2016

Computing proximal points of convex functions with inexact subgradients

Chayne Planiden
University of British Columbia, chayne@uow.edu.au

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

Publication Details

Planiden, C. & Hare, W. (2018). Computing proximal points of convex functions with inexact subgradients. Set-Valued and Variational Analysis: an international journal devoted to the theory of multifunctions, 26 (3), 469-492.

Computing proximal points of convex functions with inexact subgradients

Abstract
Locating proximal points is a component of numerous minimization algorithms. This work focuses on developing a method to find the proximal point of a convex function at a point, given an inexact oracle. Our method assumes that exact function values are at hand, but exact subgradients are either not available or not useful. We use approximate subgradients to build a model of the objective function, and prove that the method converges to the true prox-point within acceptable tolerance. The subgradient g_k used at each step k is such that the distance from g_k to the true subdifferential of the objective function at the current iteration point is bounded by some fixed epsilon>0. The algorithm includes a novel tilt-correct step applied to the approximate subgradient.

Disciplines
Engineering | Science and Technology Studies

Publication Details
Planiden, C. & Hare, W. (2018). Computing proximal points of convex functions with inexact subgradients. Set-Valued and Variational Analysis: an international journal devoted to the theory of multifunctions, 26 (3), 469-492.

Computing proximal points of convex functions with inexact subgradients

W. Hare* C. Planiden†

July 6, 2016

Abstract

Locating proximal points is a component of numerous minimization algorithms. This work focuses on developing an inexact bundle method to find the proximal point of a convex function at a given point. Our method assumes that exact function values are at hand, but exact subgradients are either not available or not useful. We use approximate subgradients to build a model of the objective function, and prove that the method converges to the true prox-point within acceptable tolerance. The algorithm includes a novel tilt-correction step applied to the approximate subgradient.

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

Keywords: Convex optimization, Proximal point, inexact subgradient, bundle method.

1 Introduction

Given a convex function $f : \mathbb{R}^n \mapsto \mathbb{R}$, the proximal point of f at z with prox-parameter $r > 0$ is defined

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

First arising in the works of Moreau [34, 35], the proximal point operator has emerged as a subproblem in a diverse collection of algorithms. The basic proximal point algorithm sets

$$x_{k+1} = \mathrm{Prox}_f^r(x_k)$$

and was shown to converge to a minimizer of f by Martinet [31]. It has since been shown to provide favorable convergence properties in a number of situations (see [1, 11, 17, 36] et al.).

*Mathematics, University of British Columbia Okanagan, Kelowna, B.C. V1V 1V7, Canada. Research by this author was partially supported by an NSERC Discovery Grant. shawn.wang@ubc.ca.

†Mathematics, University of British Columbia Okanagan, Kelowna, B.C. V1V 1V7, Canada. Research by this author was supported by UBC UGF and by NSERC of Canada. chayne.planiden@alumni.ubc.ca.

The basic proximal point algorithm has inspired a number of practical approaches, each of which evaluates the proximal point operator as a subroutine.

For example, proximal gradient methods apply the proximal point operator to a linearization of a function f [43, 44]. Proximal bundle methods advance this idea by using a bundle of information to create a convex piecewise-linear approximation of f, then apply the proximal point operator on the approximation function to determine the next iterate [4, 18, 25, 28]. Proximal splitting methods [6], such as the Douglas-Rachford algorithm [10], focus on the minimization of the sum of two functions and proceed by applying the proximal point operator on each function in alternation. Another example is the novel proximal method for composite functions $F(x) = g(f(x))$ [29, 38]. Among the most complex methods, the $\mathcal{VU}$-algorithm alternates between a proximal-point step and a '$\mathcal{U}$-Newton' step to achieve super-linear convergence in the minimization of nonsmooth convex funcftions [33].

Practical implementations of all of the above methods exist and are generally accepted as highly effective. In most implementations of these methods, the basic assumption is that the algorithm has access to an oracle that returns the function value and a subgradient vector at a given point. This provides a great deal of flexibility, and makes the algorithms suitable for nonsmooth optimization problems.

However, in many applications the user has access to an oracle that returns only function values (see for example [7, 16] and the many references therein). If the objective is smooth, then practitioners can apply gradient approximation techniques [24], and rely on classical smooth optimization algorithms. However, if the objective is nonsmooth, then practitioners generally rely on direct search methods (see [7, Ch. 7]). While direct search methods are robust and proven to converge, they do not exploit any potential structure of the problem, so there is room for improvement and new developments of algorithms applied to nonsmooth functions using oracles that return only function values.

To that end, other papers in this vein present results that have demonstrated the ability to approximate a subgradient using only function evaluations [2, 13, 15, 27]. This provides opportunity for the development of proximal-based methods for nonsmooth convex functions. Such methods will require the use of a proximal subroutine that relies on inexact subgradient information. In this paper, we develop such a method, prove its convergence, provide stopping criterion analysis, and include numerical testing on the method. This method can now be used as a foundation in proximal-based methods for nonsmooth convex functions where the oracle returns an exact function value and an inexact subgradient vector. We present the method in terms of an arbitrary approximate subgradient, noting that any of the methods from [2, 13, 15, 27], or any future method of similar style, can provide the approximate subgradient.

Remark 1.1. *It should be noted that several methods that use proximal-style subroutines and inexact subgradient vectors have already been developed [9, 19, 26, 39, 40, 42, 20, 41]. However, in each case the subroutine is embedded within the developed method, and only analyzed in light of the developed method. In this paper we develop a subroutine that uses exact function values and inexact subgradient vectors to determine the proximal point for a nonsmooth convex function. As a stand-alone method, the algorithm developed in this paper can be used as a subroutine in any proximal-style algorithm. (Some particular goals in this area are outlined in Section 6.) Some more*

technical differences between the algorithm in this work and the inexact gradient proximal-style subroutines in other works appear in Subsection 3.3.

The algorithm in this paper is for finite-valued, convex objective functions, and is based on standard bundle methods (see [4, 17]). We assume that for a given point x, the exact function value $f(x)$ and an approximate subgradient g^ε such that $\operatorname{dist}(g^\varepsilon, \partial f(x)) < \varepsilon$ are provided, where $\partial f(x)$ is the convex subdifferential as defined in [37, §8C]. Using this information, a piecewise-linear approximation of f is built, and a quadratic problem is used to determine the proximal point of the approximation function – an *approximal point*. Unlike methods that use exact subgradients, the algorithm includes a subgradient correction term that is required to ensure convergence. The algorithm is outlined in detail in Section 3.

Given a stopping tolerance $\mathrm{s}_{\mathtt{tol}}$, in Section 4 we prove that if a stopping condition is met, then a solution has been found that is within $\varepsilon + \mathrm{s}_{\mathtt{tol}}/r$ of the true proximal point, where ε is the subgradient error bound and r is the prox-parameter. We also prove that any accumulation point of the algorithm lies within $\varepsilon + \mathrm{s}_{\mathtt{tol}}/r$ of the true proximal point.

In Section 5, we discuss practical implementation of the developed algorithm. The algorithm is implemented in MATLAB version 8.4.0.150421 (R2014b), and several variants are numerically tested on a collection of randomly generated functions. Tests show that the algorithm is effective, even when ε is quite large.

Finally, in Section 6 we provide some concluding remarks, specifically pointing out some areas that should be examined in future research.

2 Preliminaries

2.1 Notation

Throughout, we assume that the objective function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is finite-valued, convex, proper, and lower semicontinuous (*lsc*).

The Euclidean vector norm is denoted $\|\cdot\|$. With $\delta > 0$, we use $B_\delta(x)$ to denote the open ball centred at x with radius δ. We denote the gradient of a function f by ∇f. The (convex) subdifferential of f is denoted ∂f, and a subgradient of f at x is denoted $g \in \partial f(x)$, as discussed in [37, §8.C]. The distance from a point $x \in \mathbb{R}^n$ to a set C is denoted $d_C(x)$, and the projection of x onto C is denoted $P_C x$.

Given $K > 0$, we say that the function f is *locally K-Lipschitz continuous* about $z \in \operatorname{dom} f$ with radius $\sigma > 0$, if

$$\|f(y) - f(x)\| \leq K\|y - x\| \text{ for all } x, y \in B_\sigma(z).$$

We say that f is *globally K-Lipschitz continuous* if σ can be taken to be ∞.

2.2 The Proximal Point

Given a proper, lsc, convex function f, a point $z \in \operatorname{dom} f$ (the *prox-centre*), and a prox-parameter $r > 0$, we consider the problem of finding the proximal point of f at z:

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

As f is convex, this point exists and is unique [37, Thm 2.26]. If f is locally K-Lipschitz continuous at z with radius $\sigma > 0$, then [17, Lemma 2] implies

$$\|\operatorname{Prox}_f^r(z) - z\| < \frac{2K}{r} \text{ whenever } \frac{2K}{r} < \sigma. \tag{2.1}$$

In particular, if f is globally K-Lipschitz continuous, then $\|\operatorname{Prox}_f^r(z) - z\| < \frac{2K}{r}$.

The proximal point can be numerically computed via an iterative method. Given an exact oracle, one method for numerically computing a proximal point of f is as follows. Let $z \in \operatorname{dom} f$ be the prox-centre. We create an information bundle, $D_k = \{(x_i, f_i, g_i) \ : \ i \in \mathcal{B}_k\}$, where x_i is a point at which the oracle has been called, $f_i = f(x_i)$ is the function value returned by the oracle, $g_i = g(x_i) \in \partial f(x_i)$ is the subgradient vector returned by the oracle, and $\mathcal{B}_k$ is the bundle index set. At each iteration k, the piecewise-linear function ϕ_k is defined:

$$\phi_k(x) := \max_{i \in \mathcal{B}_k} \left\{ f_i + g_i^\top (x - x_i) \right\}. \tag{2.2}$$

Then the proximal point of ϕ_k (the approximal point) is calculated, $x_{k+1} = \operatorname{Prox}_{\phi_k}^r(z)$, and the oracle is called at x_{k+1} to obtain f_{k+1} and g_{k+1}. If $(f_{k+1} - \phi_k(x_{k+1}))/r < \mathtt{s_{tol}}^2$, where $\mathtt{s_{tol}}$ is the stopping tolerance, then the algorithm stops and returns x_{k+1}. Otherwise, the element $(x_{k+1}, f_{k+1}, g_{k+1})$ is inserted into the bundle and the process repeats. Further information of this approach can be found in [21, Chapter XI], and in [25, 26].

Computing the approximal point is a convex quadratic program, and can therefore be solved efficiently as long as the dimension and the bundle size remain reasonable [5]. In order to keep the bundle size reasonable, various techniques such as bundle cleaning [23] and aggregate gradient cutting planes [32] have been advanced. As a result, we have a computationally tractable algorithm that, under mild assumptions, can be proved to converge to the true proximal point.

In this work, we are interested in how this method must be adapted if, instead of returning $g_k \in \partial f(x_k)$, the oracle returns

$$\tilde{g}_k^\varepsilon \in \partial f(x_k) + B_\varepsilon(0). \tag{2.3}$$

We address this issue in the next section.

3 Replacing Exactness with Approximation

3.1 The approximate model function and approximate subgradient

We denote the maximum subgradient error by ε, and use $\tilde{g}_i^\varepsilon$ to represent the inexact subgradient returned by the oracle at point x_i. We will use this information to define a new bundle element to

update the model function, but first we want to ensure that our model function will not lie above the objective function at the prox-centre. This is a necessary component of our convergence proof. So if the linear function defined by the new bundle element lies above f at z, we make a correction to $\tilde{g}_k^\varepsilon$. We set $g_k^\varepsilon = \tilde{g}_k^\varepsilon - c_k$, where the correction term c_k is nonzero if correction is needed, zero otherwise. Then, denoting the bundle index set by $\mathcal{B}_k$, the piecewise-linear model function ϕ_k^ε is defined:

$$\phi_k^\varepsilon(x) := \max_{i \in \mathcal{B}_k} \left\{ f_i + {g_i^\varepsilon}^\top (x - x_i) \right\}. \tag{3.1}$$

We use $\mathcal{D}_k$ to denote the set of bundle elements. At initialization ($k = 0$), we have $\mathcal{B}_0 = \{0\}$ and $\mathcal{D}_0 = \{(z, f_0, g_0^\varepsilon)\}$. For each $k \geq 1$, we will have at least three bundle elements: $\mathcal{B}_k \supseteq \{-1, 0, k\}$ and $\mathcal{D}_k \supseteq \{(x_k, \phi_{k-1}^\varepsilon(x_k), r(z - x_k)), (z, f_0, g_0^\varepsilon), (x_k, f_k, g_k^\varepsilon)\}$. In bundle methods, the bundle component $r(z - x_k)$ is known as the *aggregate subgradient* [9, 26, 39], and is an element of $\partial\phi_{k-1}^\varepsilon(x_k)$. In this work, we adopt the convention of using the index -1 as the label for the aggregate bundle element $(x_k, \phi_{k-1}^\varepsilon(x_k), r(z - x_k))$.We may have up to $k + 2$ elements: $\mathcal{B}_k = \{-1, 0, 1, 2, \ldots, k\}$, however, elements $-1, 0$ and k are sufficient to guarantee convergence.

Now let us consider the correction term c_k. Suppose that

$$E_k := f_k + {\tilde{g}_k^\varepsilon}^\top (z - x_k) - f(z) > 0, \tag{3.2}$$

thus necessitating a correction. We seek the minimal correction term, hence, we need to find

$$c_k \in \operatorname{argmin}\{\|c\| : c^\top (z - x_k) - E_k = 0\}.$$

This gives

$$c_k = \operatorname{Proj}_G(0), \text{ where } G = \left\{ c : \frac{c^\top (z - x_k)}{\|z - x_k\|} = \frac{E_k}{\|z - x_k\|} \right\}. \tag{3.3}$$

That is, c_k is the projection of 0 onto the hyperplane generated by the normal vector $z - x_k$ and shift constant E_k. This yields

$$c_k = \frac{E_k}{\|z - x_k\|} \frac{z - x_k}{\|z - x_k\|} = E_k \frac{z - x_k}{\|z - x_k\|^2}. \tag{3.4}$$

Now we define the approximate subgradient that we use in the algorithm:

$$g_k^\varepsilon := \begin{cases} \tilde{g}_k^\varepsilon - c_k, & \text{if } f_k + {\tilde{g}_k^\varepsilon}^\top (z - x_k) > f(z), \\ \tilde{g}_k^\varepsilon, & \text{if } f_k + {\tilde{g}_k^\varepsilon}^\top (z - x_k) \leq f(z). \end{cases}$$

Since $\tilde{g}_k^\varepsilon$ is the approximate subgradient returned by the oracle but g_k^ε is the one we want to use in construction of the model function, we must first prove that g_k^ε also respects (2.3).

Lemma 3.1. *Let f be convex. Then at any iteration k, $\operatorname{dist}(g_k^\varepsilon, \partial f(x_k)) < \varepsilon$.*

Proof. If $g_k^\varepsilon = \tilde{g}_k^\varepsilon$, then the result holds by Assumption 2.3. Suppose $f_k + \tilde{g}_k^{\varepsilon\top}(z - x_k) > f(z)$. Define

$$H := \left\{ g : f(z) = f(x_k) + g^\top (z - x_k) \right\},$$
$$J := \left\{ g : f(z) \geq f(x_k) + g^\top (z - x_k) \right\}.$$

By equation (3.3), we have that $g_k^\varepsilon = P_H(\tilde{g}_k^\varepsilon)$. Since $f(z) < f_k + \tilde{g}_k^{\varepsilon\top}(z - x_k)$, we also know $P_J(\tilde{g}_k^\varepsilon) = P_H(\tilde{g}_k^\varepsilon) = g_k^\varepsilon$. By equation (2.3), there exists $\bar{g} \in \partial f(x_k)$ such that $\|\tilde{g}_k^\varepsilon - \bar{g}\| < \varepsilon$. Since f is convex, we have

$$f(z) \geq f(x_k) + \bar{g}^\top (z - x_k), \tag{3.5}$$

hence $\bar{g} \in J$ and $P_J(\bar{g}) = \bar{g}$. Using the fact that the projection is firmly nonexpansive [3, Proposition 4.8], we have

$$\|g_k^\varepsilon - \bar{g}\| = \|P_J(\tilde{g}_k^\varepsilon) - P_J(\bar{g})\| \leq \|\tilde{g}_k^\varepsilon - \bar{g}\| < \varepsilon,$$

which is the desired result. □

In the case of exact subgradients, the resulting linear functions form cutting planes of f, so that the model function is an under-estimator of the objective function. That is, for exact subgradient g_i,

$$f(x) \geq f_i + g_i^\top (x - x_i) \text{ for all } i \in \mathcal{B}_k\,. \tag{3.6}$$

In the approximate subgradient case we do not have this luxury, but all is not lost. Using inequality (3.6) and the fact that $\|g_i^\varepsilon - g_i\| < \varepsilon$, we have that for all $i \in \mathcal{B}_k$ and for all $x \in \mathbb{R}^n$,

$$\begin{aligned} f(x) &\geq f_i + (g_i - g_i^\varepsilon + g_i^\varepsilon)^\top (x - x_i) \\ &= f_i + g_i^{\varepsilon\top}(x - x_i) + (g_i - g_i^\varepsilon)^\top (x - x_i) \\ &\geq f_i + g_i^{\varepsilon\top}(x - x_i) - \|g_i - g_i^\varepsilon\|\|x - x_i\| \\ &\geq f_i + g_i^{\varepsilon\top}(x - x_i) - \varepsilon\|x - x_i\|. \end{aligned}$$

Hence,

$$f(x) + \varepsilon\|x - x_i\| \geq \phi_k^\varepsilon(x) \text{ for all } x \in \mathbb{R}^n, i \in \mathcal{B}_k, k. \tag{3.7}$$

3.2 The algorithm

Now we present the algorithm that uses approximate subgradients. In Section 5 we implement four variants of this algorithm, comparing four different ways of updating the bundle in Step 5.

Algorithm:

Step 0: *Initialization.* Choose a prox-centre $z \in \operatorname{dom} f$, stopping tolerance $\mathrm{s}_{\mathtt{tol}} \geq 0$, and prox-parameter $r > 0$. Set $k = 0$ and $x_0 = z$. Set $\mathcal{B}_0 = \{0\}$. Use the oracle to find $f_0, \tilde{g}_0^\varepsilon$.

Step 1: *Linearization.* Compute $E_k = f_k + \tilde{g}_k^{\varepsilon\top}(z - x_k) - f(z)$, and define

$$g_k^\varepsilon := \tilde{g}_k^\varepsilon + \max\{0, E_k\}\frac{z - x_k}{\|z - x_k\|^2}.$$

Step 2: *Model.* Define

$$\phi_k^{\varepsilon}(x) := \max_{i \in \mathcal{B}_k} \left\{ f_i + {g_i^{\varepsilon}}^{\top}(x - x_i) \right\}.$$

Step 3: *Proximal Point.* Calculate the next bundle point $x_{k+1} = \text{Prox}_{\phi_k^{\varepsilon}}^{r}(z)$, and use the oracle to find $f_{k+1}, \tilde{g}_{k+1}^{\varepsilon}$.

Step 4: *Stopping Test.* If $\frac{f_{k+1} - \phi_k^{\varepsilon}(x_{k+1})}{r} \leq \mathtt{s_{tol}}^2$, output the approximal point x_{k+1} and stop.

Step 5: *Update and Loop.* Create the aggregate bundle element $(x_k, \phi_{k-1}^{\varepsilon}, r(z - x_k))$. Create $\mathcal{B}_{k+1}$ such that $\{-1, 0, k\} \subseteq B_{k+1} \subseteq \{-1, 0, 1, 2, \cdots, k\}$. Increment k and go to Step 1.

3.3 Relation to other inexact gradient proximal-style subroutines

Now that the algorithm is presented, we can provide some insights on how it relates to previously developed inexact subgradient prox-point computations. First and foremost, the presented algorithm is a stand-alone method that is not presented as a subroutine of another algorithm. To our knowledge, all of the other methods of computing prox-points that use inexact subgradients are subroutines found within other algorithms.

In 1995, [26] presented a method of computing a prox-point using inexact subgradients as a subroutine in a minimization algorithm for a nonsmooth, convex objective function. However, the algorithm assumes that the inexactness in the subgradient takes the form of an ε-subgradient. An ε-subgradient v^{ε} of f at x_k is an approximate subgradient that satisfies

$$f(x) \geq f(x_k) + {v^{\varepsilon}}^{\top}(x - x_k) - \varepsilon \text{ for all } x.$$

Thus, the method in [26] relies on the approximate subgradient forming an ε-cutting plane in each iteration.

While ε-subgradients do appear in some real-world applications [12, 22], in other situations the ability to determine ε-subgradients is an unreasonable expectation. For example, if the objective function is a black-box that only returns function values, then subgradients could be approximated numerically using the techniques developed in [2, 13, 15, 27]. These technique will return approximate subgradients that satisfy assumption (2.3), but are not necessarily ε-subgradients. The method of the present work changes the need for ε-subgradients, to approximate subgradients that satisfy assumption (2.3).

Shortly before [26] was published, a similar technique was presented in [8]. This version does not require the inf-compactness condition that we do, nor does it impose the existence of a minimum function value. However, it too requires the approximate subgradients to be ε-subgradients. A few years later [40] and [42] presented similar solvers, also as subroutines within minimization algorithms. Again, the convergence results rest upon ε-subgradients and model functions that are constructed using supporting hyperplanes of the objective function.

The algorithmic pattern in [9] is much more general in nature; it is applicable to many types of bundle methods and oracles. In [9], the authors go into detail about the variety of oracles in use (upper, dumb lower, controllable lower, asymptotically exact, and others), and the resulting

particular bundle methods that they inspire. The oracles themselves are more generalized as well, in that they deliver approximate function values instead of exact ones. The approximate subgradient is then determined based on the approximate function value, and thus is dependent on two parameters of inexactness rather than one. The algorithm iteratively calculates proximal points as does ours, but does not include the subgradient correction step.

Both [39] and [19] address the issue of non-convexity. The algorithm in [39] splits the prox-parameter in two: a local convexification parameter and a new model prox-parameter. It calls an oracle that delivers an approximate function value and approximate subgradient, which are used to construct a piecewise-linear model function. That function is then shifted down to ensure that it is a cutting-planes model. In [19] the same parameter-splitting technique is employed to deal with nonconvex functions, and the oracle returns both inexact function values and inexact subgradients. The notable difference here is that the approximate subgradient is not an ε-subgradient; it is any vector that is within ε of the subdifferential of the model function at the current point. This is the same assumption that we employ in our version. However, non-convexity forces the algorithms to involve prox-parameter corrections that obscure any prox-point subroutine. (Indeed, it is unclear if a prox-point is actually computed as a subroutine in these methods, or if the methods only use proximal directions to seek descent.)

In all of the above methods except for the last, the model function is built using lower-estimating cutting planes. In this work, the goal is to avoid this requirement and extend the class of inexact subgradients that can be used in these types of algorithms. The tilt-and-correct step in our method ensures that the model function and the objective function coincide at the prox-centre, which we show is sufficient to prove convergence in the next section. Although the last method mentioned above is for the nonconvex case and uses approximate function values, it is the most similar to the one in the present work, as it does not rely on ε-subgradients. The differentiating aspect in that method, as in all of the aforementioned methods, is that it does not make any slope-improving correction to the subgradient.

4 Convergence

To prove convergence of this routine, we need several lemmas that are proved in the sequel. Ultimately, we prove that the algorithm converges to the true proximal point of f with a maximum error of $\frac{\varepsilon}{r}$. Throughout this section, we denote $\operatorname{Prox}_f^r(z)$ by x^*. To begin, we establish some properties of ϕ_k^ε.

Lemma 4.1. *Let* $\phi_k^\varepsilon = \max_{i \in \mathcal{B}_k} \left\{ f_i + g_i^{\varepsilon\top}(x - x_i) \right\}$. *Then for all* k,

a) ϕ_k^ε *is a convex function,*

b) $\phi_k^\varepsilon(z) = f(z)$,

c) $\phi_{k+1}^\varepsilon(x) \geq \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x - x_{k+1})$ *for all* $x \in \mathbb{R}^n$,

d) $\phi_k^\varepsilon(x) \geq f(x_k) + g_k^{\varepsilon\top}(x - x_k)$ *for all* $x \in \mathbb{R}^n$, *and*

e) ϕ_k^ε is $(K+\varepsilon)$-Lipschitz.

Proof. a) Since ϕ_k^ε is the maximum of a finite number of convex functions, ϕ_k^ε is convex by [3, Proposition 8.14].

b) We have that $\phi_0^\varepsilon(z) = f(z)$ by definition. Then for any $k > 0$, the tilt-correct step guarantees that

$$f(x_k) + g_k^{\varepsilon\top}(z - x_k) \leq f(z),$$

so by equation (3.1) we have $\phi_k^\varepsilon(z) \leq f(z)$ for all k. Thus, we need only concern ourselves with the new linear function at iteration k. For $k > 0$, either $f_k + \tilde{g}_k^{\varepsilon\top}(z - x_k) > f(z)$ or $f_k + \tilde{g}_k^{\varepsilon\top}(z - x_k) \leq f(z)$. In the former case, we make the correction to $\tilde{g}_k^\varepsilon$ so that $f_k + g_k^{\varepsilon\top}(z - x_k) = f(z)$. As for the aggregate subgradient bundle element, we have that $r(z - x_k) \in \partial\phi_k^\varepsilon(x_{k+1})$ and ϕ_k^ε is convex, so that $\phi_{k-1}^\varepsilon(x_k) + r(z - x_k)(x - x_k) \leq \phi_k^\varepsilon(x)$ for all $x \in \mathbb{R}^n$. In particular, $\phi_{k-1}^\varepsilon(x_k) + r(z - x_k)(x - x_k) \leq \phi_k^\varepsilon(z) = f(z)$. Therefore,

$$f(z) = \phi_0^\varepsilon(z) \leq \phi_k^\varepsilon(z) = \max_{i\in\mathcal{B}_k} \left\{ f_i + g_i^{\varepsilon\top}(z - x_i) \right\} \leq f(z),$$

which proves (b).

c) Since $(x_{k+1}, \phi_k^\varepsilon(x_{k+1}), r(z - x_{k+1})) \in \mathcal{D}_{k+1}$, we have

$$\begin{aligned}\phi_{k+1}^\varepsilon(x) &= \max_{i\in\mathbb{B}_{k+1}} \left\{ f_i + g_i^{\varepsilon\top}(x - x_i) \right\} \\ &\geq \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x - x_{k+1}).\end{aligned}$$

d) This is true by definition of ϕ_k^ε.

e) We know that f is locally K-Lipschitz, and by Lemma 3.1, for each k we have that g_k^ε is within distance ε of $\partial f(x_k)$. Therefore, ϕ_k^ε is (globally) $(K+\varepsilon)$-Lipschitz.

□

Remark 4.2. *Lemma 4.1 makes strong use of the aggregate bundle element (indexed by -1) to prove part (c). It is possible to avoid the aggregate bundle element by setting $B_{k+1} = B_k \cup \{k\}$ at every iteration. To see this note that the tilt-correction step will only ever alter g_i^ε at iteration i. As $\phi_k^\varepsilon(x) \geq \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x - x_{k+1})$, if $B_{k+1} = B_k \cup \{k\}$, then $\phi_{k+1}^\varepsilon(x) \geq \phi_k^\varepsilon(x)$ provides the necessary information to ensure Lemma 4.1(c) still holds.*

Next, we show that at every iteration the distance between the approximal point and the true prox-point is bounded by a function of the distance between $f(x_{k+1})$ and $\phi_k^\varepsilon(x_{k+1})$. This immediately leads to an understanding of the stopping criterion.

Lemma 4.3. *At every iteration of the algorithm, the distance between the proximal point of the piecewise-linear function, $x_{k+1} = \mathrm{Prox}_{\phi_k^\varepsilon}^r(z)$, and the proximal point of the objective function, $x^* = \mathrm{Prox}_f^r(z)$, satisfies*

$$\mathrm{dist}(x^*, x_{k+1}) \leq \sqrt{\frac{f(x_{k+1}) - \phi_k^\varepsilon(x_{k+1}) + \frac{\varepsilon^2}{4r}}{r}} + \frac{\varepsilon}{2r}. \tag{4.1}$$

Proof. Since $x_{k+1} = \mathrm{Prox}^r_{\phi_k^\varepsilon}(z)$, we have that

$$\phi_k^\varepsilon(x_{k+1}) \leq \phi_k^\varepsilon(x) + \frac{r}{2}\|z - x\|^2 \text{ for all } x \in \mathbb{R}^n\,.$$

By equation (3.7), for $i = k+1 \in \mathcal{B}_{k+1}$ we have

$$f(x) + \varepsilon\|x - x_{k+1}\| \geq \phi_{k+1}^\varepsilon(x) \text{ for all } x \in \mathbb{R}^n, \tag{4.2}$$

which, by Lemma 4.1 (c), results in

$$f(x) + \varepsilon\|x - x_{k+1}\| \geq \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x - x_{k+1}) \text{ for all } x \in \mathbb{R}^n\,. \tag{4.3}$$

In particular, we have

$$f(x^*) + \varepsilon\|x^* - x_{k+1}\| \geq \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x^* - x_{k+1}). \tag{4.4}$$

Since $x^* = \mathrm{Prox}^r_f(z)$, we have $r(z - x^*) \in \partial f(x^*)$. Then

$$f(x) \geq f(x^*) + r(z - x^*)^\top(x - x^*) \text{ for all } x \in \mathbb{R}^n, \tag{4.5}$$

thus, we have

$$f(x_{k+1}) \geq f(x^*) + r(x^* - z)^\top(x^* - x_{k+1}). \tag{4.6}$$

Adding equations (4.4) and (4.6) yields

$$\begin{aligned}
f(x_{k+1}) + \varepsilon\|x^* - x_{k+1}\| &\geq \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1} - z + x^*)^\top(x^* - x_{k+1}),\\
f(x_{k+1}) - \phi_k^\varepsilon(x_{k+1}) &\geq r\|x^* - x_{k+1}\|^2 - \varepsilon\|x^* - x_{k+1}\|,\\
&= r\left[\|x^* - x_{k+1}\|^2 - \frac{\varepsilon}{r}\|x^* - x_{k+1}\| + \frac{\varepsilon^2}{4r^2} - \frac{\varepsilon^2}{4r^2}\right],\\
&= r\left[\|x^* - x_{k+1}\| - \frac{\varepsilon}{2r}\right]^2 - \frac{\varepsilon^2}{4r}.
\end{aligned}$$

Isolating the squared term above and taking the square root of both sides, we have

$$\begin{aligned}
\sqrt{\frac{f(x_{k+1}) - \phi_k^\varepsilon(x_{k+1}) + \frac{\varepsilon^2}{4r}}{r}} &\geq \left|\|x^* - x_{k+1}\| - \frac{\varepsilon}{2r}\right| \geq \|x^* - x_{k+1}\| - \frac{\varepsilon}{2r},\\
\mathrm{dist}(x^*, x_{k+1}) &\leq \sqrt{\frac{f(x_{k+1}) - \phi_k^\varepsilon(x_{k+1}) + \frac{\varepsilon^2}{4r}}{r}} + \frac{\varepsilon}{2r}.
\end{aligned}$$

□

Remark 4.4. *Lemma 4.3 not only sets up our analysis of the stopping criterion, but is also provides the necessary insight to understand the algorithm's output if premature stopping is invoked. In particular, if the algorithm is used as a subroutine inside of larger method, and the larger method stops the subroutine (perhaps because desirable decrease is detected), then equation* (4.1) *still applies. As such, the optimizer can still compute an error bound on the distance of the output to the true prox-point.*

Corollary 4.5. *If the stopping criterion is satisfied, then* $\operatorname{dist}(x^*, x_{k+1}) \leq \mathtt{s_{tol}} + \frac{\varepsilon}{r}$.

Proof. Substituting the stopping criterion into inequality (4.1) yields

$$\operatorname{dist}(x^*, x_{k+1}) \leq \sqrt{\mathtt{s_{tol}}^2 + \frac{\varepsilon^2}{4r^2}} + \frac{\varepsilon}{2r} \leq \mathtt{s_{tol}} + \frac{\varepsilon}{r}.$$

□

Corollary 4.5 is our first convergence result, showing that if for some k, $\phi_k^\varepsilon(x_{k+1})$ comes close enough to $f(x_{k+1})$ to trigger the stopping condition of the algorithm, then x_{k+1} is within a fixed distance of the true prox-point. Now we aim to prove that the stopping condition will always be activated at some point, and the algorithm will not run indefinitely. We begin with Lemma 4.6, which shows that if at any iteration the new point is equal to the previous one, the stopping condition is triggered and the approximal point is within $\frac{\varepsilon}{r}$ of x^*.

Lemma 4.6. *If* $x_{k+2} = x_{k+1}$ *for some* k*, then the algorithm stops, and* $\operatorname{dist}(x_{k+2}, x^*) \leq \frac{\varepsilon}{r}$.

Proof. We have $\phi_{k+1}^\varepsilon(x) = \max_{i \in \mathcal{B}_{k+1}} \{f_i + g_i^{\varepsilon\top}(x - x_i)\}$, so in particular $\phi_{k+1}^\varepsilon(x_{k+1}) = \max_{i \in \mathcal{B}_{k+1}} \{f_i + g_i^{\varepsilon\top}(x_{k+1} - x_i)\}$. Since $k+1 \in \mathcal{B}_{k+1}$, $f(x_{k+1}) + g_{k+1}^{\varepsilon\top}(x_{k+1} - x_{k+1}) = f(x_{k+1})$. Hence, $\phi_{k+1}^\varepsilon(x_{k+1}) \geq f(x_{k+1})$, which if $x_{k+1} = x_{k+2}$ is equivalent to $\phi_{k+1}^\varepsilon(x_{k+2}) \geq f(x_{k+2})$, and the stopping criterion is satisfied. Then by Lemma 4.3, we have

$$\begin{aligned}
\operatorname{dist}(x^*, x_{k+2}) &\leq \sqrt{\frac{f(x_{k+2}) - \phi_{k+1}^\varepsilon(x_{k+2}) + \frac{\varepsilon^2}{4r}}{r}} + \frac{\varepsilon}{2r} \\
&\leq \sqrt{\frac{\varepsilon^2}{4r^2}} + \frac{\varepsilon}{2r} \\
&= \frac{\varepsilon}{r}.
\end{aligned}$$

□

Next, we prove convergence within $\frac{\varepsilon}{r}$ in the case that the stopping condition is never triggered and the algorithm does not stop. We show that this is true by establishing Lemmas 4.7 through 4.9, which lead to Theorem 4.10, the main convergence result.

Lemma 4.7. *Suppose the algorithm never stops. Then the function*

$$\Phi(k) := \phi_k^\varepsilon(x_{k+1}) + \frac{r}{2}\|z - x_{k+1}\|^2$$

is strictly increasing and bounded above.

Proof. Recall that $\phi_k^\varepsilon(z) = f(z)$ by Lemma 4.1 (b). Since x_{k+1} is the proximal point of ϕ_k^ε at z, we have

$$\phi_k^\varepsilon(x_{k+1}) + \frac{r}{2}\|z - x_{k+1}\|^2 \leq \phi_k^\varepsilon(z) + \frac{r}{2}\|z - z\|^2 = f(z).$$

Therefore, $\Phi(k)$ is bounded above by $f(z)$ for all k. Define

$$L_k(x) := \phi_k^\varepsilon(x_{k+1}) + \frac{r}{2}\|z - x_{k+1}\|^2 + \frac{r}{2}\|x - x_{k+1}\|^2.$$

Since $x_{k+1} = \mathrm{Prox}^r_{\phi_k^\varepsilon}(z)$, we have

$$L_k(x_{k+1}) = \phi_k^\varepsilon(x_{k+1}) + \frac{r}{2}\|z - x_{k+1}\|^2 \le \phi_k^\varepsilon(z) = f(z).$$

By Lemma 4.1 (c) with $x = x_{k+2}$, we have

$$\phi_{k+1}^\varepsilon(x_{k+2}) \ge \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x_{k+2} - x_{k+1}). \tag{4.7}$$

Using inequality (4.7) we have

$$\begin{aligned} L_{k+1}(x_{k+2}) &= \phi_{k+1}^\varepsilon(x_{k+2}) + \frac{r}{2}\|z - x_{k+2}\|^2 \\ &\ge \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x_{k+2} - x_{k+1}) + \frac{r}{2}\|z - x_{k+2}\|^2 \\ &= L_k(x_{k+1}) + r(z - x_{k+1})^\top(x_{k+2} - x_{k+1}) + \frac{r}{2}\|z - x_{k+2}\|^2 - \frac{r}{2}\|z - x_{k+1}\|^2. \end{aligned}$$

Expanding the norms above, we have

$$\begin{aligned} &\frac{r}{2}\|z - x_{k+2}\|^2 - \frac{r}{2}\|z - x_{k+1}\|^2 \\ =&\frac{r}{2}\left[\|x_{k+2}\|^2 - 2x_{k+2}^\top x_{k+1} + \|x_{k+1}\|^2 - 2z^\top(x_{k+2} - x_{k+1}) + 2x_{k+2}^\top x_{k+1} - 2\|x_{k+1}\|^2\right] \\ =&\frac{r}{2}\|x_{k+2} - x_{k+1}\|^2 - r(z - x_{k+1})^\top(x_{k+2} - x_{k+1}). \end{aligned}$$

This gives us that

$$\begin{aligned} &L_{k+1}(x_{k+2}) \\ \ge&L_k(x_{k+1}) + r(z - x_{k+1})^\top(x_{k+2} - x_{k+1}) + \frac{r}{2}\|x_{k+2} - x_{k+1}\|^2 - r(z - x_{k+1})^\top(x_{k+2} - x_{k+1}) \\ =&L_k(x_{k+1}) + \frac{r}{2}\|x_{k+2} - x_{k+1}\|^2. \end{aligned}$$

Since $x_{k+2} \ne x_{k+1}$ for all k by Lemma 4.6, the equality above becomes

$$\phi_{k+1}^\varepsilon(x_{k+2}) + \frac{r}{2}\|z - x_{k+2}\|^2 \ge \phi_k^\varepsilon(x_{k+1}) + \frac{r}{2}\|z - x_{k+1}\|^2 + \frac{r}{2}\|x_{k+2} - x_{k+1}\|^2,$$

which by the definition of Φ yields

$$\Phi(k+1) \ge \Phi(k) + \frac{r}{2}\|x_{k+2} - x_{k+1}\|^2 > \Phi(k). \tag{4.8}$$

Therefore, $\Phi(k)$ is a strictly increasing function. □

Corollary 4.8. *Suppose the algorithm never stops. Then* $\lim_{k\to\infty} \|x_{k+1} - x_k\| = 0$.

Proof. By inequality (4.8), we have

$$\begin{aligned} 0 &\leq \frac{r}{2}\|x_{k+2} - x_{k+1}\|^2 \\ &\leq \Phi(k+1) - \Phi(k). \end{aligned}$$

By Lemma 4.7, both terms on the right-hand side above converge, and they converge to the same place. Then

$$0 \leq \lim_{k\to\infty} \frac{r}{2}\|x_{k+2} - x_{k+1}\|^2 \leq 0,$$

and since $r \neq 0$, we have that $\|x_{k+2} - x_{k+1}\| \to 0$. □

We point out here that the sequence $\{x_k\}$ has a convergent subsequence. This is because the iterates are contained in a compact set (a ball about z), so that the Bolzano-Weierstrass theorem applies. We use this fact to prove the results that follow.

Lemma 4.9. *Let f be locally K-Lipschitz. Suppose the algorithm never stops. Then for any accumulation point p of $\{x_k\}$, $\lim_{j\to\infty} \phi^\varepsilon_{k_j}(x_{k_j+1}) = f(p)$, where $\{x_{k_j}\}$ is any subsequence converging to p.*

Proof. We have $f_{k+1} \geq \phi^\varepsilon_k(x_{k+1})$ (otherwise, the stopping criterion is satisfied and the algorithm stops). By Lemma 4.1 (e), ϕ^ε_k is $(K+\varepsilon)$-Lipschitz:

$$\|\phi^\varepsilon_k(x_{k+1}) - \phi^\varepsilon_k(x_k)\| \leq (K+\varepsilon)\|x_{k+1} - x_k\|,$$

and by Corollary 4.8 we have that $\lim_{k\to\infty} \|\phi^\varepsilon_k(x_{k+1}) - \phi^\varepsilon_k(x_k)\| = 0$. We also have

$$\begin{aligned} f(x_{k+1}) &\geq \phi^\varepsilon_k(x_{k+1}) - \phi^\varepsilon_k(x_k) + \phi^\varepsilon_k(x_k) \\ &\geq \phi^\varepsilon_k(x_k) - (K+\varepsilon)\|x_{k+1} - x_k\|. \end{aligned}$$

By Lemma 4.1 (d) with $x = x_k$,

$$f(x_{k+1}) \geq \phi^\varepsilon_k(x_k) - (K+\varepsilon)\|x_{k+1} - x_k\| \geq f(x_k) - (K+\varepsilon)\|x_{k+1} - x_k\|. \tag{4.9}$$

Select any subsequence $\{x_{k_j}\}$ such that $\lim_{j\to\infty} x_{k_j} = p$. Since $\lim_{j\to\infty} \|x_{k_j+1} - x_{k_j}\| = 0$ by Corollary 4.8, we have that $\lim_{j\to\infty} x_{k_j+1} = p$ as well. Hence, taking the limit of inequality (4.9) as $j \to \infty$, and employing Corollary 4.8, we have

$$f(p) \geq \lim_{j\to\infty} \phi^\varepsilon_{k_j}(x_{k_j}) \geq f(p).$$

Therefore, $\lim_{j\to\infty} \phi^\varepsilon_{k_j}(x_{k_j}) = f(p)$, and since $\lim_{j\to\infty} \|\phi^\varepsilon_{k_j}(x_{k_j+1}) - \phi^\varepsilon_{k_j}(x_{k_j})\| = 0$, we have that

$$\lim_{j\to\infty} \phi^\varepsilon_{k_j}(x_{k_j+1}) = f(p).$$

□

Now the stage is set for the following theorem, which proves that the algorithm converges to a vector that is within a fixed distance of $\mathrm{Prox}_f^r(z)$.

Theorem 4.10. *Let f be locally K-Lipschitz. Suppose the algorithm never stops. Then for any accumulation point p of $\{x_k\}$, $\mathrm{dist}(x^*, p) \leq \frac{\varepsilon}{r}$.*

Proof. By inequality (4.3),

$$f(x) + \varepsilon\|x - x_{k+1}\| \geq \phi_k^\varepsilon(x_{k+1}) + r(z - x_{k+1})^\top(x - x_{k+1}) \text{ for all } x \in \mathbb{R}^n\,.$$

Select any subsequence $\{x_{k_j}\}$ such that $\lim_{j\to\infty} x_{k_j} = p$. Taking the limit as $j \to \infty$ and using Corollary 4.8 and Lemma 4.9, we have

$$\begin{aligned} f(x) + \varepsilon\|x - p\| &= \lim_{j\to\infty} f(x) + \varepsilon\|x - x_{k_j+1}\| \\ &\geq \lim_{j\to\infty} \left[\phi_{k_j}^\varepsilon(x_{k_j+1}) + r(z - x_{k_j+1})^\top(x - x_{k_j+1})\right] \\ &= f(p) + r(z - p)^\top(x - p) \text{ for all } x \in \mathbb{R}^n\,. \end{aligned} \tag{4.10}$$

By equation (4.5), we have

$$f(p) \geq f(x^*) + r(x^* - z)^\top(x^* - p). \tag{4.11}$$

By equation (4.10), in particular we have

$$f(x^*) + \varepsilon\|x^* - p\| \geq f(p) + r(z - p)^\top(x^* - p). \tag{4.12}$$

Adding equations (4.11) and (4.12) yields

$$\begin{aligned} \varepsilon\|x^* - p\| &\geq r(x^* - z + z - p)^\top(x^* - p) \\ &= r\|x^* - p\|^2. \end{aligned}$$

Therefore, $\mathrm{dist}(x^*, p) \leq \frac{\varepsilon}{r}$. □

Lastly, we show that the algorithm will always terminate. With the proof of Theorem 4.11 below, we will have proved that the algorithm does not run indefinitely, and that when it stops the output is within tolerance of the point we seek.

Theorem 4.11. *Let f be locally K-Lipschitz. If $\|x_{k+1} - x_k\| < \frac{r\,\mathtt{Stol}^2}{K+2\varepsilon}$, then the stopping condition is satisfied and the algorithm stops.*

Proof. By Lemma 4.1 (d) with $x = x_{k+1}$, we have

$$\begin{aligned} \phi_k^\varepsilon(x_{k+1}) &\geq f(x_k) + g_k^{\varepsilon\top}(x_{k+1} - x_k) \\ f(x_k) - \phi_k^\varepsilon(x_{k+1}) &\leq -g_k^{\varepsilon\top}(x_{k+1} - x_k) \\ &\leq \|g_k^\varepsilon\|\|x_{k+1} - x_k\|. \end{aligned}$$

By Lemma 4.1 (e), ϕ_k^ε is $(K+\varepsilon)$-Lipschitz. Hence, $\|g_k^\varepsilon\| \leq K+\varepsilon$, and

$$f(x_{k+1}) - \phi_k^\varepsilon(x_{k+1}) \leq (K+\varepsilon)\|x_{k+1} - x_k\|.$$

Therefore, if $\|x_{k+1} - x_k\| < \frac{r\,\mathtt{s_{tol}}^2}{K+\varepsilon}$, then

$$f(x_{k+1}) - \phi_k^\varepsilon(x_{k+1}) < (K+\varepsilon)\frac{r\,\mathtt{s_{tol}}^2}{K+\varepsilon}$$
$$\frac{f(x_{k+1}) - \phi_k^\varepsilon(x_{k+1})}{r} < \mathtt{s_{tol}}^2.$$

□

With this, we have that the sequence $\{x_k\}$ generated by the algorithm must have an accumulation point p that is within $\frac{\varepsilon}{r}$ of $\mathrm{Prox}_f^r(z)$, and the algorithm will always terminate at some point x_k such that $\|x_k - x^*\| \leq s_{\mathtt{tol}} + \frac{\varepsilon}{r}$.

5 Numerical Tests

In this section, we present the results of a number of numerical tests performed using this algorithm. The tests were run using MATLAB version 8.4.0.150421 (R2014b), on a 2.8 GHz Intel Core 2 Duo processor with a 64-bit operating system.

5.1 Bundle variants

We set $r = 1$, and compare four bundle variants: 3-bundle, $(k+2)$-bundle, active-bundle, and almost-active-bundle. In the 3-bundle variant, each iteration uses the three bundle elements indexed by $\mathcal{B}_k = \{-1, 0, k\}$. In the $(k+2)$-bundle variant, we keep all the bundle elements from each previous iteration, and add the new one. So the bundle index set is $\mathcal{B}_k = \{-1, 0, 1, 2, 3, \ldots, k\}$, for a total of $k+2$ elements.[1] In the active-bundle variant, we keep the indices $-1, 0, k$, and add in any indices i that satisfy

$$\phi_k^\varepsilon(x_{k+1}) = \phi_k^\varepsilon(x_i) + g_i^{\varepsilon\top}(x_{k+1} - x_i).$$

These are the linear functions that are active at iteration $k-1$. Finally, the almost-active-bundle keeps the indices $-1, 0, k$, and adds in any indices that satisfy

$$\phi_k^\varepsilon(x_{k+1}) < \phi_k^\varepsilon(x_i) + g_i^{\varepsilon\top}(x_{k+1} - x_i) + 10^{-6}.$$

These are the linear functions that are 'almost' active at iteration $k-1$, which allows for software rounding errors to be discounted.

[1] The index -1 is not necessary for convergence when indices 0 through k are in the bundle. However, since our convergence analysis focused on the aggregate subgradient, we keep index -1 in every bundle variant.

5.2 Max-of-quadratics Tests

For our first tests, we use a max-of-quadratics generator to create problems. Each problem to be solved is a randomly generated function $f(x) := \max\{q_1(x), q_2(x), ..., q_m(x)\}$, where q_i is convex quadratic for all i. There are four inputs to the generator function: n, nf, nf_{x^*}, and nf_z. The number n is the dimension of x, nf is the number of quadratic functions used, nf_{x^*} is the number of quadratic functions that are active at the true prox-point, and nf_z is the number of quadratic functions that are active at the point z. These features can all be controlled, as seen in [17]. The approximate subgradient is constructed by finding the gradient of the first active quadratic functions, and giving it some random error of magnitude less than ε by using the `randsphere` routine.[1] That is,

$$\tilde{g}_k^{\varepsilon} = \nabla f_i(x) + \varepsilon\, \texttt{randsphere}(1, n, 1),$$

where i is the first active index. Though we use random error, the random function is seeded, so that the results are reproducible. The primal quadratic program is solved using MATLAB's `quadprog` solver.

Two sets of problems were generated: low-dimension trials and high-dimensions trials. For the low-dimension trials the Hessians of the q_i functions were dense and we attempted to solve ten problems at each possible state of $n \in \{4, 10, 25\}$, $nf \in \left\{1, \lceil \frac{n}{3} \rceil, \lceil \frac{2n}{3} \rceil, n\right\}$, $nf_{x^*} \in \left\{1, \lceil \frac{n}{3} \rceil, \lceil \frac{2n}{3} \rceil, n\right\}$, $nf_z \in \left\{1, \lceil \frac{n}{3} \rceil, \lceil \frac{2n}{3} \rceil, n\right\}$, with four variants of the algorithm and three subgradient error levels: $\varepsilon \in \{0, \mathrm{s}_{\texttt{tol}}, 10\,\mathrm{s}_{\texttt{tol}}\}$. This amounts to a total of 2700 problems[2] attempted by each of the four variants, 10,800 runs altogether. For the high-dimension trials the Hessians of the q_i functions were sparse, with 95% density of zeros. In high-dimension, we attempted two problems at each possible state of $n \in \{100, 200\}$ and the same conditions as above on the rest of the variables, for another 360 problems attempted by each variant, 1440 runs altogether.

The performance of the variants is presented in two performance profile graphs and a table of averages. The table provides average CPU times, average number of iterations, and average number of tilt-corrections for each of the four bundle variants. One performance profile graph is for low dimension $n = 4, 10, 25$, and the other is for high dimension $n = 100, 200$. A performance profile is a standard method of comparing algorithms, using the best-performing algorithm for each category of test as the basis for comparison. Here, we compare the four variants based on CPU time used, and on number of iterations used, to solve each problem. We set $\mathrm{s}_{\texttt{tol}} = 10^{-3}$ for all tests, and declare a problem as solved if the stopping criterion is properly triggered within $100n$ iterations for the low-dimension set, and $20n$ for the high-dimension set. The x-axis, on a natural logarithmic scale, is the factor τ of the best possible ratio, and on the y-axis is the percentage of problems solved within a factor of τ of the solve time of the best solve time for a given function.

In the low-dimension case, we see from Figure 1 that the $(k+2)$-bundle is the most efficient, and the almost active bundle follows close behind. The 3-bundle and active bundle coincide almost

[1] Randsphere is a MATLAB function that outputs a uniformly distributed set of random points in the interior of an n-dimensional hypersphere of radius r with center at the origin. The code is found at `http://www.mathworks.com/matlabcentral/fileexchange/9443-random-points-in-an-n-dimensional-hypersphere/content/randsphere.m`.

[2] These totals take into account that nf_z and nf_{x^*} cannot exceed nf at any stage. When $n = 4$, we have fewer possibilities due to the low values of $\lceil \frac{n}{3} \rceil$ and $\lceil \frac{2n}{3} \rceil$.

exactly; their curves overlap. Comparing the results for each error level in terms of CPU time vs. number of iterations (each pair of side-by-side graphs), there is no notable difference. With the 3-bundle and active bundle, about 22% of the problems timed out, meaning that the upper bound of $100n$ iterations was reached before the stopping condition was triggered. With the almost active bundle, that figure drops to about 10%, and the $(k+2)$-bundle solved all of the problems. However, it is interesting to note that about 95% of those timed-out problems still ended with $\mathrm{dist}(x_k, x^*) \leq \mathrm{s}_{\mathtt{tol}} + \frac{\varepsilon}{r}$.

In the high-dimension case, Figure 2 tells us that the 3-bundle and active bundle perform well in terms of CPU time for the problems they solved, however they were only able to solve about two thirds of the problems within the allotted time limit. The $(k+2)$-bundle took more time, and the almost active bundle much more still, but the former solved all the problems and the latter solved 97% of them. In terms of number of iterations, the same general pattern as was found in the low-dimension case appears. The only difference is that the almost active bundle uses noticeably more iterations to complete the job, but the curve still lies below that of the $(k+2)$-bundle and above the other two. The average CPU time, number of iterations, and number of tilt-corrections for both sets of problems are displayed in Table 1. It is interesting to note that for this type of problem the tilt-correction was used sparingly in low dimension, and in high dimension it was not needed at all.

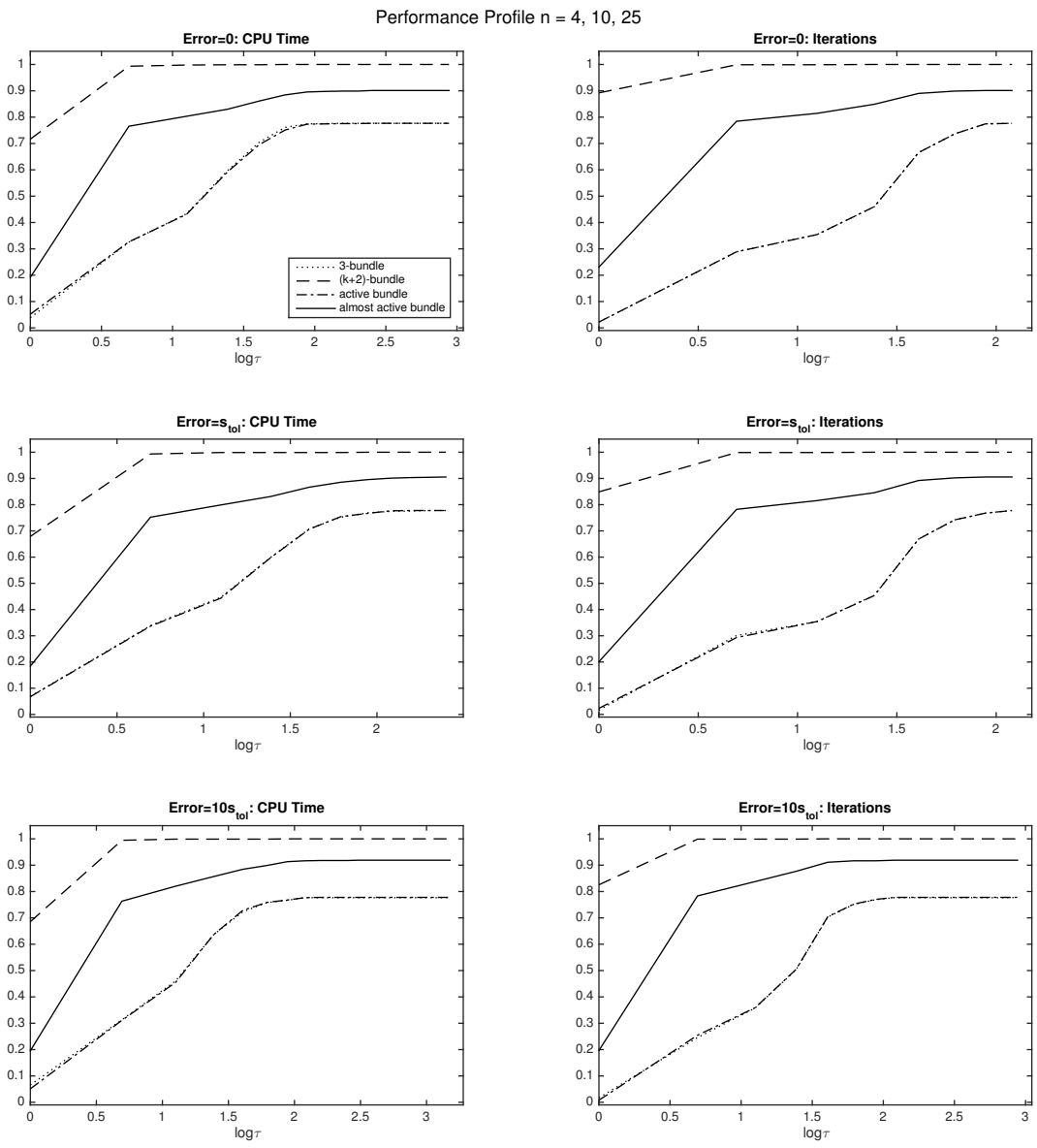

Figure 1: Low-dimension performance profile.

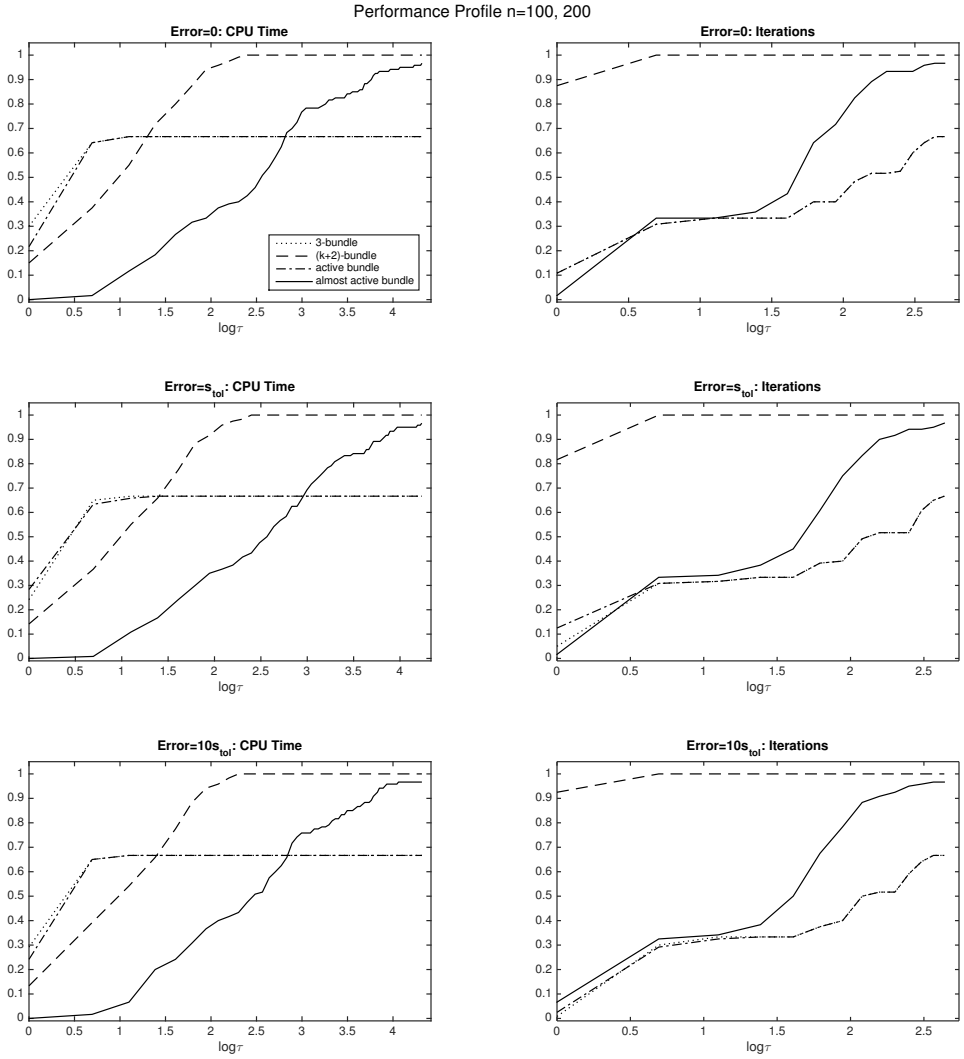

Figure 2: High-dimension performance profile.

Bundle	Average CPU Time	Average Iterations	Average Tilt-corrections
3-bundle	n low: 1.36s n high: 19.90s	n low: 191 n high: 2034	n low: 0.0122 n high: 0
$(k+2)$-bundle	n low: 0.61s n high: 66.31s	n low: 45 n high: 250	n low: 0.0015 n high: 0
Active bundle	n low: 1.39s n high: 19.90s	n low: 191 n high: 2035	n low: 0.0111 n high: 0
Almost active	n low: 1.96s n high: 531.06s	n low: 109 n high: 1438	n low: 0.003 n high: 0

Table 1: Average values among the four bundle variants.

5.3 Derivative-free optimization tests

To see how the algorithm might preform in the setting of derivative-free optimization, we selected a test set of ten functions and ran the algorithm using the approximate subgradient developed in robust approximate gradient sampling algorithm [15]. The robust approximate gradient sampling algorithm approximates a subgradient of the objective function by using function evaluations and linear interpolation (details can be found in [15]). Most of the problems are taken from [30], some of which were adjusted slightly to make them convex, further details appear in Appendix A. A brief description of the functions is found in Table 2; more details are available from the authors and from [30].

Function	Dimension	Description	Reference
P alpha	2	max of 2 quadratics	
DEM	2	max of 3 quadratics	[30, Problem 3.5]
Wong 3 (adjusted)	20	max of 18 quadratics	[30, Problem 2.21]
CB 2	2	max of exponential, quartic, quadratic	[30, Problem 2.1]
Mifflin 2	2	absolute value + quadratic	[30, Problem 3.9]
EVD 52 (adjusted)	3	max of quartic, quadratics, linears	[30, Problem 2.4]
OET 6 (adjusted)	2	max of exponentials	[30, Problem 2.12]
MaxExp	12	max of exponentials	
MaxLog	30	max of negative logs	
Max10	10	max of 10^{th}-degree polynomials	

Table 2: Set of test problems using simplex gradients.

Each bundle variant was run 100 times on each test problem. In all cases, the algorithm located the correct proximal point. The average CPU time, average number of iterations, and average number of tilt-correction steps used appears in Table 3.

Bundle	Average CPU Time	Average Iterations	Average Tilt-corrections
3-bundle	0.85s	147	1.01
$(k+2)$-bundle	1.17s	45	1.18
Active bundle	0.84s	146	0.93
Almost active	1.18s	59	0.79

Table 3: Average values among the four bundle variants.

As before, in terms of iterations, the $(k+2)$-bundle and the almost active variants greatly outperform the 3-bundle and active bundle variants. (In terms of CPU time, all methods used approximately 1 second per problem.) However, unlike the previous test set, this test set required an average of 1 tilt-correction step per problem. This suggests that the minimalist use of the tilt-correction steps in the previous test set is likely due to the structure of the test set.

6 Conclusion

We have presented an approximate-subgradient bundle method for numerically finding the proximal point of a convex function at a given point. The method assumes the availability of an oracle that delivers an exact function value and a subgradient that is within distance ε of the true subdifferential, but does not depend on the approximate subgradient being an ε-subgradient, nor on the model function being a cutting-planes function (one that bounds the objective function from below). The method is proved to converge to the true prox-point within $\mathrm{s}_{\mathtt{tol}} + \frac{\varepsilon}{r}$, where $\mathrm{s}_{\mathtt{tol}}$ is the stopping tolerance of the algorithm, ε is the bound on the approximate subgradient error, and r is the prox-parameter.

From a theoretical standpoint, two questions immediately present themselves. First, could the method be extended to work for nonconvex functions; second, could the method be extended to work in situations where the function value is inexact? Some of the techniques in this paper were inspired by [17, 19], which suggests that the answer to both questions could be positive. However, the extensions are not as straightforward as they may first appear. The key difficulty in extending this algorithm in either of these directions is that, when multiple potential sources of error are present, it is difficult to determine the best course of action. For example, suppose $f_k + \tilde{g}_k^{\varepsilon\top}(z - x_k) - f(z) > 0$. Then, by convexity, we know the inexactness of the subgradient is at fault and we perform a tilt-correction step. However, if the function is nonconvex, then the above inequality could occur due to nonconvexity or due to the inexactness of the subgradient. If the error is due to inexactness, then a tilt-correction step is still the right thing to do. If, on the other hand, the error is due to nonconvexity, then it might be better to redistribute the prox-parameter as suggested in [17]. These issues are equally complex if inexact function values are allowed, and even more complex if both nonconvex functions and inexact function values are permitted.

Another obvious theoretical question would be what happens if ε_k asymptotically tends to 0? Would the algorithm converge to the exact proximal point? Most likely, because past information is preserved in the aggregate subgradient, allowing $\varepsilon_k \to 0$ inside this routine will not result in an

asymptotically exact algorithm. However, this is not a concern, as the purpose of this algorithm is to be used as a subroutine inside a minimization algorithm, where ε_k can be made to tend to zero outside the proximal routine. This has the effect of resetting the routine with ever smaller values of ε, which yields asymptotic exactness.

From a numerical standpoint, we found that when subgradient errors are present, it seems that it is best to keep all the bundle information and use it at every iteration. The other bundle methods also solve the problems, but clearly not as robustly as the biggest bundle does. There are many more numerical questions that could be investigated. For example, the effect of varying the prox-parameter r on the algorithm could be tested, and error bounds/accuracy on the results could be analyzed. Also, our software applied `quadprog` to solve the quadratic sub-problems. While `quadprog` is readily available, it is not recognized as among the top-quality quadratic program solvers. Testing the algorithm using different solvers might produce different results. However, it is also possible that the inexactness of the subgradients might override any improvement in solver quality.

The more immediate goal is to examine this new method in light of the $\mathcal{VU}$-algorithm [33], which alternates between a proximal step and a quasi-Newton step to minimize nonsmooth convex functions. The results of this paper theoretically confirm that the proximal step will work in the setting of inexact subgradient evaluations. Combined with [14], which theoretically examines the quasi-Newton step, the development of a $\mathcal{VU}$-algorithm for inexact subgradients appears achievable.

A Test Problem Details

Three of the test functions taken from [30] were adjusted to make them convex. Four other test functions did not come from [30]. Those details are the following. If a function does not appear in this list, it is unchanged from [30].

(i) *P alpha.* This function is defined $P_\alpha : \mathbb{R}^2 \to \mathbb{R}$,

$$P_\alpha(x) := \max\{x_1^2 + \alpha x_2, x_1^2 - \alpha x_2\}.$$

For this test, α was set to 10,000.

(ii) *Wong 3 adjusted.* From Wong 3 [30, Problem 2.21], the term $x_1 x_2$ was removed from f_1, and the term $-2x_1 x_2$ was removed from f_5. All other sub-functions remained the same.

(iii) *EVD 52 adjusted.* From EVD 52 [30, Problem 2.4], the term $2x_1^3$ in f_5 was changed to $2x_1^4$. All other functions remained the same.

(iv) *OET 6 adjusted.* From OET 6 [30, Problem 2.12], the term $x_1 e^{x_3 t_i}$ was changed to $x_1 + e^{x_3 t_i}$, and the term $x_2 e^{x_4 t_i}$ was changed to $x_2 + e^{x_4 t_i}$ for all i.

(v) *MaxExp.* This function is defined $f : \mathbb{R}^{12} \to \mathbb{R}$,

$$f(x) = \max_{i \in \{1,2,\ldots,12\}} \left\{ i e^{x_i} \right\}.$$

(vi) *MaxLog.* This function is defined $f : \mathbb{R}^{30}_{+} \to \mathbb{R}$,

$$f(x) = \max_{i \in \{1,2,\ldots,30\}} \{-i \ln x_i\}.$$

(vii) *Max10.* This function is defined $f : \mathbb{R}^{10} \to \mathbb{R}$,

$$f(x) = \max_{i \in \{1,2,\ldots,10\}} \left\{i x_i^{10}\right\}.$$

References

[1] F. Al-Khayyal and J. Kyparisis. Finite convergence of algorithms for nonlinear programs and variational inequalities. *Journal of Optimization Theory and Applications*, 70(2):319–332, 1991.

[2] A. M. Bagirov, B. Karasözen, and M. Sezer. Discrete gradient method: derivative-free method for nonsmooth optimization. *Journal of Optimization Theory and Applications*, 137(2):317–334, 2008.

[3] H. H. Bauschke and P. L. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. CMS Books in Mathematics/Ouvrages de Mathématiques de la SMC. Springer, New York, 2011.

[4] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, and C. Sagastizábal. *Numerical optimization: Theoretical and Practical Aspects*. Universitext. Springer-Verlag, Berlin, second edition, 2006.

[5] T. F. Coleman and L. Hulbert. A globally and superlinearly convergent algorithm for convex quadratic programs with simple bounds. Technical report, Cornell University, 1990.

[6] P. L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011.

[7] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*, volume 8. Siam, 2009.

[8] R. Correa and C. Lemaréchal. Convergence of some algorithms for convex minimization. *Mathematical Programming*, 62(1-3):261–275, 1993.

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

[10] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American mathematical Society*, 82(2):421–439, 1956.

[11] M. C. Ferris. Finite termination of the proximal point algorithm. *Mathematical Programming*, 50(1-3):359–366, 1991.

[12] M. L. Fisher. An applications oriented guide to Lagrangian relaxation. *Interfaces*, 15(2):10–21, 1985.

[13] A. M. Gupal. A method for the minimization of almost-differentiable functions. *Cybernetics and Systems Analysis*, 13(1):115–117, 1977.

[14] W. Hare. Numerical analysis of $\mathcal{VU}$-decomposition, $\mathcal{U}$-gradient, and $\mathcal{U}$-Hessian approximations. *SIAM Journal on Optimization*, 24(4):1890–1913, 2014.

[15] W. Hare and J. Nutini. A derivative-free approximate gradient sampling algorithm for finite minimax problems. *Computational Optimization and Applications*, 56(1):1–38, 2013.

[16] W. Hare, J. Nutini, and S. Tesfamariam. A survey of non-gradient optimization methods in structural engineering. *Advances in Engineering Software*, 59:19–28, 2013.

[17] W. Hare and C. Sagastizábal. Computing proximal points of nonconvex functions. *Math. Program.*, 116(1-2, Ser. B):221–258, 2009.

[18] W. Hare and C. Sagastizábal. A redistributed proximal bundle method for nonconvex optimization. *SIAM J. Optim.*, 20(5):2442–2473, 2010.

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

[20] M. Hintermüller. A proximal bundle method based on approximate subgradients. *Computational Optimization and Applications*, 20(3):245–266, 2001.

[21] J. B. Hiriart-Urruty and C. Lemarechal. Convex analysis and minimization algorithms. *Grundlehren der mathematischen Wissenschaften*, 305, 1993.

[22] E. Kalvelagen. Benders decomposition with gams, 2002.

[23] E. Karas, A. Ribeiro, C. Sagastizábal, and M. Solodov. A bundle-filter method for nonsmooth convex constrained optimization. *Mathematical Programming*, 116(1-2):297–320, 2009.

[24] C. T. Kelley. *Iterative methods for optimization*, volume 18. Siam, 1999.

[25] K. C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical programming*, 46(1-3):105–122, 1990.

[26] K. C. Kiwiel. Approximations in proximal bundle methods and decomposition of convex programs. *Journal of Optimization Theory and applications*, 84(3):529–548, 1995.

[27] K. C. Kiwiel. A nonderivative version of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM Journal on Optimization*, 20(4):1983–1994, 2010.

[28] C. Lemaréchal and C. Sagastizábal. Variable metric bundle methods: from conceptual to implementable forms. *Mathematical Programming*, 76(3):393–410, 1997.

[29] A. S. Lewis and S. J. Wright. A proximal method for composite minimization. *Mathematical Programming*, pages 1–46, 2008.

[30] L. Lukšan and J. Vlcek. Test problems for nonsmooth unconstrained and linearly constrained optimization. *Technická zpráva*, 798, 2000.

[31] B. Martinet. Régularisation d'inéquations variationnelles par approximations successives. *Rev. Française Informat. Recherche Opérationnelle*, 4(Ser. R-3):154–158, 1970.

[32] J. Mayer. *Stochastic linear programming algorithms: A comparison based on a model management system*, volume 1. CRC Press, 1998.

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

[34] J.-J. Moreau. Propriétés des applications "prox". *C. R. Acad. Sci. Paris*, 256:1069–1071, 1963.

[35] J.-J. Moreau. Proximité et dualité dans un espace hilbertien. *Bull. Soc. Math. France*, 93:273–299, 1965.

[36] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control Optimization*, 14(5):877–898, 1976.

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

[38] C. Sagastizábal. Composite proximal bundle method. *Mathematical Programming*, 140(1):189–233, 2013.

[39] J. Shen, X.-Q. Liu, F.-F. Guo, and S.-X. Wang. An approximate redistributed proximal bundle method with inexact data for minimizing nonsmooth nonconvex functions. *Mathematical Problems in Engineering*, 2015, 2015.

[40] J. Shen, Z.-Q. Xia, and L.-P. Pang. A proximal bundle method with inexact data for convex nondifferentiable minimization. *Nonlinear Analysis: Theory, Methods & Applications*, 66(9):2016–2027, 2007.

[41] M. V. Solodov. A bundle method for a class of bilevel nonsmooth convex minimization problems. *SIAM Journal on Optimization*, 18(1):242–259, 2007.

[42] W.-Y. Sun, R. J. B. Sampaio, and M. A. B. Candido. Proximal point algorithm for minimization of dc function. *Journal of Computational Mathematics-International Edition*, 21(4):451–462, 2003.

[43] P. Tseng. Approximation accuracy, gradient methods, and error bound for structured convex optimization. *Mathematical Programming*, 125(2):263–295, 2010.

[44] C. Wang and A. Xu. An inexact accelerated proximal gradient method and a dual newton-cg method for the maximal entropy problem. *Journal of Optimization Theory and Applications*, 157(2):436–450, 2013.