order expansion of f in the $\mathcal{U}$-space allows the $\mathcal{VU}$-algorithm to take $\mathcal{U}$-Newton steps, which in turn allows for rapid convergence. However, in order to be effective, the algorithm must seek out iterates at which the $\mathcal{U}$-space lines up with the $\mathcal{U}$-space at the minimizer. This is accomplished through the proximal point operation. When f is convex, the proximal mapping Prox_f^r is a singleton, called the proximal point. When computed close to a minimizer $\bar{x}$, the proximal point has a very important relationship with the smooth trajectory provided by the $\mathcal{U}$-Lagrangian minimizers, called the *primal track* in [49, §1].

As shown in [42, Corollary 3.5], for sufficiently small u the trajectories created from the set of $\mathcal{V}$-space minimizers, that is $\bar{x} + \mathrm{U}\,u + \mathrm{V}\,v(u)$, are smooth and are tangent to $\mathcal{U}$ at $\bar{x}$, because $v(u) = O(\|u\|^2)$. When, in addition, the Hessian of $L_{\mathcal{U}}(u;0)$ exists at $u = 0$ (see [42, Definition 3.8] and the preamble), the second-order expansion of $L_{\mathcal{U}}$ is possible [50, Section 2.2]. Lemma 3 of [50] shows that in that case, the derivative of the trajectory provides a $\mathcal{C}^1$ $\mathcal{U}$-gradient.

The connection with the proximal point is given by the following very useful equivalence.

Theorem 2.2. [50, Theorem 4] *Let $\chi(u)$ be a primal track leading to a minimizer $\bar{x} \in \mathbb{R}^n$. Suppose that $0 \in \partial f(\bar{x})$ and that we have a function $r(x) > 0$ such that $r(x)\|x-\bar{x}\| \to 0$ when $x \to \bar{x}$. Define $u_r(x) = (\text{Prox}_f^r(x) - \bar{x})_{\mathcal{U}}$. Then for all x close enough to $\bar{x}$ and $r = r(x)$, we have that*

$$\text{Prox}_f^r(x) = \chi(u_r(x)) = \bar{x} + u_r(x) \oplus v(u_r(x)).$$

Moreover, $u_r(x) \to 0$ as $x \to \bar{x}$.

In Theorem 2.2, $r(x)$ plays the role of a prox-parameter that can be dynamically selected within an algorithm (provided $r(x)\|x - \bar{x}\| \to 0$ when $x \to \bar{x}$). The conclusion of Theorem 2.2 allows us to concentrate on finding the proximal point instead of being concerned about how to find the primal track, since close to $\bar{x}$ they are one and the same. Moreover, note that Theorem 2.2 does not require $r(x)$ to be constant. This provides valuable flexibility that greatly improves numerical performance in $\mathcal{VU}$-algorithms. Finally, note that a derivative-free routine for finding the proximal point of a convex function at a given point already exists [37]. We give a brief summary of the method next.

2.2 Computing proximal points

Given a convex function f and an initial point y_0, at iteration j of the bundle routine we choose any subgradient $g_j \in \partial f(y_j)$ and define the *linearization error*:

$$E_j = E(x, y_j) = f(x) - f(y_j) - g_j^\top (x - y_j),$$

with $E_j \geq 0$ by convexity of f. Since $f(z) \geq f(y_j) + g_j^\top (z - y_j)$ for all $z \in \mathbb{R}^n$,

$$f(z) \geq f(x) + g_j^\top (z - x) - E_j \text{ for all } z \in \mathbb{R}^n .$$

In other words, $g_j \in \partial_{E_j} f(x)$. The bundle $\{(E_j, g_j)\}_{j \in \mathcal{B}}$, where $\mathcal{B}$ is a set that indexes information from previous iterations, is used to construct a convex piecewise-linear function φ_j that approximates and minorizes f. Then the new iteration point $y_{j+1} = \text{Prox}^r_{\varphi_j}(y_j)$ is found and the process repeats. This method is proved in [17] to converge to $\text{Prox}^r_f(y_0)$.

The cutting-plane model φ_j uses subgradient information that is not available in our case. In the DFO setting, subgradients will be estimated by means of certain *simplex gradients* using functional information only; see Section 3.

2.3 The $\mathcal{VU}$-algorithm

When a primal track exists, the $\mathcal{VU}$-algorithm takes a step approximately following the primal track by way of a *predictor step ($\mathcal{U}$-step)*, which is a Newton-like step parallel to the $\mathcal{U}$-space, followed by a *corrector step ($\mathcal{V}$-step)*, which is a bundle subroutine estimate of the proximal point in the $\mathcal{V}$-space. The $\mathcal{V}$-step outputs a potential primal track point, which is then checked and either accepted or rejected, depending on whether sufficient descent is achieved. We now state an abbreviated version of the conceptual $\mathcal{VU}$-algorithm presented in [50].

Conceptual $\mathcal{VU}$-algorithm

Step 0: Initialize the starting point x_0, proximal parameter $r > 0$, iteration counter $k = 0$ and other parameters.

Step 1: Given $g \in \partial f(x_k)$, compute bases V and U for the $\mathcal{VU}$-decomposition

Step 2: Compute an approximate proximal point $x_{k+1} \approx \text{Prox}^r_f(x^k)$. Increment $k \mapsto k + 1$.

Step 3: If x_k does not show sufficient descent, then declare a null step and repeat Step 2 to higher precision. If x_k does show sufficient descent, then check stopping conditions and either stop or continue to Step 4.

Step 4: Compute the $\mathcal{U}$-gradient $\nabla L_{\mathcal{U}}(0; g_{\mathcal{V}})$ and $\mathcal{U}$-Hessian $\nabla^2 L_{\mathcal{U}}(0; g_{\mathcal{V}})$. Take a $\mathcal{U}$-Newton step by solving

$$\nabla^2 L_{\mathcal{U}}(0; g_{\mathcal{V}})\Delta u = -\nabla L_{\mathcal{U}}(0; g_{\mathcal{V}})$$

for Δu and setting $x_{k+1} = x_k + \mathrm{U}\,\Delta u$.
Increment $k \mapsto k+1$, update r, and go to Step 1.
End algorithm.

Notice in the conceptual $\mathcal{VU}$-algorithm that k is incremented both in Step 2 and Step 4. This is intentional and best understood by thinking of Steps 1-2 as the 'inner loop' of Steps 1-4. Hence, the conceptual algorithm increments k at the end of each inner loop and the end of the outer loop.

3 Defining inexact subgradients and related approximations

We now consider how to make implementable the conceptual $\mathcal{VU}$-algorithm in a derivative-free setting, as provided by Assumptions 1.4 through 1.6. In order to prove convergence, we make use of the results of [21] and [26]. We use the techniques in [21] to approximate a subgradient, the $\mathcal{VU}$-decomposition, the $\mathcal{U}$-gradient and the $\mathcal{U}$-Hessian for the function f at a point $\bar{x}$.

To define an inexact subgradient for f, we make use of the *simplex gradients* of each f_i. The simplex gradient is defined as the gradient of the approximation resulting from a linear interpolation of f over a set of $n+1$ points in $\mathbb{R}^n$ [35].

Definition 3.1. *Let $Y = [y_0, y_1, \ldots, y_n]$ be a set of affinely independent points in $\mathbb{R}^n$. Then it is said that Y* forms a simplex*, with* simplex diameter

$$\varepsilon = \max_{j=1,\ldots n} \|y_j - y_0\|.$$

The simplex gradient *of a function f_i over Y is given by*

$$\nabla_\varepsilon f_i(Y) = M^{-1}\delta f_i(Y),$$

where

$$M = \begin{bmatrix} y_1 - y_0 \cdots y_n - y_0 \end{bmatrix}^\top, \quad \text{and} \quad \delta f_i(Y) = \begin{bmatrix} f_i(y_1) - f_i(y_0) \\ \vdots \\ f_i(y_n) - f_i(y_0) \end{bmatrix}.$$

The condition number *of Y is given by $\|\hat{M}^{-1}\|$, where*

$$\hat{M} = \frac{1}{\varepsilon}[y_1 - y_0 \;\; y_2 - y_0 \;\; \dots \;\; y_n - y_0]^\top.$$

An important aspect of the condition number is that it is always possible to keep it bounded away from zero while simultaneously making ε arbitrarily close to zero (see Remark 3.4). The following result provides an error bound for the distance between the simplex gradient and the exact gradient for a smooth function. We present a slightly simplified version, as we have not defined notation for the more general version ($f_i \in \mathcal{C}^2$ can be relaxed to $f_i \in \mathcal{C}^{1+}$).

Theorem 3.2. [35, Lemma 6.2.1] *Consider $f_i \in \mathcal{C}^2$. Let $Y = [y_0, y_1, \dots, y_n]$ form a simplex. Then there exists μ constant depending on n and the local Lipschitz constant of ∇f_i such that*

$$\|\nabla_\varepsilon f_i(Y) - \nabla f_i(y_0)\| \le \varepsilon\mu\|\hat{M}^{-1}\|.$$

We set $y_0 = \bar{x}$, and y_1 through y_n to $\bar{x} + \varepsilon e_i$, where e_i is the i^{th} canonical basis vector. If desired, a rotation matrix can be used to prevent the y_i vectors from being oriented in the coordinate directions every time. Now we define Subroutine 3.3, which we use to find an approximate subgradient g^ε, approximations of the subspace bases V and U and the approximate $\mathcal{U}$-gradient $\nabla_\varepsilon L_\mathcal{U}(0; g^\varepsilon_\mathcal{V})$.

Subroutine 3.3 (First-order approximations)**.**

Step 0: *Input $\bar{x}$ and ε.*

Step 1: *Set $Y = [\bar{x} \quad \bar{x} + \varepsilon e_1 \quad \bar{x} + \varepsilon e_2 \quad \cdots \quad \bar{x} + \varepsilon e_n]$.*

Step 2: *Find $A(\bar{x})$ and calculate $\nabla_\varepsilon f_i(Y)$ for each $i \in A(\bar{x})$.*

Step 3: *Set*

(i) $\tilde{g}^\varepsilon = \frac{1}{|A(\bar{x})|} \sum\limits_{i \in A(\bar{x})} \nabla_\varepsilon f_i(Y)$;

(ii) V *to be the matrix of column vectors*

$$\nabla_\varepsilon f_i(Y) - \nabla_\varepsilon f_I(Y)$$

for each $i \in A(\bar{x}) \setminus \{I\}$, where I is the first element of $A(\bar{x})$;

(iii) $\mathrm{U} = \operatorname{null} \mathrm{V} / |\operatorname{null} \mathrm{V}|$;

(iv) $\nabla_\varepsilon L_{\mathcal{U}}(0; \tilde{g}^\varepsilon_{\mathcal{V}}) = \mathrm{U}^\top \tilde{g}^\varepsilon$.

End subroutine.

Remark 3.4. *Using Y from Step 1, we have*

$$\hat{M} = \frac{1}{\varepsilon}[\bar{x} + \varepsilon e_1 - \bar{x} \quad \bar{x} + \varepsilon e_2 - \bar{x} \quad \cdots \bar{x} + \varepsilon e_n - \bar{x}] = \mathtt{I},$$

so that $\|\hat{M}^{-1}\| = 1$ while ε can be arbitrarily small.

Remark 3.5. *By fixing Y in Step 1, we have g^ϵ and $\nabla_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}})$ defined directly using ϵ. This is done primarily to simplify notation. If a more flexible implementation is desired, the notation $g^\epsilon(Y)$ and $\nabla_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}})(Y)$ could be employed.*

The following theorem shows that the outputs g^ε and $\nabla_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}})$ from Subroutine 3.3 are good approximations.

Theorem 3.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$ satisfy Assumptions 1.4 and 1.6. Fix $\bar{x} \in \operatorname{dom} f$. Then there exist μ constant depending on $\bar{x}$ and $g \in \operatorname{ri} \partial f(\bar{x})$ such that for $\varepsilon > 0$ sufficiently small, one can obtain*

(i) *an approximate subgradient g^ε such that*

$$\|g^\varepsilon - g\| \le \varepsilon\mu, \|g^\varepsilon_{\mathcal{U}} - g_{\mathcal{U}}\| \le \varepsilon\mu \text{ and } \|g^\varepsilon_{\mathcal{V}} - g_{\mathcal{V}}\| \le \varepsilon\mu;$$

(ii) *the approximate $\mathcal{U}$-gradient $\nabla_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}})$ such that*

$$\|\nabla_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}}) - \nabla L_{\mathcal{U}}(0; g_{\mathcal{V}})\| \le \varepsilon\mu.$$

Proof. By Theorem 3.2 with $\|\hat{M}^{-1}\| = 1$ as per Remark 3.4, there exists $\mu_1 > 0$ such that $\|\nabla_\varepsilon f_i(Y) - \nabla f_i(\bar{x})\| \le \varepsilon\mu_1$. Then by [21, Lemma 2.6], there exist unique $\lambda_i \ge 0$ with $\sum_{i \in A(\bar{x})} \lambda_i = 1$ such that

$$g = \sum_{i \in A(\bar{x})} \lambda_i \nabla f_i(\bar{x}) \in \operatorname{ri} \partial f(\bar{x}) \qquad \text{and} \qquad g^\varepsilon = \sum_{i \in A(\bar{x})} \lambda_i \nabla_\varepsilon f_i(Y) \in \operatorname{ri} \partial_\varepsilon f(\bar{x})$$

are such that $\|g^\varepsilon - g\| \le \varepsilon\mu_1$. By [21, Lem 4.3] and [21, Thm 5.3], we have the existence of $\mu_2, \mu_3 > 0$ such that $\|g^\varepsilon_{\mathcal{U}} - g_{\mathcal{U}}\| \le \varepsilon\mu_2$, $\|g^\varepsilon_{\mathcal{V}} - g_{\mathcal{V}}\| \le \varepsilon\mu_2$ and $\|\nabla_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}}) - \nabla L_{\mathcal{U}}(0; g_{\mathcal{V}})\| \le \varepsilon\mu_3$. Setting $\mu = \max\{\mu_1, \mu_2, \mu_3\}$ completes the proof. □

Next, we find the approximate $\mathcal{U}$-Hessian $\nabla^2_\varepsilon L_\mathcal{U}(0; g^\varepsilon_\mathcal{V})$, as outlined in [21]. To do so, we need the Frobenius norm.

Definition 3.7. *The Frobenius norm $\|M\|_F$ of a matrix $M \in \mathbb{R}^{p\times q}$ with elements a_{ij} is defined by*

$$\|M\|_F = \sqrt{\sum_{i=1}^{p}\sum_{j=1}^{q} a_{ij}^2}.$$

We define the matrix $Z \in \mathbb{R}^{n\times(2n+1)}$:

$$Z = [\bar{x} \quad \bar{x}+\varepsilon e_1 \quad \bar{x}-\varepsilon e_1 \quad \cdots \quad \bar{x}+\varepsilon e_n \quad \bar{x}-\varepsilon e_n].$$

To build an approximate Hessian of $f_i(\bar{x})$ for each $i \in A(\bar{x})$, we solve the minimum Frobenius norm problem:

$$\{(H_i^*, D_i^*, C_i^*)\} \in \operatorname*{argmin}_{H,D,C} \left\{ \|H_i\|_F : \frac{1}{2} z_j^\top H_i z_j + D_i^\top z_j + C_i = f_i(z_j) \text{ for all } z_j \in Z \right\}$$

and set $\nabla^2_\varepsilon f_i(Z) = H_i^*$, where $z_j \in Z$ means z_j is the j^{th} column of Z. The solution is obtained by solving a quadratic program. We then set

$$H = \frac{1}{|A(\bar{x})|} \sum_{i\in A(\bar{x})} \nabla^2_\varepsilon f_i(Z)$$

and define the approximate $\mathcal{U}$-Hessian of $f(\bar{x})$:

$$\nabla^2_\varepsilon L_\mathcal{U}(0; g^\varepsilon_\mathcal{V}) = \mathrm{U}^\top H \, \mathrm{U}\,.$$

The following result provides the error bound for the approximate Hessian.

Theorem 3.8. [21, Theorem 6.1] *Let $\bar{x}$ be fixed. Suppose that Assumption 1.6 holds and that for any $\varepsilon > 0$ there exists μ constant depending on $\bar{x}$ such that $\|\nabla_\varepsilon f_i(\bar{x}) - \nabla f(\bar{x})\| < \varepsilon\mu$ and $\|\nabla^2_\varepsilon f_i(\bar{x}) - \nabla^2 f(\bar{x})\| < \varepsilon\mu$. Then*

$$\|\nabla^2 L_\mathcal{U}(0; g_\mathcal{V}) - \nabla^2_\varepsilon L_\mathcal{U}(0; g^\varepsilon_\mathcal{V})\| \le \varepsilon \left[2\sqrt{2}\sqrt{|A(\bar{x})|-1}\|\,\mathrm{V}^\dagger\,\|\|H\|(2\mu + \mu^2\varepsilon) + \mu \right],$$

where $\mathrm{V}^\dagger$ represents the Moore-Penrose pseudo-inverse of V. Thus,

$$\lim_{\varepsilon \searrow 0} \nabla^2_\varepsilon L_\mathcal{U}(0; g^\varepsilon_\mathcal{V}) = \nabla^2 L_\mathcal{U}(0; g_\mathcal{V}).$$

Now we state Subroutine 3.9, which is used to find the approximate $\mathcal{U}$-Hessian of f at $\bar{x}$.

Subroutine 3.9 (Second-order approximation)**.**

Step 0: *Input* $\bar{x}$, ε, $A(\bar{x})$ *and* U.

Step 1: *Set* $Z = [\bar{x} \quad \bar{x} + \varepsilon e_1 \quad \bar{x} - \varepsilon e_1 \quad \cdots \quad \bar{x} + \varepsilon e_n \quad \bar{x} - \varepsilon e_n]$.

Step 2: *Calculate* $\nabla^2_\varepsilon f_i(Z)$ *for each* $i \in A(\bar{x})$.

Step 3: *Set* $\nabla^2_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}}) = \mathrm{U}^\top \left(\frac{1}{|A(\bar{x})|} \sum\limits_{i \in A(\bar{x})} \nabla^2_\varepsilon f_i(Z) \right) \mathrm{U}$.

End subroutine.

Remark 3.10. *Similar to Subroutine 3.3, by fixing Z in Step 1, there is no need to put it in the notation for our approximate $\mathcal{U}$-Hessian. If a more flexible algorithm were desired, then the notation $\nabla^2_\varepsilon L_{\mathcal{U}}(0; g^\varepsilon_{\mathcal{V}})(Z)$ could be used.*

Theorems 3.6 and 3.8 provide us with the tools needed to perform the approximate $\mathcal{U}$-step in the derivative-free $\mathcal{VU}$-algorithm. In order to perform the approximate $\mathcal{V}$-step, we need to be able to approximate a proximal point in a derivative-free setting. A subroutine that accomplishes this, called the *tilt-correct DFO proximal bundle method*, was introduced in [26]. Details are reproduced in Step 2 of the DFO $\mathcal{VU}$-algorithm below. At iteration j of said subroutine, a subgradient is approximated by modelling f with a piecewise-linear function φ_j and then finding the proximal point of φ_j. This method is proved in [26] to converge to the desired proximal point within a preset tolerance. Theorem 3.6(i) provides the approximate subgradients required for this step.

The tilt-correct DFO proximal bundle method involves a possible correction to the approximate subgradient found at each iteration (Step 1.1 of the upcoming DFO $\mathcal{VU}$-algorithm), which ensures that the model function value at the current iterate x_k is not greater than the objective function value at x_k. This is not a concern when exact subgradients are available, because then the model function naturally bounds the (convex) objective function from below, but when using approximate subgradients it is possible for the model function to lie partially above the objective function. In that case, tilting the linear piece down until the model and true function values are consistent at x_k makes the model no worse [26, Lemma 3.1]. The tilt procedure is explained in [26, §3.1]. Suffice it to say here that once g^ε is found, it can be replaced by the approximate subgradient defined by (3), which complies with all of our requirements.

3.1 The DFO $\mathcal{VU}$-Algorithm

In the following algorithm, we use k for the outer counter and j for the inner ($\mathcal{V}$-step subroutine) counter. Henceforth, we refer to this algorithm as DFO-VU.

Step 0: *Initialization.* Choose a stopping tolerance $\delta \geq 0$, an accuracy tolerance $\varepsilon_{\mathtt{min}} \geq 0$ for the subgradient errors, a descent-check parameter $m \in (0,1)$ and a proximal parameter $r > 0$. Choose an initial point $x_0 \in \operatorname{dom} f$ and an initial subgradient accuracy $\varepsilon_0 \geq 0$. Set $k = 0$.

Step 1: *$\mathcal{V}$-step.*

Step 1.0: *Initialization.* Set $j = 0$, $z_0 = x_k$ and $\mathcal{B}_0 = \{0\}$.

Step 1.1: *Linearization.* Call Subroutine 3.3 with input (z_j, ε_k) to find $\tilde{g}_j^{\varepsilon_k}$. Compute $E_j = f(z_j) + \tilde{g}_j^{\varepsilon_k\top}(z_0 - z_j) - f(z_0)$ and set

$$g_j^{\varepsilon_k} = \tilde{g}_j^{\varepsilon_k} + \max(0, E_j)\frac{z_0 - z_j}{\|z_0 - z_j\|^2}. \tag{3}$$

Step 1.2: *Model.* Define

$$\varphi_j^{\varepsilon_k}(z) = \max_{i \in \mathcal{B}_j}\left\{f(z_i) + g_i^{\varepsilon_k\top}(z - z_i)\right\}.$$

Step 1.3: *Proximal Point.* Calculate $z_{j+1} = \operatorname{Prox}^r_{\varphi_j^{\varepsilon_k}}(z_0)$.

Step 1.4: *Stopping Test.* If $f(z_{j+1}) - \varphi_j^{\varepsilon_k}(z_{j+1}) \leq \varepsilon_k^2/r$, set $x_{k+1} = z_{j+1}$, calculate the aggregate subgradient of the model function: $s_{k+1} = r(z_0 - z_{j+1})$, and go to Step 2.

Step 1.5: *Update and Loop.* Create the aggregate bundle element

$$(z_{j+1}, \varphi_j^{\varepsilon_k}, r(z_0 - z_{j+1})).$$

Create $\mathcal{B}_{j+1}$ such that $\{-1, 0, j+1\} \subseteq \mathcal{B}_{j+1} \subseteq \{-1, 0, 1, 2, \cdots, j+1\}$. Increment $j \mapsto j+1$ and go to Step 1.1.

Step 2: *Stopping Test.* If $\|s_{k+1}\|^2 \leq \delta$ and $\varepsilon_k \leq \varepsilon_{\mathtt{min}}$, output x_{k+1} and stop.

Step 3: *Update and Loop.*

Case 3.1: If $f(x_k) - f(x_{k+1}) \geq \frac{m}{2r}\|s_{k+1}\|^2$ and $\|s_{k+1}\|^2 \leq \delta$ and $\varepsilon_k > \varepsilon_{\mathtt{min}}$, declare SERIOUS STEP and set $\varepsilon_{k+1} = \varepsilon_k/2$.

Case 3.2: If $f(x_k) - f(x_{k+1}) \geq \frac{m}{2r}\|s_{k+1}\|^2$ and $\|s_{k+1}\|^2 > \delta$, declare SERIOUS STEP and set $\varepsilon_{k+1} = \varepsilon_k$.

Case 3.3: If $f(x_k) - f(x_{k+1}) < \frac{m}{2r}\|s_{k+1}\|^2$, declare NULL STEP and set $\varepsilon_{k+1} = \varepsilon_k/2$.

Increment $k \mapsto k+1$. If SERIOUS STEP, go to Step 4. If NULL STEP, go to Step 1.

Step 4: *$\mathcal{U}$-step.* Call Subroutine 3.3 with input (x_k, ε_k) to find $A(x_k)$, $g_k^{\varepsilon_k}$, U_k and $\nabla_\varepsilon L_{\mathcal{U}}(0; (g_k^{\varepsilon_k})_{\mathcal{V}})$. Call Subroutine 3.9 with input $(x_k, \varepsilon_k, A(x_k), \mathrm{U}_k)$ to find $\nabla_\varepsilon^2 L_{\mathcal{U}}(0; (g_k^{\varepsilon_k})_{\mathcal{V}})$. Compute an approximate $\mathcal{U}$-quasi-Newton step by solving the linear system

$$\nabla_\varepsilon^2 L_{\mathcal{U}}(0; (g_k^{\varepsilon_k})_{\mathcal{V}})\Delta u_k = -\nabla_\varepsilon L_{\mathcal{U}}(0; (g_k^{\varepsilon_k})_{\mathcal{V}}) \qquad (4)$$

for Δu_k. Set $x_{k+1} = x_k + \mathrm{U}_k \, \Delta u_k$ and $\varepsilon_{k+1} = \varepsilon_k$. Increment $k \mapsto k+1$ and go to Step 1.

End algorithm.

Remark 3.11. *In Step 0, the stopping tolerance δ and accuracy tolerance $\varepsilon_{\min}$ can be set to 0. Doing so effectively makes the algorithm run without stopping conditions. This allows for theoretical analysis of the algorithm, but, of course, these values should never be used in practice.*

Remark 3.12. *In Step 1.1, the call to Subroutine 3.3 yields the active set, approximate $\mathcal{U}$-basis and approximate $\mathcal{U}$-gradient in addition to $\tilde{g}_j^{\varepsilon_k}$. However, $\tilde{g}_j^{\varepsilon_k}$ is the only information we use from Subroutine 3.3 in the $\mathcal{V}$-step, so we do not mention the other outputs in the statement of the algorithm.*

Remark 3.13. *The ε_k and the iteration counter k are updated in Step 3 and again in Step 4 (if applicable). This is explained by the fact that Step 4 is not called at every iteration. An alternate formatting of the algorithm might have at the start of each iteration a decision on whether to do a $\mathcal{V}$-step or a $\mathcal{U}$-step. $\mathcal{V}$-step iterations are frequent and can occur multiple times in a row. This is captured in Step 3. $\mathcal{U}$-step iterations only occur after successful $\mathcal{V}$-steps and only in batches of one (i.e., a $\mathcal{U}$-step is never followed by another $\mathcal{U}$-step). This is captured in Step 4.*

3.2 Theoretical comparison to other DFO methods

Relative to other DFO methods, the **DFO $\mathcal{VU}$-algorithm** falls under the category of a model-based method [9, Part 4]. In this case, it uses function calls to construct a model of the objective function and then applies a $\mathcal{VU}$-style method to the model function.

Most other DFO methods for nonsmooth optimization fall under the catergory of direct search methods [9, Part 3]. Direct search methods work by setting an incumbent solution and then polling around it to seek a point that provides a better function value. If improvement is found, then the incumbent solution is updated. Otherwise, the algorithm reduces the polling radius and repeats. If polling is done carefully, then convergence to a critical point can be proved, even for nonsmooth functions [9, Chpt 7]. These ideas are the core of the Mesh Adaptive Direct Search (MADS) algorithm developed in [1, 6, 8] (among other papers). We mention the MADS algorithm, as we use it as one basis of comparison in Section 5.

A few derivative-free model-based methods for nonsmooth optimization have arisen in the last decade. The first such approach appeared in 2008, in the work of Bagirov, Karasösen and Sezer [10]. The method proceeds by constructing a large number of approximate gradients at points near the incumbent solution and using them to build an approximation of the subdifferential. The approximate subdifferential is then used to drive a conjugate subgradient style algorithm. This method was implemented and tested under the name DGM. The authors show the method can achieve four digits of accuracy, but do not include information on the number of function evaluations used, nor provide software. As such, direct comparison to this method is not possible.

A similar idea was proposed by Kiwiel [39]. In Kiwiel's approach, a large number of approximate gradients is used to construct an approximate subdifferential, which in turn is used in a gradient sampling style algorithm. Only a theoretical development of this algorithm was presented.

In relation to [10] and [39], the algorithm herein also generates a collection of gradient approximations and uses them to construct nonsmooth first-order objects. However, the algorithm herein uses the grey-box structure of the problem to control the construction of these approximation gradients. In particular, the number of approximate gradients constructed at an incumbent solution is guided by the number of active indices at that point. Furthermore, the algorithm herein uses the approximate gradients to approximate both subdifferentials and $\mathcal{VU}$-structure in the problem. This sets our algorithm distinctly apart from these previous works.

Between direct-search methods and model-based methods lies the implicit filtering approach of Kelley [36]. The implicit filtering approach can be thought of as beginning with a direct-search poll step, but if success occurs, then instead of simply accepting the new point, the poll information is used to construct approximate gradients and a line search is applied to seek improvement. Convergence of the implicit filtering algorithm is based on a (locally) smooth objective function.

4 Convergence

In this section, we examine the convergence of the DFO $\mathcal{VU}$-algorithm, starting with the $\mathcal{V}$-step. By [26, Corollary 4.6], if the $\mathcal{V}$-step never terminates, then

$$\lim_{j\to\infty} \|z_{j+1} - z_j\| = 0.$$

Then [26, Theorem 4.9] states that if f is locally K-Lipschitz (which a finite-max function is), then

$$\|z_{j+1} - z_j\| \leq \frac{\varepsilon_k^2}{r(K + 2\varepsilon_k)} \Rightarrow f(z_{j+1}) - \varphi_j^{\varepsilon_k}(z_{j+1}) \leq \frac{\varepsilon_k^2}{r} \tag{5}$$

and the routine terminates. The properties of $\varphi_j^{\varepsilon_k}$ established in [26, Lemma 4.1] show that if the $\mathcal{V}$-step with input z_0 stops at iteration j and outputs z_{j+1}, then

$$\operatorname{dist}(\operatorname{Prox}_f^r(z_0), z_{j+1}) \leq \frac{(\mu + 1)\varepsilon_k}{r},$$

where μ is the constant of Theorem 3.6. Now in order to prove the convergence of the main algorithm, we show that either the algorithm terminates in a finite number of steps or, in the case where no stopping occurs, $\varepsilon_k \to 0$ and $\liminf \|s_k\| \to 0$. In either case, we arrive at a good approximation of the minimizer of f. To accomplish that goal, we need the following definitions.

Definition 4.1. *Let $\varepsilon \geq 0$. The* ε-directional derivative *of f at x in direction d is defined*

$$f'_\varepsilon(x; d) = \inf_{t>0} \frac{f(x+td) - f(x) + \varepsilon}{t} = \max_{s \in \partial_\varepsilon f(x)} \{s^\top d\}.$$

Definition 4.2. *Let $\varepsilon, \eta \geq 0$. A vector v is an* (ε, η)-subgradient *of f at $\bar{x}$, denoted $v \in \partial_\varepsilon^\eta f(\bar{x})$, if for all x,*

$$f(x) \geq f(\bar{x}) + v^\top(x - \bar{x}) - \eta\|x - \bar{x}\| - \varepsilon.$$

Notice that by setting $\eta = 0$ we recover the definition of the ε-subgradient and by setting $\varepsilon = \eta = 0$ we have the convex analysis subgradient. The next lemma provides enlightenment on the (ε, η)-subgradient in the general case.

Lemma 4.3. *Let $\varepsilon, \eta \geq 0$ and f be convex with $\bar{x} \in \operatorname{dom} f$. Then*

$$g \in \partial_\varepsilon^\eta f(\bar{x}) \Leftrightarrow g \in \partial_\varepsilon f(\bar{x}) + B_\eta. \tag{6}$$

Proof. ($\Rightarrow$) Suppose $g \in \partial_\varepsilon^\eta f(\bar{x})$. Since $\partial_\varepsilon f$ is closed and convex [31, Theorem 1.1.4], we define

$$\bar{g} = \operatorname{Proj}_{\partial_\varepsilon f(\bar{x})}(g)$$

and we have $\bar{g} \in \partial_\varepsilon f(\bar{x})$. Set $v = g - \bar{g}$, so that $g = \bar{g} + v$, and for $t > 0$ we use $x = \bar{x} + tv$ in the definition of the (ε, η)-subgradient:

$$\begin{aligned}
f(\bar{x} + tv) &\geq f(\bar{x}) + (\bar{g} + v)^\top tv - \eta\|tv\| - \varepsilon, \\
\frac{f(\bar{x} + tv) - f(\bar{x}) + \varepsilon}{t} - v^\top \bar{g} &\geq \|v\|^2 - \eta\|v\|, \\
\inf_{t>0} \frac{f(\bar{x} + tv) - f(\bar{x}) + \varepsilon}{t} - v^\top \bar{g} &\geq \|v\|^2 - \eta\|v\|, \\
f'_{\varepsilon_1}(\bar{x}; v) - v^\top \bar{g} &\geq \|v\|^2 - \eta\|v\|. \qquad (7)
\end{aligned}$$

By the Projection Theorem, we have

$$p = \operatorname{Proj}_{\partial_\varepsilon f(\bar{x})} y \Leftrightarrow (y - p)^\top (z - p) \leq 0 \text{ for all } z \in \partial_\varepsilon f(\bar{x}).$$

So for all $\tilde{g} \in \partial_\varepsilon f(\bar{x})$ we have

$$\begin{aligned}(g - \bar{g})^\top (\tilde{g} - \bar{g}) &\leq 0,\\ v^\top \tilde{g} &\leq v^\top \bar{g}.\end{aligned}$$

Hence,

$$v^\top \bar{g} = \sup_{\tilde{g} \in \partial_\varepsilon f(\bar{x})} \{v^\top \tilde{g}\}.$$

Using this together with Definition 4.1, (7) becomes

$$\begin{aligned}\|v\|^2 - \eta\|v\| &\leq 0,\\ \|v\| &\leq \eta.\end{aligned}$$

Therefore, $v \in B_\eta$, and we have $g = \bar{g} + v \in \partial_\varepsilon f(\bar{x}) + B_\eta$.
($\Leftarrow$) Suppose that $g \in \partial_\varepsilon f(\bar{x}) + B_\eta$. Set $g = \bar{g} + v$ where $\bar{g} \in \partial_\varepsilon f(\bar{x})$ and $v \in B_\eta$. Then, by the definition of ε-subgradient and the Cauchy-Schwarz inequality, we have

$$\begin{aligned}f(x) - f(\bar{x}) - g^\top (x - \bar{x}) &= f(x) - f(\bar{x}) - \bar{g}^\top (x - \bar{x}) - v^\top (x - \bar{x}),\\ &\geq -\varepsilon - v^\top (x - \bar{x}),\\ &\geq -\varepsilon - \|v\|\|x - \bar{x}\|,\\ &\geq -\varepsilon - \eta\|x - \bar{x}\|.\end{aligned}$$

Therefore, $g \in \partial_\varepsilon^\eta f(\bar{x})$. □

Now we are ready to show that the inexact aggregate subgradient at any step is a good approximation of a true subgradient.

Lemma 4.4. *Let f satisfy Assumptions 1.4 and 1.6. Let K be the Lipschitz constant of f. If* DFO-VU *at iteration k gives output (x_{k+1}, s_{k+1}), then*

$$s_{k+1} \in \partial_{\frac{\varepsilon_k^2}{r}}^{\varepsilon_k K} f(x_{k+1}).$$

Proof. In [26, (4.3)], it is shown that

$$f(x) + \varepsilon_k K \|\hat{M}^{-1}\| \|x - x_{k+1}\| \geq \varphi_j^{\varepsilon_k}(x_{k+1}) + s_{k+1}^\top (x - x_{k+1}).$$

Remark 3.4 shows that $\|\hat{M}^{-1}\| = 1$. Since iteration k has completed, the stopping test in Step 1.4 has passed[1], thus

$$\varphi_j^{\varepsilon_k}(x_{k+1}) - f(x_{k+1}) \geq -\frac{\varepsilon_k^2}{r}.$$

This implies

$$\begin{aligned} f(x) &\geq \varphi_j^{\varepsilon_k}(x_{k+1}) - f(x_{k+1}) + f(x_{k+1}) + s_{k+1}^\top (x - x_{k+1}) - \varepsilon_k K\|x - x_{k+1}\| \\ &\geq -\frac{\varepsilon_k^2}{r} + f(x_{k+1}) + s_{k+1}^\top (x - x_{k+1}) - \varepsilon_k K\|x - x_{k+1}\|. \end{aligned}$$

Thus, $s_{k+1} \in \partial_{\frac{\varepsilon_k^2}{r}}^{\varepsilon_k K} f(x_{k+1})$ by Definition 4.2. $\square$

There are two special cases of Lemma 4.4 that are of interest; we consider what happens when the aggregate subgradient is zero and when the maximum subgradient error is zero.

Corollary 4.5. *Let f satisfy Assumptions 1.4 and 1.6. Let K be the Lipschitz constant of f. If at iteration k* DFO-VU *gives output (x_{k+1}, s_{k+1}), then the following hold.*

(i) *If $s_{k+1} = 0$, then $0 \in \partial_{\frac{\varepsilon_k^2}{r}} f(x_{k+1}) + B_{K\varepsilon_k}$, and by Lemma 4.4 we have that for all $x \in \mathbb{R}^n$,*

$$f(x) \geq f(x_{k+1}) - \varepsilon_k K\|x - x_{k+1}\| - \frac{\varepsilon_k^2}{r}.$$

(ii) *If $\varepsilon_k = s_{k+1} = 0$, then $0 \in \partial f(x_{k+1})$ and x_{k+1} is a minimizer of f.*

Remark 4.6. *Since $\varepsilon_k = 0$ can only occur if $\varepsilon_0 = 0$, item (ii) could alternately be stated using "If $\varepsilon_0 = s_{k+1} = 0$".*

Now we need to consider the possibility that the algorithm does not terminate and what the effect would be. We begin with the scenario where an infinite number of serious steps is taken.

[1]Recall that [26, Corollary 4.6] and [26, Theorem 4.9] ensure this happens in finite time.

Theorem 4.7. *Let f satisfy Assumptions 1.4, 1.5, and 1.6. Suppose* DFO-VU *is run without stopping conditions (i.e., $\delta = \varepsilon_{\min} = 0$). If there is an infinite number of serious steps taken in Step 3, then $\varepsilon_k \to 0$ and*

$$\liminf_{k\to\infty} \|s_k\| = 0.$$

Proof. Note that f is bounded below, due to Assumption 1.5. Suppose that out of the infinite number of serious steps, $\|s_{k+1}\|^2$ is bounded away from 0. That is, suppose there exists $\hat{\delta} > 0$ such that $\|s_{k+1}\|^2 > \hat{\delta}$ whenever $f(x_k) - f(x_{k+1}) \geq \frac{m}{2r}\|s_{k+1}\|^2$, and $f(x_k) - f(x_{k+1}) \geq \frac{m}{2r}\|s_{k+1}\|^2$ occurs an infinite number of times. Then we have

$$f(x_k) - f(x_{k+1}) \geq \frac{m}{2r}\|s_{k+1}\|^2 > \frac{m\hat{\delta}}{2r}$$

an infinite number of times. Since $\frac{m\hat{\delta}}{2r}$ is constant, we have

$$\lim_{k\to\infty} [f(x_0) - f(x_k)] = \infty,$$

which contradicts the fact that f is bounded below. Hence, eventually $\|s_{k+1}\|^2 \leq \hat{\delta}$, so $\liminf_{k\to\infty} \|s_k\| = 0$.

Since we are supposing that the algorithm does not stop, we must have $\varepsilon_k > \varepsilon_{\mathtt{min}} = 0$ and we set $\varepsilon_{k+1} = \varepsilon_k/2$. This happens an infinite number of times, which gives $\varepsilon_k \to 0$. □

Next comes the scenario where a finite number of serious steps is taken, yet the algorithm does not terminate.

Lemma 4.8. *Let f satisfy Assumptions 1.4, 1.5, and 1.6. Suppose* DFO-VU *is run without stopping conditions (i.e., $\delta = \varepsilon_{\min} = 0$). If there is a finite number of serious steps taken in Step 3, then for all k sufficiently large,*

$$\varepsilon_k > \left(1 - \frac{m}{2}\right)^{1/2} \|s_{k+1}\|. \tag{8}$$

Proof. Let $\bar{k}$ be the final iteration where a serious step occurs, so that a null step occurs at every $k > \bar{k}$. Since $s_{k+1} = r(x_k - x_{k+1})$ is the aggregate subgradient of the model function $\varphi_j^{\varepsilon_k}$ at $z_{j+1} = x_{k+1}$, we have $s_{k+1} \in \partial\varphi_j^{\varepsilon_k}(x_{k+1})$ [26, Lemma 4.1(c)]. Thus,

$$\varphi_j^{\varepsilon_k}(x) \geq \varphi_j^{\varepsilon_k}(x_{k+1}) + s_{k+1}^\top(x - x_{k+1}) \text{ for all } x.$$

By the tilt-correction (E_j in Step 1.1), we have that at x_k,

$$\begin{aligned}
f(x_k) &\geq \varphi_j^{\varepsilon_k}(x_{k+1}) + s_{k+1}^\top(x_k - x_{k+1}) \text{ [26, Lemma 4.1(b)]},\\
&= \varphi_j^{\varepsilon_k}(x_{k+1}) + \frac{1}{r}s_{k+1}^\top[r(x_k - x_{k+1})],\\
&= \varphi_j^{\varepsilon_k}(x_{k+1}) + \frac{1}{r}\|s_{k+1}\|^2,\\
&= \varphi_j^{\varepsilon_k}(x_{k+1}) - f(x_{k+1}) + f(x_{k+1}) + \frac{1}{r}\|s_{k+1}\|^2,\\
f(x_k) - f(x_{k+1}) &\geq \varphi_j^{\varepsilon_k}(x_{k+1}) - f(x_{k+1}) + \frac{1}{r}\|s_{k+1}\|^2. \qquad (9)
\end{aligned}$$

By the stopping test in Step 1.4, we have

$$\varphi_j^{\varepsilon_k}(x_{k+1}) - f(x_{k+1}) \geq -\frac{\varepsilon_k^2}{r}. \qquad (10)$$

Combining (9) and (10) yields

$$f(x_k) - f(x_{k+1}) \geq \frac{1}{r}\|s_{k+1}\|^2 - \frac{\varepsilon_k^2}{r}. \qquad (11)$$

Then for all $k > \bar{k}$, by Step 3(iii) and (11) we have

$$\begin{aligned}
\frac{m}{2r}\|s_{k+1}\|^2 &> f(x_k) - f(x_{k+1}),\\
\frac{m}{2r}\|s_{k+1}\|^2 &> \frac{1}{r}\|s_{k+1}\|^2 - \frac{\varepsilon_k^2}{r},\\
\varepsilon_k^2 &> \left(1 - \frac{m}{2}\right)\|s_{k+1}\|^2,
\end{aligned}$$

and (8) is proved. □

Corollary 4.9. *Let f satisfy Assumptions 1.4, 1.5, and 1.6. Suppose* DFO-VU *is run without stopping conditions. If there is a finite number of serious steps taken in Step 3, then $\varepsilon_k \to 0$ and $\|s_k\| \to 0$.*

Proof. Since there is a finite number of successes and the algorithm does not terminate, there is an infinite number of failures. By Step 3(iii), $\varepsilon_k \to 0$. By (8), $\|s_k\| \to 0$. □

Notice, if $\delta = \varepsilon_{\texttt{min}} = 0$, then by Step 2 of the algorithm we see that the only way it will terminate is if $s_{k+1} = 0$ and $\varepsilon_k = 0$. Since $\varepsilon_k > 0$, this cannot occur. (If it could occur, then Corollary 4.5 would imply x_{k+1} is a minimizer of f.)
Theorem 4.10 below unites the convergence results of this section.

Theorem 4.10. *Let f satisfy Assumptions 1.4, 1.5, and 1.6. Suppose* DFO-VU *is run on f and generates the sequence $\{x_k\}$. Then either the algorithm terminates at some iteration $\bar{k}$ with $\|s_{\bar{k}+1}\| \leq \sqrt{\delta}$ and $\varepsilon_{\bar{k}} \leq \varepsilon_{\texttt{min}}$, or* $\liminf_{k\to\infty} \|s_k\| = 0$ *and $\varepsilon_k \to 0$. In the latter case, any cluster point $\bar{x}$ of a subsequence $\{x_{k_j}\}$ such that $s_{k_j} \to 0$ satisfies $0 \in \partial f(\bar{x})$.*

Proof. If the algorithm terminates at iteration $\bar{k}$, by Step 2 we have $\|s_{\bar{k}+1}\| \leq \sqrt{\delta}$ and $\varepsilon_{\bar{k}} \leq \varepsilon_{\texttt{min}}$. Suppose the algorithm does not terminate. If there is an infinite number of serious steps taken, then $\liminf_{k\to\infty} \|s_k\| = 0$ and $\varepsilon_k \to 0$ by Theorem 4.7. If there is a finite number of serious steps taken, then $s_k \to 0$ and $\varepsilon_k \to 0$ by Corollary 4.9. In either case, consider any subsequence $\{x_{k_j}\}$ with cluster point $\bar{x}$ such that $s_{k_j} \to 0$. Since $\varepsilon_k \to 0$, we have $\varepsilon_{k_j} \to 0$. By [4, Proposition 5], we have that the $(\varepsilon_{k_j}^2/r)$-subdifferential of f is continuous jointly as a function of (x, ε_{k_j}) on $(\operatorname{ri}\operatorname{dom} f) \times (0, \infty)$. Since $B_{K\varepsilon_{k_j}}$ is continuous as well, by [51, Proposition 3.2.7-3] and (6) we have that $\partial_{\varepsilon_{k_j}^2/r}^{\varepsilon_{k_j}K}$ is continuous. Therefore, since $s_{k_j+1} \in \partial_{\varepsilon_{k_j}^2/r}^{\varepsilon_{k_j}K} f(x_{k_j+1})$ by Lemma 4.4, $s_{k_j+1} \to 0$, $\varepsilon_{k_j} \to 0$ and $x_{k_j+1} \to \bar{x}$, we have $0 \in \partial f(\bar{x})$. □

5 Numerical Results

We now present several tests run on a 3.2 GHz Intel Core i5 processor with a 64-bit operating system, using MATLAB version 9.3.0.713579 (R2017b). Our testing includes the following algorithms.

1. DFO-VU using the default of $n+1$ function calls per $\mathcal{U}$-gradient approximation and $2n+1$ function calls per $\mathcal{U}$-Hessian approximation;

2. InexBun, an inexact bundle method along the lines of [54], with access to the grey-box available to DFO-VU: the value function is exact and the subgradient is approximated by means of the DFO approach in Subroutine 3.3;

3. ExBun, a classical bundle method in proximal form [12, Part II];

4. CompBun, the Composite Bundle method from [59];

5. Nomad version 3.9.1, a well-established DFO method from [41].

6. RAGS, the DFO method from [24].

Algorithms DFO-VU, InexBun, ExBun, and CompBun are bundle algorithms, while Nomad and RAGS are DFO solvers. Algorithms ExBun and CompBun use *exact subgradient information.* As such, we expect those solvers to outperform both DFO-VU and InexBun. These inexact variants are on equal ground and we expect to see a positive impact of the $\mathcal{VU}$-decomposition in terms of accuracy.

We use different groups of test problems, refered below as the MQ and NK families. The MQ-family used in the bundle and DF benchmarks is formed by convex, maximum-of-quadratics, functions satisfying all the Assumptions 1.4-1.6; see (12). The NK-functions, extracted from the collection in [33] and used in Section 5.3, may not satisfy Assumption 1.6. A last group with nonconvex MQ-functions is employed in Section 5.5. Testing with the latter set was done solely for prospective illustrative purposes, since the convergence of DFO-VU for nonconvex functions is beyond the scope of this work. As such, for the nonconvex problems we only examine the behaviour of DFO-VU.

5.1 Convex test functions and benchmark rules for the bundle solvers

In this test, for the purpose of comparing DFO-VU with ExBun, InexBun and CompBun, we considered 301 max-of-quadratics functions. The first one is the classical maxquad function in nonsmooth optimization [12, Part II], for which the dimension is $n = 10$, the optimal value is $\bar{f} = -0.841408334596418 14$, and $\dim \mathcal{V}$ at a solution is equal to 3. The remaining 300 problems were generated randomly in dimensions $n \in \{10, 20, 30, 40, 50\}$. Each problem is generated such that the minimizer is $\bar{x} = 0 \in \mathbb{R}^n$ with $\bar{f} = 0$. The problems are designed with various final $\mathcal{V}$-dimensions $\dim \mathcal{V} \in \{\lfloor 0.25n \rfloor, 0.5n, \lfloor 0.75n \rfloor\}$. The functions were generated as follows; given $m \geq$

$|A(\bar{x})| = \dim \mathcal{V} + 1$,

$$f(x) = \max_{j \in \{1,2,\dots,m\}} \left\{ \frac{1}{2} x^\top H_j x + b_j^\top x \right\}, \tag{12}$$

for random $H_j \in S_+^n$ and $b_j \in \mathbb{R}^n$. The symmetric, positive semidefinite matrices H_j have condition number equal to $(\operatorname{rank} H)^2 = (\dim \mathcal{V})^2$, and the set of vectors $\{b_2 - b_1, \dots, b_{\dim \mathcal{V}+1} - b_1\}$ is linearly independent. The above setting guarantees that all the assumptions in Section 1.2 hold for the considered instances.

We must acknowledge and accept that some of the inner workings of each solver make it difficult to compare the results fairly. First, CompBun and ExBun make blackbox-calls (bb-calls) that yield exact values for the function and a subgradient, while InexBun and DFO-VU call a grey-box that yields exact function values and approximate subgradients. Second, to avoid machine error due to a near-singular matrix in the second-order approximation created in Subroutine 3.9, DFO-VU stops when in Step 4 the parameter ε_k becomes smaller than 10^{-5}. Third, InexBun stops when there are more than 18 consecutive noise-attenuation steps; we refer the reader to [54] for details. Barring the above, the parameters for CompBun, ExBun, and InexBun are those chosen for the Composite Bundle solver in [59]. In an effort to make the comparisons as fair as possible, we adopted the following rules.

1. All solvers use the same quadratic programming built-in MATLAB solver, `quadprog`.

2. For all solvers, the stopping tolerance was set to 10^{-2}, which for DFO-VU means that in Step 2, $\delta = \varepsilon_{\min} = 10^{-2}$.

3. The maximum number of bb-calls was set to $\max_{\mathtt{S}} = 800 \min(n, 20)$. This corresponds to function and subgradient evaluations for the exact variants and to function evaluations for the inexact variants.

4. For all solvers, a run was declared a failure when $\max_{\mathtt{S}}$ was reached or when there was an error in the QP solver.

5. The methods use the same starting points, with components randomly drawn in $[-1, 1]$. We ran all the instances with two starting points, for a total of 602 runs.

For those readers interested in implementing DFO-VU, we mention the following additional numerical tweaks that had a positive impact in the algorithm's performance.

1. In the $\mathcal{U}$-step, finding the active index set $A(x_k)$ in Subroutine 3.3 is tricky. We note that using an absolute criterion ($f_i(x_k) = f(x_k)$) was worse than the following soft-thresholding test:
$$i \in A(x_k) \text{ when } |f(x_k) - f_i(x_k)| \leq 0.001|f(x_k)| \,. \tag{13}$$

2. In Step 1.3, it is often preferable to calculate the proximal point z_{j+1} by solving the dual of the quadratic programming problem defining $\text{Prox}^r_{\varphi_j^{\varepsilon_k}}(z_0)$.

3. The tilting of gradients in (3) is done only when E_j is larger than 10^{-8}. Otherwise, we set $g_j^{\varepsilon_k} = \tilde{g}_j^{\varepsilon_k}$.

4. As long as the proximal parameter remains uniformly bounded, it can vary along iterations without impairing the convergence results. We have found the following rule to be effective and use it in our testing,
$$t_k = \begin{cases} \frac{0.5|g(x_k)|^2}{1+|f(x_k)|}, & \text{if } |f(x_k)| > 10^{-10}, \\ 2, & \text{otherwise,} \end{cases}$$
and let
$$r_k = \max\left\{1, \min\left\{\frac{1}{t_k}, 100r_{k-1}, 10^6\right\}\right\} \,.$$

5. In Step 2.5, the new bundle $\mathcal{B}_{j+1}$ keeps *almost active* indices. As can be seen from (13), we accept as active the subfunctions that are close to active at each iteration point, so as not to dismiss those that are active but do not quite appear to be so because of numerical error.

5.2 Benchmark of bundle solvers

We first describe the indicators defined to compare the solvers. The number of iterations is not a meaningful measure for comparison, because each solver makes a very different computational effort per iteration. This depends not only on the solver, but also on how many evaluations are done per iteration.

Moreover, since the exact variants do not spend calls to make the DFO subgradient approximation, neither the total solving time nor the number of bb-calls are meaningful measures. As the optimal values $\bar{f}$ are known for the considered instances, we compare the accuracy reached by each solver. Denoting the best function value of the analyzed case by $f^{\texttt{found}}$,

$$RA = \max\left(0, \min\left(16, -\log\left(\frac{f^{\texttt{best}} - \bar{f}}{f(x^0) - \bar{f}}\right)\right)\right), \tag{14}$$

is the number of digits of accuracy achieved by the solver. We also analyze the ability of each solver in capturing the (known) exact $\mathcal{V}$-dimension, by looking at the cardinality of $A(x^{\texttt{found}})$ as in (13), for $x^{\texttt{found}}$ the final point found by each solver, and computing $v^{\texttt{found}} = |A(x^{\texttt{found}})| - 1$.

Since MAXQUAD is a well-known test function for bundle methods, in Table 1 we report separately the measures obtained for this function, running the four solvers with two starting points.

Table 1: Results for MAXQUAD test function, $\dim \mathcal{V}(\bar{x}) = 3$.

		CompBun	ExBun	InexBun	DFO-VU
First x_0	RA	5	2	1	3
	$v^{\texttt{found}}$	3	1	1	3
Second x_0	RA	5	2	1	3
	$v^{\texttt{found}}$	3	0	1	2

DFO-VU performs very well, both in terms of accuracy and $\mathcal{V}$-dimension, which is underestimated in the second run. Such underestimation means that DFO-VU is taking $\mathcal{U}$-steps in a larger subspace. Of course, the price to be paid (especially with our rudimentary implementation) is computing time, which passes from a few seconds with CompBun-InexBun, to 2 minutes with DFO-VU.

The solver performance for the remaining 600 runs was similar. For each problem and the two random starting points, we organized the output into five groups, corresponding to increasing percentages of the $\mathcal{V}$-dimension at $\bar{x}$ with respect to n. Each row in Table 2 reports for each solver the mean value of the digits of accuracy, averaged for each group. The bottom line in Table 2 contains the total number of instances considered for the test and the total average values for RA.

Table 2: Average RA for 602 (MAXQUAD and 300 random problems, each with 2 starting points) runs.

$\dim \mathcal{V}(\bar{x})$	# of runs	CompBun	ExBun	InexBun	DFO-VU
$\in (0\%, 15\%)n$	96	3.99	0.78	0.58	1.44
$\in [15\%, 30\%)n$	182	4.79	1.12	0.89	1.63
$\in [30\%, 45\%)n$	134	3.93	0.91	0.61	1.05
$\in [45\%, 60\%)n$	106	4.21	0.96	0.62	1.16
$\in [60\%, 100\%)n$	84	5.75	1.36	1.07	2.15
$\in (0\%, 100\%)n$	602	4.50	1.02	0.76	1.46

As conjectured, in terms of accuracy on the optimal value, CompBun is far superior to all the other variants. The inexact bundle method InexBun performs reasonably well, but is systematically outperformed by DFO-VU. An interesting feature is that, in spite of using only approximate subgradient information, DFO-VU achieves better function accuracy than the exact classical bundle method, ExBun. This fact confirms the interest of exploiting available structure in the bundle method, even if the information is inexact.

Table 3 below gives another indication of the performance of DFO-VU and InexBun in predicting the dimension of the $\mathcal{V}$-space. Out of the 602 runs, we list the number of times that each algorithm returned the exact $\mathcal{V}$-dimension, the number of times $v^{\texttt{found}}$ was within 1, 2 or 5, and the number of times it was more than 5 away from the correct $\mathcal{V}$-dimension.

Table 3: The $\mathcal{V}$-dimension prediction comparison between the inexact solvers.

	# Exact	# ± 1	# ± 2	# ± 5	# > 5
InexBun	161 (27%)	351 (58%)	441 (73%)	528 (88%)	74 (12%)
DFO-VU	408 (68%)	486 (81%)	513 (85%)	551 (92%)	51 (8%)

In almost 70% of the runs, DFO-VU correctly predicted the $\mathcal{V}$-dimension, more than double what InexBun was able to do. This is a strong indicator that DFO-VU is able to do what it is meant to do in that respect; InexBun is not meant to make this prediction, so we expect to see the results that we have.

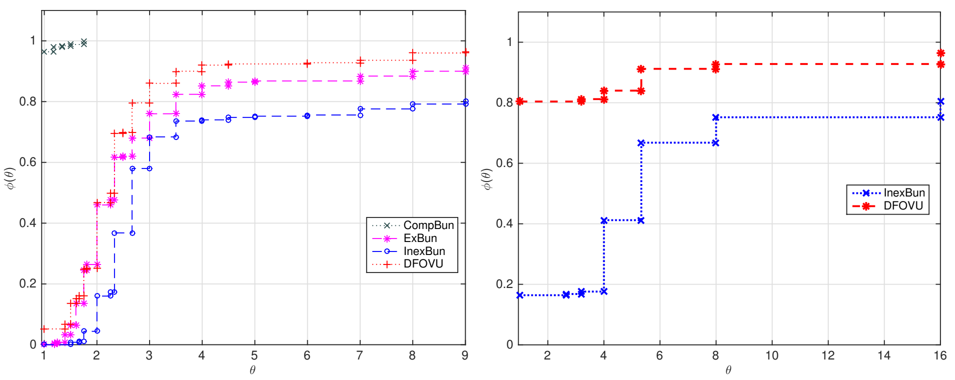


Figure 1: Accuracy Profile: reciprocal of accuracy, all solvers (left) and solvers InexBun and DFO-VU(right).

In order to interpret the output graphically, we created profiles for the accuracy over the full set of 602 instances, see Figure 1. In the graph, each curve represents the cumulative probability distribution $\phi_s(\theta)$ of the resource "f-accuracy", measured in terms of the reciprocal of RA. The use of 1/RA as an indicator stems from the fact that usually smaller values of the abscissa θ mean better performance of the resource. As in our case higher accuracy is preferred, we invert the relation to plot the profile. In this manner, in the two profiles that follow, the solvers with the highest curves are the best ones for the given indicator of performance. For the left endpoint $\theta = \theta_{\min}$ in the graph, the probability $\phi_s(\theta_{\min})$ of a particular solver is the probability that the solver will provide the highest accuracy among all algorithms. Looking at the highest value for the left endpoint in the left plot in Figure 1, we conclude that the most precise solver is CompBun in all of the runs, as expected. The DFO-VU solver is the second-best, followed by ExBun.

In general, for a particular solver s, the ordinate $\phi_s(\theta)$ gives information on the percentage of problems that the considered method will solve if given θ times the resource employed by the best one. Looking at the value of $\theta = 3$, we see that DFO-VU solves about 85% of the 602 problems ($\phi(3) > 0.8$) with a third (=$1/\theta$) of the accuracy obtained by CompBun, while InexBun solves less than 70% ($\phi(3) < 0.7$).

Considering that the comparison with exact variants is not entirely fair, we repeat the profile, this time comparing only InexBun and DFO-VU. The values of $\theta = 1$ in the right plot in Figure 1 show that InexBun was more accurate than DFO-VU in fewer than 20% of the runs ($\phi(1) < 0.2$).

We now comment on CPU time, function evaluations and failures of bundle solvers. Naturally, the gain in accuracy of DFO-VU comes at the price of CPU time. As expected, the fastest solver in all of the runs is CompBun, followed by ExBun, InexBun, and DFO-VU. The average CPU time in seconds was 0.47 for CompBun, 0.28 for ExBun, 0.40 for InexBun, and 61 for DFO-VU. The time increase for DFO-VU is better understood when examining the respective average number of calls to the oracle, equal to 8 for CompBun, 26 for ExBun, 504 for InexBun, and 52330 for DFO-VU. There is a factor of close to 20 when passing from ExBun to InexBun, whose only difference is in the use of the inexact (simplex) gradients. The factor of 100 between the oracle calls required by InexBun and those required by DFO-VU is explained by the fact that DFO-VU approximates not only the gradient, but also the $\mathcal{U}$-Hessian. Such an increase is not a surprise, as our implementation of DFO-VU is not optimized and the computational burden required by DFO-VU is much higher than that required by InexBun. We comment on possible numerical enhancements in this regard in Section 6.

Regarding failures, there was none for CompBun, ExBun and InexBun, whose respective stopping tests were triggered in all 602 runs. DFO-VU failed 104 times having reached the maximum number of allowed evaluations ($\max_{\mathtt{S}}$), and twice when the parameter ε_k became unduly small. This figure represents 17.5% of all the runs. Most of the failures of DFO-VU by $\max_{\mathtt{S}}$ remained even after increasing $\max_{\mathtt{S}}$ by a factor of 10. It is our understanding that the method reached its limit of accuracy in those instances, which likely had worse conditioning and were too difficult to solve with our inexact method. By constrast, a close observation of failures of DFO-VU in previous runs that were due to a small ε_k gave us some hints for improvement of the algorithm's performance. We noticed that when ε_k becomes too small, the stopping test in Step 1.4 becomes hard to attain and the $\mathcal{V}$-step becomes dismayingly slow. It is important to tune the manner in which ε_k decreases, so that the reduction is not done too fast. For our experiments, we update ε_k in Steps 3.1 and 3.3 of DFO-VU by $\varepsilon_{k+1} = 0.9\varepsilon_k$. This appeared a reasonable setting for the considered 602 instances. These precautions help to ensure that the solution to (4) does not become near-singular.

We finish by noting that ExBun, the classical bundle method, is extremely reliable, but neither as accurate nor as fast as CompBun, which fully exploits structure of composite functions and uses exact gradient information. Of the four solvers, if the gradient evaluations can be done exactly, CompBun is to be preferred. Otherwise, DFO-VU seems a good option for cases when accuracy of the solution is a more important concern than solving time.

5.3 DFO solvers

Having analyzed the qualities and weaknesses of DFO-VU with respect to other bundle methods, we now examine the behaviour of DFO-VU when compared to established DFO solvers Nomad [41] and RAGS [24]. Nomad is a program that uses as its basis the MADS (Mesh Adaptive Direct Search) algorithm [7]. In the MADS algorithm, trial points on a mesh are evaluated and the mesh size for the next iteration is adjusted according to the findings of the current one. We refer the reader to [7, 41] for details on the implementation of Nomad and the structure of the MADS algorithm. RAGS is a DFO solver built specifically for finite-max problems (convex and nonconvex). It is an approximate gradient descent method that generates an approximate subdifferential and a search direction at every iteration, followed by a line search. We refer the reader to [24] for details on implementation of the RAGS method.

Comparisons are done on the MQ and NK test-functions. The MQ family is the same as the one used in the previous section, comparing bundle solvers. There are 614 runs, corresponding to two random starting points for each of 301 functions and 5-10 starting points for a couple of the smaller-dimension functions.

The NK family includes those unconstrained convex problems in [33] whose objective function that can be written as a finite-max function:

1. Generalization of MAXQ
$$f(x) = \max\left\{x_i^2 : i = 1, \ldots, n\right\} .$$

2. Generalization of MXHILB
$$f(x) = \max\left\{\left|\sum_{j=1}^{n} \frac{x_j}{i+j-1}\right| : j = 1, \ldots, n\right\} .$$

3. Short Chained LQ, converting the original Chained LQ in [33]
$$f(x) = \sum_{i=1}^{n-1} \max\left\{-x_i - x_{i+1}, -x_i - x_{i+1} + x_i^2 + x_{i+1}^2 - 1\right\} .$$

to a new version without the outside max-operation:

$$f(x) = \max\left\{-x_i - x_{i+1}, -x_i - x_{i+1} + x_i^2 + x_{i+1}^2 - 1 : i = 1, \dots, n-1\right\} .$$

4. Long Chained LQ, adjusting the original function to a finite-max format with 2^{n-1} terms, as follows. Any function of the form

$$f(x) = \sum_{i=1}^{n-1} \max\{f_{i,1}, f_{i,2}\},$$

which can be rewritten as a purely finite-max function by creating

$$\begin{aligned} f(x) &= \max\{f_{1,1}, f_{1,2}\} + \max\{f_{2,1}, f_{2,2}\} + ... + \max\{f_{n-1,1}, f_{n-1,2}\} \\ &= \max\{f_{1,1} + f_{2,1} + f_{n-1,1},\ f_{1,2} + f_{2,1} + f_{n-1,1}, \\ &\qquad f_{1,1} + f_{2,2} + f_{n-1,1},\ f_{1,2} + f_{2,2} + f_{n-1,1}, ... \\ &\qquad f_{1,2} + f_{2,2} + f_{n-1,n}\}. \end{aligned}$$

The long Chained LQ has 2^{n-1} sub-functions inside of the max-term:

$$f(x) = \max\{f_{1,1}, f_{1,2}, f_{2,1}, f_{2,2}, ..., f_{n-1,2}\}.$$

5. Short Chained CB3 I, converting the original Chained CB3 I in [33]

$$f(x) = \sum_{i=1}^{n-1} \max\left\{x_i^4 + x_{i+1}^2, (2-x_i)^2 + (2-x_{i+1})^2, 2e^{-x_i+x_{i+1}}\right\} .$$

to a version without the outside sum, as in short Chained LQ.

6. Long Chained CB3 I, adjusted to a finite-max format, this time creating 3^{n-1} sub-functions inside of the max-term.

7. Chained CB3 II

$$f(x) = \max\left\{\sum_{i=1}^{n-1} x_i^4 + x_{i+1}^2, \sum_{i=1}^{n-1}(2-x_i)^2 + (2-x_{i+1})^2, \sum_{i=1}^{n-1} 2e^{-x_i+x_{i+1}}\right\} .$$

These seven functions have variable dimension and known optimal value, reported in Table 4. There is a total of 117 runs, corresponding to 2-5 random starting points for each of 40 test functions (the seven functions in different dimensions, $n \in \{10, 20, 30, 40, 50, 100\}$.

Table 4: Optimal values for the NK-family

NK-1	NK-2	NK-3	NK-4	NK-5	NK-6	NK-7
0	0	-$\sqrt{2}$	2	$2(n-1)$	-$\sqrt{2}$	2

5.4 Benckmark of DFO solvers

The benchmark includes runs with different time budgets, allowing the solvers to spend 1, 10, and 60 seconds per test function. In most analyses the results for 10 and 60 second time budgets were very similar. As such, results for 60 second time budgets are only presented when they are notably different than the 10 second time budget.

5.4.1 Solvers' accuracy

Since all the optimal values are known, the performance indicator in the profile is the absolute error, that is, the difference between the final function value of the solver and the optimal value. Figure 2 reports the profiles, gathering all 731 runs. In this first comparison, DFO-VU seems to perform

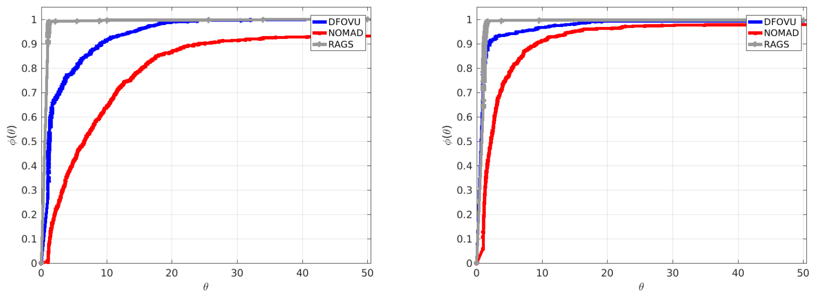


Figure 2: Absolute Error performance profile, MQ and NK functions, 1 second (left) and 10 seconds (right).

better than Nomad, but is outperformed by RAGS.

In order to understand if the solvers behave differently on the two families of test functions, for each solver, time budget and family of test functions we plotted the relative accuracy, given by the value RA in (14).

Figure 3 reports the results for the MQ-family, for the three time budgets. Notice the similarity of the middle and right graphs (10 and 60 seconds, respectively).

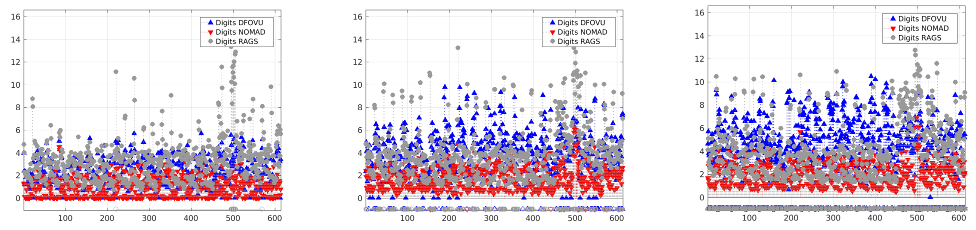


Figure 3: Accuracy, MQ functions, 1 (left), 10 (middle) and 60 (right) seconds. In the bottom line, a △, ▽ or ○ indicates when the stopping test of the corresponding solver was triggered (respectively, DFO-VU, Nomad, RAGS).

Unless the stopping test is triggered, the solvers should achieve more accuracy when allowed to run longer. For the MQ family, the comparison of the triangles (blue) and circles (grey), when moving from the left to the right graph in Figure 3 shows much improvement for DFO-VU and RAGS (for Nomad, in red, the improvement is less significant).

Additional information is given in the bottom line of each graph in Figure 3, reporting the instances at which that solver's stopping test was triggered, with tolerance 10^{-4}. We observe that mostly with DFO-VU and RAGS and with the 10 and 60 second budget, in many instances the solvers stopped. From the figures given in Table 5 below, we see that with the time limit of 1 second, only RAGS' stopping test was triggered, 13 out of 614 times. When given a 10 second budget, DFO-VU, Nomad, and RAGS stopped in 120, 28, and 294 runs, respectively, while when running up to 60 seconds, the respective figures are 432, 139, and 565, amounting to 70%, 23%, and 92% of the total instances.

Table 5 reports the main indicators for the MQ family, discriminating by problem dimension. The first indicator refers to accuracy, averaged to whole numbers. The graphical illustration provided by Figure 3 is confirmed: DFO-VU systematically outperforms Nomad in terms of accuracy.

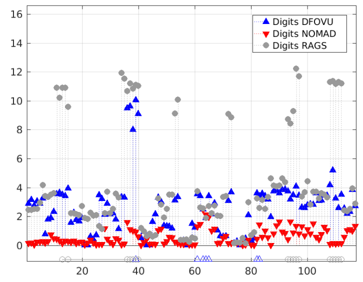
In terms of accuracy for the tested MQ instances, given the time limits of 1 and 10 seconds, the best solver is RAGS, while DFO-VU came out on top with the 60-second time budget. Additionally, with the time budgets of 1 and 10 seconds, as the problem dimensions (the leftmost columns and bottom lines in the table) increase, DFO-VU and RAGS reach a similar number of digits of accuracy. Of course, for both solvers the gain with respect to

Table 5: MQ Accuracy for all time budgets

	DFO-VU 1sec	NOMAD 1sec	RAGS 1sec	DFO-VU 10sec	NOMAD 10sec	RAGS 10sec	DFO-VU 60sec	NOMAD 60sec	RAGS 60sec
MQ-10 RA	4.00	1.00	5.00	6.00	2.00	6.00	6.00	3.00	6.00
MQ-20 RA	3.00	1.00	3.00	4.00	2.00	4.00	6.00	2.00	4.00
MQ-30 RA	2.00	1.00	3.00	4.00	1.00	4.00	5.00	2.00	4.00
MQ-40 RA	2.00	1.00	3.00	3.00	1.00	4.00	4.00	2.00	4.00
MQ-50 RA	2.00	1.00	2.00	3.00	2.00	3.00	4.00	2.00	4.00
Mean RA	2.60	1.00	3.20	4.00	1.60	4.20	5.00	2.20	4.40
# Stop. Test	0/614	0/614	13/614	120/614	28/614	294/614	432/614	139/614	565/614

NOMAD is only obtained at the expense of more function evaluations; see Table 5.

Regarding the NK family, the graphs in Figure 4 seem to indicate that achieving high accuracy is easier than with the MQ functions. The improvement in NOMAD's accuracy is noticeable when passing from the 1-second graph to the 10-second graph, as red triangles are higher in the latter. As with the MQ family, when given more time, the stopping test was triggered for some solvers (see the final line in Table 6).



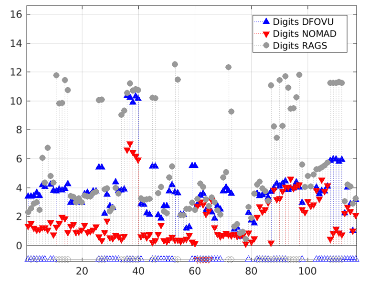


Figure 4: Accuracy, NK functions, 1 (left) and 10 (right) seconds. In the bottom line, a △, ▽ or ◯ indicates when the stopping test of the corresponding solver was triggered (respectively, DFO-VU,NOMAD,RAGS).

Similarly to the MQ family, RAGS' performance on the NK family is better than DFO-VU's, which in turn is better than NOMAD's. However, differently from the MQ functions, the number of digits computed by DFO-VU always remained below those obtained with RAGS, for all the runs and time budgets. For each solver, the achieved accuracy, averaging over each NK-subfunction, is given in Table 8, as well as the number of times the stopping test was triggered.

The improvement on NOMAD's accuracy is noticeable with the longer time limit, the average number of digits went up from 1.57 with 10 seconds

Table 6: NK Accuracy for all time budgets

		DFO-VU 1sec	NOMAD 1sec	RAGS 1sec	DFO-VU 10sec	NOMAD 10sec	RAGS 10sec	DFO-VU 60sec	NOMAD 60sec	RAGS 60sec
NH-1	RA	5.00	1.00	6.00	6.00	3.00	6.00	6.00	7.00	7.00
NH-2	RA	2.00	0.00	3.00	3.00	1.00	5.00	4.00	3.00	7.00
NH-3	RA	3.00	1.00	5.00	3.00	2.00	7.00	4.00	5.00	7.00
NH-4	RA	2.00	0.00	3.00	3.00	1.00	5.00	4.00	4.00	6.00
NH-5	RA	2.00	0.00	3.00	3.00	1.00	4.00	4.00	3.00	6.00
NH-6	RA	3.00	1.00	5.00	4.00	2.00	6.00	4.00	3.00	7.00
NH-7	RA	2.00	1.00	6.00	4.00	1.00	6.00	4.00	3.00	7.00
Mean	RA	2.71	0.57	4.43	3.71	1.57	5.57	4.29	4.00	6.71
# Stop.Test		7/117	0/117	17/117	68/117	5/117	49/117	82/117	23/117	74/117

to 4.00 with 60 seconds.

5.4.2 Tables with CPU time and calls to the black box

Regarding CPU times, it is important to notice that the total running time is not equal to the time budget: the limit given by the latter is checked inside each solver, in some place suitable for the considered method. To give a complete picture of the solvers' performances, in Tables 7 and 8 we report the total number of `bb`-calls and CPU time, for both families.

Table 7: MQ `bb`-calls and total running time (sec) for all time budgets

		DFO-VU 1sec	NOMAD 1sec	RAGS 1sec	DFO-VU 10sec	NOMAD 10sec	RAGS 10sec
MQ-10	$\bar{s}$	3382	227	3983	22160	5704	10586
	$\bar{T}$	1	0	1	15	9	2
MQ-20	$\bar{s}$	4103	75	3804	34101	1345	25641
	$\bar{T}$	2	1	2	18	10	10
MQ-30	$\bar{s}$	3838	60	3293	34576	687	32359
	$\bar{T}$	2	1	2	23	10	20
MQ-40	$\bar{s}$	3261	49	2762	31829	613	27783
	$\bar{T}$	2	1	2	22	10	23
MQ-50	$\bar{s}$	2624	139	2211	25948	1835	22398
	$\bar{T}$	2	0	2	20	10	21
Mean	$\bar{s}$	3442	110	3211	29723	2037	23753
	$\bar{T}$	2.00	1.00	2.00	20.00	10.00	15.00

On average, DFO-VU and RAGS use between 15 and 20 times the number of `bb`-calls employed by NOMAD. DFO-VU is implemented in a Newtonian version that, to estimate the U-Hessian at every x_k providing sufficient descent, uses the `bb`-calls to approximate second-order behaviour for *each* subfunction that is active at x_k. A quasi-Newton variant should reduce significantly those figures, likely without negatively impacting the accuracy.

Table 8: NK `bb`-calls and total running time (sec) for all time budgets

		DFO-VU 1sec	NOMAD 1sec	RAGS 1sec	DFO-VU 10sec	NOMAD 10sec	RAGS 10sec
NH-1	$\bar{s}$	5189	150	3052	15450	1523	23720
	$\bar{T}$	2	1	1	6	10	9
NH-2	$\bar{s}$	3928	113	3152	24172	863	22889
	$\bar{T}$	2	1	2	12	10	13
NH-3	$\bar{s}$	4503	105	3517	23433	1043	21174
	$\bar{T}$	2	1	2	11	10	11
NH-4	$\bar{s}$	3737	89	2733	26362	950	17666
	$\bar{T}$	2	1	2	13	10	13
NH-5	$\bar{s}$	3588	127	2391	20216	1348	14429
	$\bar{T}$	2	1	2	12	10	14
NH-6	$\bar{s}$	5375	160	3956	14804	2034	23028
	$\bar{T}$	1	1	1	4	8	8
NH-7	$\bar{s}$	5683	123	4185	27202	1435	28266
	$\bar{T}$	2	0	2	8	10	11
Mean	$\bar{s}$	4572	124	3284	21663	1314	21596
	$\bar{T}$	2.00	1.00	2.00	9.00	10.00	11.00

5.4.3 Comparing DFO-VUand RAGS

In order to see if in some situations the behaviour of DFO-VU compares to the best solver, RAGS, we run an additional bench test with 400 MQ functions in the higher dimensions (ranging between 55 and 100), referred as "HardMQ" below.

The accuracy achieved by each solver is plotted in the graphs in Figure 5, for the two considered time budgets. The fact that the HardMQ instances are harder than the MQ functions in Section 5.4 is clear from these graphs. The stopping test was never triggered for either solver and the digits obtained are all below 6 (compare with the graphs in Figure 3).

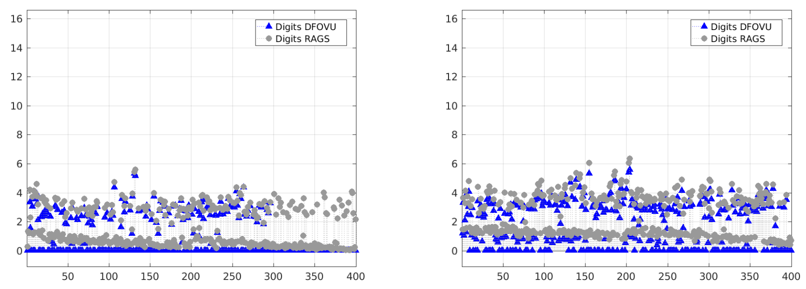


Figure 5: Accuracy, HardMQ functions, 1 (left) and 10 (right) seconds.

Table 9 reports the average accuracy, while Table 10 contains the number of bb-calls and CPU time, discriminating by problem dimension.

It is interesting to compare the number of bb-calls reported for DFO-VU and RAGS in Table 7 with those in Table 10. Roughly speaking, the respective factors are about 4 to 1: the 1-second mean values drop from more than 3000 to about 700; for the 10-second set, instead of more than 20000, the solvers could only make about 4500 evaluations. A similar comparison between Tables 5 and 9 shows that the accuracy diminished by less than a factor of 2 (in fact, for DFO-VU with one second, the accuracy is the same with both families).

Table 9: HardMQ Accuracy for all time budgets

		DFO-VU 1sec	RAGS 1sec	DFO-VU 0sec	RAGS 0sec
MQ-55	RA	3.00	2.00	2.00	3.00
MQ-60	RA	2.00	2.00	2.00	2.00
MQ-65	RA	3.00	2.00	2.00	2.00
MQ-70	RA	2.00	2.00	2.00	2.00
MQ-75	RA	3.00	1.00	3.00	3.00
MQ-80	RA	3.00	2.00	2.00	2.00
MQ-85	RA	3.00	1.00	3.00	2.00
MQ-90	RA	2.00	1.00	3.00	3.00
MQ-95	RA	3.00	1.00	3.00	3.00
MQ-100	RA	3.00	2.00	3.00	2.00
Mean	RA	2.70	1.60	2.50	2.40
# Stop.	Test	0/400	0/400	0/400	0/400

For this harder family of functions, DFO-VU slightly outperforms RAGS in terms of accuracy (particularly for the 1-second time budget), having consumed a comparable number of function evaluations and CPU time.

The final profiles in Figure 6 confirm that for the harder instances, both solvers DFO-VU and RAGS have practically the same performance. Figure 6 contains the data that used the 10-second time out only; the 1-second data profiles were essentially indistinguishable due to the low scale of ratios. Notice that the abscissa scale is much reduced this time, compared to the scale of order 10,000 in the MQ and NK data profiles, with the function call to problem dimension ratio between 30 and 120.

Figure 6: Data profiles for HardMQ functions, accuracy 2 (left) and 3 (right) digits.

Table 10: HardMQ `bb`-calls and total running time (sec) for all time budgets

		DFO-VU 1sec	RAGS 1sec	DFO-VU 0sec	RAGS 0sec
MQ-55	$\bar{S}$	934	1222	5361	6518
	$\bar{T}$	1	2	13	14
MQ-60	$\bar{S}$	961	1062	6095	6295
	$\bar{T}$	2	2	13	14
MQ-65	$\bar{S}$	898	931	4983	5982
	$\bar{T}$	2	2	16	18
MQ-70	$\bar{S}$	851	811	5452	5296
	$\bar{T}$	2	2	16	18
MQ-75	$\bar{S}$	771	668	4948	4780
	$\bar{T}$	2	2	16	17
MQ-80	$\bar{S}$	703	557	4539	4230
	$\bar{T}$	2	2	16	16
MQ-85	$\bar{S}$	619	505	4144	3810
	$\bar{T}$	2	2	16	16
MQ-90	$\bar{S}$	532	410	3720	3318
	$\bar{T}$	2	2	16	15
MQ-95	$\bar{S}$	455	361	3263	2851
	$\bar{T}$	2	2	18	18
MQ-100	$\bar{S}$	383	270	2962	2422
	$\bar{T}$	2	2	18	18
Mean	$\bar{S}$	711	680	4547	4550
	$\bar{T}$	2.00	2.00	16.00	16.00

5.5 Behaviour of DFO-VU on nonconvex problems

Even though our convergence analysis was developed for convex problems only, we also ran DFO-VU on a battery of nonconvex functions, to check numerically how the method behaves in this case. The test functions are of the form (12), again randomly generated with a known functional value at a critical point, satisfying $0 \in \partial f(\bar{x})$ for Clarke's subdifferential. Nonconvexity is induced by taking, among all the matrices H_j defining the quadratic subfunctions in (12), at least one that is negative definite. However, since we are dealing with unconstrained problems, to have a local solution that is finite-valued, one of the m random matrices is forced to be positive definite.

We note that only DFO-VU was tested on these problems. Algorithms InexBun, ExBun, and CompBun are specifically designed for convex functions and no longer work when subgradients from nonconvex functions are used. We did not attempt to use RAGS on this set and were unable to make Nomad perform in a reasonable manner on these problems.

This test set consists of 1000 test functions, 200 problems with 5 starting points each, with $n \in \{10, 20, 30, 40, 50\}$. The results are encouraging. In fact, without making any changes in the implementation, the DFO-VU stopping test was triggered in 958 cases. For these successful runs, Table 11 reports the number of oracle calls, CPU time and digits of accuracy, again shown in average separately for each one of the five groups of test functions.

Table 11: Average output for DFO-VU on 958 successful nonconvex runs

Problem set	grey-box calls	time	RA
$n = 10$ (189 successful runs)	2067	0.87	1.08
$n = 20$ (200 successful runs)	3135	0.96	0.75
$n = 30$ (191 successful runs)	1440	0.80	0.44
$n = 40$ (190 successful runs)	2659	1.67	0.35
$n = 50$ (188 successful runs)	2690	2.00	0.16

Clearly, the accuracy levels are not as good as for the convex case. However, the time and grey-box calls were improved over the convex case. This suggests that the stopping condition is somehow easier to trigger in the nonconvex setting. In general, the output is consistent, with worse indicators for problems in higher dimensions. The group with $n = 30$ is an exception, since it required fewer function evaluations (1440) than the easier instances, with $n = 10$ or 20. The fact that the solver ends up with false positive output is not unexpected, considering the stopping test is designed on the basis of the convergence analysis, which holds for convex problems only.

Among the 42 runs in which DFO-VU failed, the parameter ε_k became too small for the problems in higher dimensions ($n \geq 30$). This is not surprising, as the method had already shown high sensitivity to that parameter in the convex instances. In lower dimensions ($n \leq 20$), sometimes DFO-VU failed in building a suitable matrix $\hat{M}$. In view of Remark 3.4, this suggests the need to fine-tune the parameter ε_k, to adapt its iterative definition to the nonconvex setting.

Overall, the results indicate that it might be worthwhile to extend both the theory and the implementation of DFO-VU to tackle nonconvex functions.

6 Conclusion

We have presented a complete and fully-functional DFO $\mathcal{VU}$-algorithm for convex finite-max objective functions on $\mathbb{R}^n$ under reasonable assumptions.

This extends the original algorithm of [50] into the derivative-free setting, where exact function values are available but approximations of subgradients are sufficient for convergence. Numerical testing suggests that, at the expense of increased CPU time and number of function calls, the DFO $\mathcal{VU}$-algorithm provides an improvement on final function value accuracy when compared to other inexact methods, and even compared to the ExBun method that uses exact first-order information. Convergence rate analysis was not performed in this paper; we leave that for a future project.

When compared to other DFO methods, on the tested instances, DFO-VU presents a good performance with respect to Nomad in terms of accuracy, and behaves slightly less well than RAGS. The gain in accuracy is associated with a large increase in the number of functional evaluations. In the numerical implementation of DFO-VU there is much room for improvement of its performance. We did hand-tune the parameters to get good performance, but other tweaks in the code that were not done would help as well. The replacement of the U-Hessian by a quasi-Newton version is a possibility. Also, a hard reset happens at every iteration, which means that nearby function values already calculated are not reused in the construction of the next model function. Retaining a cache of function calls and referencing it before making new evaluations would reduce the total number of grey-box calls. In addition, in the construction of the simplex gradient we used the coordinate directions. A method such as Householder transformation [57] could be used to rotate the coordinates so that the first canonical vector points in the previous descent direction. These adjustments can reduce the number of function calls by a factor between n and n^2, so it is encouraging to know that future work on this project should result in quite a significant enhancement of the algorithm.

As mentioned in the introduction, one limitation of this algorithm is that convergence applies the assumption that the objective function is convex. It is unclear how neccesary this assumption is for finite-max grey-box functions, however it would obviously be beneficial if the assumption could be relaxed. One starting point might be the research on proximal bundle methods for nonconvex functions [3, 28] and $\mathcal{VU}$-structures for nonconvex functions [49]. However, as was pointed out in [28], the difficulty in working with nonconvex functions lies in the fact that the cutting-plane models are no longer guaranteed to lie below the function. Since the DF-simplex gradient is an approximation of some true subgradient, this adds one layer of inexactness to the linearizations defining the cutting-place model. There are errors from the approximate gradients and errors from nonconvexity. Without a clear manner to distinguish one error from another, the whole mechanism breaks down. Of course, as demonstrated in the numerical tests, the algorithm is still well-defined for nonconvex functions.

Another interesting approach may come from the line of recent work by Salomão, Santos, and Simões [60, 61, 62]. In [62], the authors present a gradient sampling method that has improved convergence speed, thanks to $\mathcal{VU}$-decomposition. They stress the point that gradient sampling is convenient when the objective function is nonconvex, avoiding the complications that arise when bundle methods such as the $\mathcal{VU}$-algorithm are applied to nonconvex functions. The algorithm therein retains some components of the $\mathcal{VU}$-algorithm in order to speed up convergence; it uses quasi-Newton techniques in the $\mathcal{U}$-space and cutting-plane techniques in the $\mathcal{V}$-space. It is our belief that the derivative-free methods presented in this paper will be applicable in the algorithm of [62] and other similar algorithms. Exact first-order data are employed currently, but since we have seen that such gradients (at least for finite-max problems, which the authors of [62] cite as motivation for their method) can be approximated to any desired degree of accuracy, there is good reason to conjecture that the same approach would work there. It is a natural direction for the continuation of this line of research.

References

[1] M. Abramson, C. Audet, J. Dennis, Jr., and S. Le Digabel. OrthoMADS: a deterministic MADS instance with orthogonal directions. *SIAM J. Optim.*, 20(2):948–966, 2009.

[2] W. van Ackooij and W. de Oliveira. Level bundle methods for constrained convex optimization with various oracles. *Comput. Optim. Appl.*, 57(3):555–597, 2014.

[3] P. Apkarian, D. Noll, and L. Ravanbod. Nonsmooth bundle trust-region algorithm with applications to robust stability. *Set-Valued Var. Anal.*, 24(1):115–148, 2016.

[4] E. Asplund and R. Rockafellar. Gradients of convex functions. *Trans. Amer. Math. Soc.*, 139:443–467, 1969.

[5] C. Audet. *A Survey on Direct Search Methods for Blackbox Optimization and their Applications*, pages 31–56. Springer, New York, 2014.

[6] C. Audet, V. Béchard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *J. Global Optim.*, 41(2):299–318, 2008.

[7] C. Audet, V. Béchard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization*, 41(2):299–318, 2008.

[8] C. Audet and J. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim.*, 17(1):188–217, 2006.

[9] C. Audet and W. Hare. *Derivative-free and Blackbox Optimization.* Springer International Publishing AG, Switzerland, 2017. (Final edits submitted Aug., 2017. Published online Dec. 2017. Hardcopy available Jan. 2018.).

[10] A. Bagirov, B. Karasözen, and M. Sezer. Discrete gradient method: derivative-free method for nonsmooth optimization. *J. Optim. Theory Appl.*, 137(2):317–334, 2008.

[11] K. Bigdeli, W. Hare, J. Nutini, and S. Tesfamariam. Optimizing damper connectors for adjacent buildings. *Optimization and Engineering*, 17(1):47–75, 2016.

[12] J. Bonnans, J. Gilbert, C. Lemaréchal, and C. Sagastizábal. *Numerical Optimization. Theoretical and Practical Aspects.* Universitext. Springer-Verlag, Berlin, 2006. Second edition, pp. xiv+490.

[13] S. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, Cambridge, 2004.

[14] P. Combettes and J. Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, volume 49 of *Springer Optim. Appl.*, pages 185–212. Springer, New York, 2011.

[15] A. Conn, K. Scheinberg, and P. Toint. On the convergence of derivative-free methods for unconstrained optimization. In *Approximation theory*

and optimization (Cambridge, 1996), pages 83–108. Cambridge Univ. Press, Cambridge, 1997.

[16] A. Conn, K. Scheinberg, and L. Vicente. *Introduction to derivative-free optimization*, volume 8 of *MPS/SIAM Series on Optimization*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA; Mathematical Programming Society (MPS), Philadelphia, PA, 2009.

[17] R. Correa and C. Lemaréchal. Convergence of some algorithms for convex minimization. *Math. Program.*, 62(2, Ser. B):261–275, 1993.

[18] A. Custódio and L. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM J. Optim.*, 18(2):537–555, 2007.

[19] A. Frangioni and E. Gorgone. Bundle methods for sum-functions with "easy" components: applications to multicommodity network design. *Math. Program.*, 145(1-2, Ser. A):133–161, 2014.

[20] A. Fuduli, M. Gaudioso, and G. Giallombardo. A DC piecewise affine model and a bundling technique in nonconvex nonsmooth minimization. *Optim. Methods Softw.*, 19(1):89–102, 2004.

[21] W. Hare. Numerical analysis of VU-decomposition, U-gradient, and U-hessian approximations. *SIAM J. Optim.*, 24(4):1890–1913, 2014.

[22] W. Hare and Y. Lucet. Derivative-free optimization via proximal point methods. *J. Optim. Theory Appl.*, 160(1):204–220, 2014.

[23] W. Hare and M. Macklem. Derivative-free optimization methods for finite minimax problems. *Optim. Methods Softw.*, 28(2):300–312, 2013.

[24] W. Hare and J. Nutini. A derivative-free approximate gradient sampling algorithm for finite minimax problems. *Comput. Optim. Appl.*, 56(1):1–38, 2013.

[25] W. Hare, J. Nutini, and S. Tesfamariam. A survey of non-gradient optimization methods in structural engineering. *Adv. Eng. Soft.*, 59:19–28, 2013.

[26] W. Hare and C. Planiden. Computing proximal points of convex functions with inexact subgradients. *(to appear) Set-Valued Var. Anal.*, pages 1–24, .

[27] W. Hare, C. Sagastizábal, and M. Solodov. A proximal bundle method for nonsmooth nonconvex functions with inexact information. *Comput. Optim. Appl.*, 63(1):1–28, 2016.

[28] W. Hare, C. Sagastizábal, and M. Solodov. A proximal bundle method for nonsmooth nonconvex functions with inexact information. *Comput. Optim. Appl.*, 63(1):1–28, 2016.

[29] C. Helmberg, M. Overton, and F. Rendl. The spectral bundle method with second-order information. *Optim. Methods Softw.*, 29(4):855–876, 2014.

[30] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM J. Optim.*, 10(3):673–696, 2000.

[31] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex analysis and minimization algorithms. II*, volume 306 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1993. Advanced theory and bundle methods.

[32] K. Joki, A. Bagirov, N. Karmitsa, and M. Mäkelä. A proximal bundle method for nonsmooth DC optimization utilizing nonconvex cutting planes. *J. Global Optim.*, 68(3):501–535, 2017.

[33] N. Karmitsa. Test problems for large-scale nonsmooth minimization. *Reports of the Department of Mathematical Information Technology. Series B, Scientific computing*, 4, 2007.

[34] N. Karmitsa, A. Bagirov, and S. Taheri. New diagonal bundle method for clustering problems in large data sets. *European J. Oper. Res.*, 263(2):367–379, 2017.

[35] C. Kelley. *Iterative methods for optimization*, volume 18 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

[36] C. Kelley. *Implicit filtering*, volume 23 of *Software, Environments, and Tools*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2011.

[37] K. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Math. Program.*, 46(1, (Ser. A)):105–122, 1990.

[38] K. Kiwiel. A proximal bundle method with approximate subgradient linearizations. *SIAM J. Optim.*, 16(4):1007–1023, 2006.

[39] K. Kiwiel. A nonderivative version of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM J. Optim.*, 20(4):1983–1994, 2010.

[40] J. Larson, M. Menickelly, and S. Wild. Manifold sampling for ℓ_1 nonconvex optimization. *SIAM J. Optim.*, 26(4):2540–2563, 2016.

[41] S. Le Digabel. Nomad: Nonlinear optimization with the mads algorithm. *Rapport technique G-2009-39, Les cahiers du GERAD*, 2009.

[42] C. Lemaréchal, F. Oustry, and C. Sagastizábal. The U-Lagrangian of a convex function. *Trans. Amer. Math. Soc.*, 352(2):711–729, 2000.

[43] A. Lewis and S. Wright. A proximal method for composite minimization. *Math. Program.*, 158(1-2, Ser. A):501–546, 2016.

[44] R. Mifflin and C. Sagastizábal. VU-decomposition derivatives for convex max-functions. In *Ill-posed variational problems and regularization techniques (Trier, 1998)*, volume 477 of *Lecture Notes in Econom. and Math. Systems*, pages 167–186. Springer, Berlin, 1999.

[45] R. Mifflin and C. Sagastizábal. Functions with primal-dual gradient structure and U-Hessians. In *Nonlinear optimization and related topics (Erice, 1998)*, volume 36 of *Appl. Optim.*, pages 219–233. Kluwer Acad. Publ., Dordrecht, 2000.

[46] R. Mifflin and C. Sagastizábal. On VU-theory for functions with primal-dual gradient structure. *SIAM J. Optim.*, 11(2):547–571, 2000.

[47] R. Mifflin and C. Sagastizábal. Proximal points are on the fast track. *J. Convex Anal.*, 9(2):563–579, 2002.

[48] R. Mifflin and C. Sagastizábal. Primal-dual gradient structured functions: second-order results; links to epi-derivatives and partly smooth functions. *SIAM J. Optim.*, 13(4):1174–1194, 2003.

[49] R. Mifflin and C. Sagastizábal. VU-smoothness and proximal point results for some nonconvex functions. *Optim. Methods Softw.*, 19(5):463–478, 2004.

[50] R. Mifflin and C. Sagastizábal. A VU-algorithm for convex minimization. *Math. Program.*, 104(2-3, Ser. B):583–608, 2005.

[51] L. Nel. *Continuity Theory.* Springer, first edition, 2016.

[52] D. Noll, O. Prot, and A. Rondepierre. A proximity control algorithm to minimize nonsmooth and nonconvex functions. *Pac. J. Optim.*, 4(3):571–604, 2008.

[53] W. de Oliveira. Proximal bundle methods for nonsmooth DC programming. *preprint*, 2017. `http://www.oliveira.mat.br/publications`.

[54] W. de Oliveira, C. Sagastizábal, and C. Lemaréchal. Convex proximal bundle methods in depth: a unified analysis for inexact oracles. *Math. Program.*, 148(1-2, Ser. B):241–277, 2014.

[55] W. de Oliveira and M. Solodov. A doubly stabilized bundle method for nonsmooth convex optimization. *Math. Program.*, 156(1-2, Ser. A):125–159, 2016.

[56] M. Powell. Developments of NEWUOA for minimization without derivatives. *IMA J. Numer. Anal.*, 28(4):649–664, 2008.

[57] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge, third edition, 2007.

[58] R. Rockafellar and J.-B. Wets. *Variational analysis.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]. Springer-Verlag, Berlin, 1998.

[59] C. Sagastizábal. Composite proximal bundle method. *Math. Program.*, 140(1):189–233, 2013.

[60] E. Salomão, S. Santos, and L. Simões. On the differentiability check in gradient sampling methods. *Optim. Methods Softw.*, 31(5):983–1007, 2016.

[61] E. Salomão, S. Santos, and L. Simões. On the local convergence analysis of the gradient sampling method. *preprint, Optimization Online*, 2016.

[62] E. Salomão, S. Santos, and L. Simões. A second-order information-based gradient and function sampling method for nonconvex nonsmooth optimization. *preprint, Optimization Online*, 2017.