Exploring Proximal and Level Bundle Methods for Regularized Support Vector Machines

Bachelor's Project General Mathematics, July 2023

Student: Radovan Rusnák, S4508440
First Supervisor: Prof. dr. J.G. Peypouquet
Second Supervisor: dr. ir. H.J. van Waarde

Abstract: In this paper, we study the Proximal and Level Bundle Methods and their effectiveness in handling the convex non-smooth optimization problem associated with Regularized Support Vector Machine. Many of the traditional optimization algorithms fail to handle non-smooth objective functions, resulting in unstable behaviour and slow convergence. Furthermore, in practical computational problems, dealing with large datasets requires efficient memory management during computations. Bundle Methods tackle these challenges through a distinctive compression mechanism that enables working with a bounded amount of information at each iteration and guaranteed stability. The versatility of Bundle Methods makes them highly applicable to a wide range of optimization problems, and they are currently one of the most promising and popular optimization techniques in many practical applications.

Keywords: Non-smooth optimization, Proximal Bundle Method, Level Bundle Method, Support Vector Machine, proximal operator, projection operator, bundle aggregation, KKT conditions.

Contents

1 Introduction

In both mathematics and the real world, the pursuit of finding the most optimal solution to a given problem is a fundamental and recurring objective. In particular, the problem of minimizing or maximizing a function over some set of constraints has a rich and extensive history in the field of mathematics.

> "Nothing takes place in the world whose meaning is not that of some maximum or minimum."
> (Leonhard Euler)

Traditional optimization methods often struggle when faced with non-smooth objective functions that are prevalent in many real-world applications. This is where Bundle Methods shine, offering a superior approach to tackling challenging, large-scale optimization problems [19]. This paper aims to offer an overview of two prominent classes of Bundle Methods: the Proximal Bundle Method and the Level Bundle Method [21]. We also aim to investigate the application of these methods in solving the problem of Regularized Support Vector Machine.

In Section 2 we provide an introduction to the mathematical tools used throughout this paper. We delve into concepts such as convexity, various types of continuity, the generalized notion of gradients for convex functions, optimality condition for non-smooth optimization problems, and the widely regarded Lagrange Multiplier Theorem and Karush-Kuhn-Tucker (KKT) conditions. We also provide brief comments introducing linear and quadratic programming problems.

Section 3 is dedicated to the description of Bundle Methods, starting with the Cutting-Planes Method and the General Bundle Method. As we progress, we delve into its more specific variants, namely the Proximal and Level Bundle Methods. Furthermore, we introduce an essential aggregation mechanism that enables us to control the amount of information required for constructing the polyhedral model of the objective function in the optimization problem.

In Section 4 we provide an in-depth convergence analysis of the Proximal and Level Bundle Method. This analysis explores the convergence properties of these methods, examining the conditions under which they converge to optimal solutions.

Next, in Section 5, we demonstrate the practical application of Bundle Methods in conjunction with the famous supervised learning model known as the Regularized Support Vector Machine. This machine learning model finds extensive usage in practical applications for solving classification problems, including tasks such as image classification and text categorization [5].

This paper concludes by providing a summary of the subjects that have been explored throughout the entire paper. Additionally, we provide insights into the results obtained from the numerical experiments conducted in Section 5.

2 Theoretical framework

2.1 Optimization theory preliminaries

In this section, we intend to introduce the main mathematical preliminaries needed to describe bundle methods rigorously and to perform the convergence analysis in Section 4. The following definitions, propositions, theorems, and proofs can be found with more detailed explanations in [2], [28] (general optimization theory), [26], [3], [24] (convex analysis), [22] (proximal algorithms), [19] (overview of the Bundle Methods).

Consider the following (convex) optimization problem

$$\begin{cases} \min f(\mathbf{x}) \\ \text{subject to } \mathbf{x} \in X, \end{cases} \tag{2.1}$$

where $f\colon dom(f) \subset \mathbb{R}^n \to \mathbb{R}$ and a closed set $X \subset dom(f) \subset \mathbb{R}^n$. The objective function f is assumed to be *convex*, i.e.,

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y}) \tag{2.2}$$

for every $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and for all $\lambda \in [0, 1]$.

Similarly, we say that the set $X \subset \mathbb{R}^n$ is convex when

$$\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in X, \tag{2.3}$$

for all $\lambda \in [0, 1]$ and for all pairs $(\mathbf{x}, \mathbf{y}) \in X$. In other words, if we pick any two points from the set X, we can draw a line segment between them, which is fully contained in X. Alternatively, we can view convex functions as functions bounded from below by a linear function.

Next, we present a very important concept in convex analysis called *Lipschitz continuity.* Lipschitz continuity implies continuity but also puts a bound on the rate of change of the function f.* It is, therefore, a stronger condition than continuity in its classical sense.

Definition 2.1 *We say that a function $f\colon dom(f) \subset \mathbb{R}^n \to \mathbb{R}$ is Lipschitz continuous at $\mathbf{x} \in dom(f)$ if there exists a constant L such that*

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L\|\mathbf{x} - \mathbf{y}\|, \tag{2.4}$$

for all $\mathbf{x}, \mathbf{y} \in dom(f)$. In particular, f is said to be locally Lipschitz continuous on X if (2.4) holds for all $\mathbf{x}, \mathbf{y} \in X \subset dom(f)$.

Definition 2.2 *Assume we have a function $f\colon dom(f) \subset \mathbb{R}^n \to \mathbb{R}$, we say that*

- *$\mathbf{x}^* \in dom(f)$ is a local minimizer of f if there exists a neighbourhood U of $\mathbf{x}^*$ such that $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for every $\mathbf{x} \in dom(f) \cap U$,*
- *$\mathbf{x}^* \in dom(f)$ is a global minimizer of f if $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for all $\mathbf{x} \in dom(f)$,*
- *$\mathbf{x}^*$ is the unique minimizer if it is the only global minimizer.*

Definition 2.3 *Continuously differentiable, convex function f is said to be strongly convex with modulus of convexity $\mu > 0$ if*

$$f(\lambda \mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}) - \frac{\mu}{2}\lambda(1 - \lambda)\|\mathbf{x} - \mathbf{y}\|^2, \tag{2.5}$$

for all $\lambda \in [0, 1]$.

Using Taylor expansion, we can show that the above definition of a strongly convex function is equivalent to the following definition [28].

Definition 2.4 *Continuously differentiable, convex function f is said to be strongly convex with modulus of convexity $\mu > 0$ if*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\intercal (\mathbf{y} - \mathbf{x}) + \frac{\mu}{2}\|\mathbf{y} - \mathbf{x}\|^2. \tag{2.6}$$

This means, that a strongly convex function can be bounded from below by a quadratic function. Notice that if there is $\mathbf{x}^*$ such that $\nabla f(\mathbf{x}^*)^\intercal = 0$, then $\mathbf{x} = \mathbf{x}^*$ is the unique global minimizer of f.

Next, we present a crucial tool in convex optimization - the subgradient. As it is not uncommon in practice to deal with convex non-smooth objective functions, it might be desirable to somehow extend the notion of differentiability for convex functions.

Definition 2.5 *Consider the convex function $f\colon dom(f) \subset \mathbb{R}^n \to \mathbb{R}$. We say that $\xi \in \mathbb{R}^n$ is a subgradient of f at $\mathbf{x} \in dom(f)$ if*

$$f(\mathbf{z}) \geq f(\mathbf{x}) + \langle \xi, \mathbf{z} - \mathbf{x} \rangle, \tag{2.7}$$

for all $\mathbf{z} \in dom(f)$. We denote the set of all subgradients of f at $\mathbf{x}$ as $\partial f(\mathbf{x})$. Such a set is said to be the subdifferential of f at $\mathbf{x}$.

Notice that if f is differentiable at $\mathbf{x}$, we have $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$. This suggests that subgradients can be seen as generalized gradients for convex functions.

*See for example [28].

Example 2.1 Consider a nonempty, closed convex set X. Then the *indicator function* of X is defined as

$$\mathbb{1}_X(\mathbf{x}) = \begin{cases} 0 & \mathbf{x} \in X \\ +\infty & \mathbf{x} \notin X. \end{cases} \tag{2.8}$$

Notice that for all $\mathbf{z} \in \mathbb{R}^n$, the subdifferential of $\mathbb{1}_X$ at $\mathbf{x}$, i.e.,

$$\partial \mathbb{1}_X(\mathbf{x}) = \{\xi \mid \mathbb{1}_X(\mathbf{z}) \geq \mathbb{1}_X(\mathbf{x}) + \langle \xi, \mathbf{z} - \mathbf{x} \rangle\}$$

is equal to $\emptyset$ when $\mathbf{x} \notin X$.

On the other hand, if $\mathbf{x} \in X$, then we have

$$\partial \mathbb{1}_X(\mathbf{x}) = \{\xi \mid \mathbb{1}_X(\mathbf{z}) \geq \langle \xi, \mathbf{z} - \mathbf{x} \rangle\},$$

which is again equal to $\emptyset$, if $\mathbf{z} \notin X$. Consequently, if $\mathbf{z} \in X$, then

$$\partial \mathbb{1}_X(\mathbf{x}) = \{\xi \mid 0 \geq \langle \xi, \mathbf{z} - \mathbf{x} \rangle\}.$$

We call the subdifferential of $\mathbb{1}_X(\cdot)$ a *normal cone* to the set X, and we denote it by $\mathcal{N}_X(\cdot)$. Geometrically, this is the set of all normal vectors to a closed convex set at a given point [24].

Theorem 2.1 (Optimality condition) *The point $\mathbf{x}^*$ is the minimizer of the convex function $f\colon dom(f) \subset \mathbb{R}^n \to \mathbb{R}$ if and only if $0 \in \partial f(\mathbf{x}^*)$.*

Proof: By substituting $\xi = 0$ into (2.7) with $\mathbf{x} = \mathbf{x}^*$ we have

$$f(\mathbf{z}) \geq f(\mathbf{x}^*),$$

for all $\mathbf{z} \in dom(f)$, which shows that $\mathbf{x}^*$ is the global minimizer of f as it is stated in Definition 2.2. Similarly, if the inequality above holds, then $\xi = 0$ satisfies Definition 2.5. Hence, $0 \in \partial f(\mathbf{x}^*)$. □

Definition 2.6 *A convex function $f\colon dom(f) \subset \mathbb{R}^n \to \mathbb{R}$ is said to be lower semi-continuous at $\mathbf{x}$ if for all sequences $\{\mathbf{y}_k\}$ such that $\mathbf{y}_k \to \mathbf{x}$ we have*

$$\liminf_{k \to \infty} f(\mathbf{y}_k) \geq f(\mathbf{x}). \tag{2.9}$$

We are now able to define one of the key tools in optimization theory. Many optimization algorithms rely on the *proximal operator* as it allows us to handle the non-smoothness of functions.

Definition 2.7 *Let $f\colon \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a lower semi-continuous function. Moreover, assume it is closed and proper convex, i.e., its epigraph*

$$\mathbf{epi} f = \{(\mathbf{x}, t) \in \mathbb{R}^n \times \mathbb{R} \mid f(\mathbf{x}) \geq t\} \tag{2.10}$$

is a nonempty closed convex set. Then the proximal operator $\mathbf{prox}_f\colon \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\mathbf{prox}_f(\mathbf{v}) = \arg\min_{\mathbf{x}} \left(f(\mathbf{x}) + \frac{1}{2}\|\mathbf{x} - \mathbf{v}\|^2 \right). \tag{2.11}$$

Similarly, the proximal operator of f with parameter $\lambda > 0$ is defined as

$$\mathbf{prox}_{\lambda f}(\mathbf{v}) = \arg\min_{\mathbf{x}} \left(f(\mathbf{x}) + \frac{1}{2}\lambda\|\mathbf{x} - \mathbf{v}\|^2 \right). \tag{2.12}$$

Definition 2.8 *A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is said to be polyhedral if its epigraph (2.10) is a polyhedron, that is, a finite intersection of closed subspaces.*

Example 2.2 An important example of polyhedral functions, that we shall encounter later, are functions that can be expressed as a maximum of a finite number of affine functions. These are also called piecewise linear functions, written as

$$f(\mathbf{x}) = \max_{1 \leq i \leq k} \left(\mathbf{a}_i^\intercal \mathbf{x} + b_i \right), \tag{2.13}$$

where $\mathbf{a}_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$, $i = 1, \dots, k$. The epigraph of such functions can be written as

$$\mathbf{epi}f = \{(\mathbf{x}, t) \in \mathbb{R}^n \times \mathbb{R} \mid f(\mathbf{x}) \geq \max_{1 \leq i \leq k} (\mathbf{a}_i^\intercal \mathbf{x} + b_i)\},$$

and consequently,

$$\mathbf{epi}f = \{(\mathbf{x}, t) \in \mathbb{R}^n \times \mathbb{R} \mid f(\mathbf{x}) \geq \mathbf{a}_i^\intercal \mathbf{x} + b_i,\ i = 1, \dots, k\}$$

which shows that it is the polyhedron.

Let us now go back to our initial problem (2.1), without the constraint $\mathbf{x} \in X$. We have the following, fundamental result about the proximal operator.

Theorem 2.2 *The point $\mathbf{x}^*$ minimizes f if and only if*

$$\mathbf{x}^* = \mathbf{prox}_f(\mathbf{x}^*), \tag{2.14}$$

that is, the point $\mathbf{x}^*$ is a fixed point of $\mathbf{prox}_f$.

Proof: Assume that f has at least one subgradient at every point of its domain (even though the result is true in general, see [22]). If $\mathbf{x}^*$ minimizes f, then $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ implies that

$$f(\mathbf{x}) + \frac{1}{2}\|\mathbf{x} - \mathbf{x}^*\|^2 \geq f(\mathbf{x}^*) + \frac{1}{2}\|\mathbf{x}^* - \mathbf{x}^*\|^2,$$

for all $\mathbf{x}$. In other words, $\mathbf{x}^*$ minimizes $f(\mathbf{x}) + \frac{1}{2}\|\mathbf{x} - \mathbf{x}^*\|^2$ which is precisely the definition of the proximal operator, i.e. $\mathbf{x}^* = \mathbf{prox}_f(\mathbf{x}^*)$.

Conversely, if $\mathbf{prox}_f(\mathbf{v})$ has a minimizer $\mathbf{y}$, then according to the optimality condition (2.1) (see also [24]),

$$0 \in \partial f(\mathbf{y}) + (\mathbf{y} - \mathbf{v}).$$

By taking $\mathbf{y} = \mathbf{v} = \mathbf{x}^*$ we have $0 \in \partial f(\mathbf{x}^*)$, thus $\mathbf{x}^*$ minimizes f. □

The proofs of the following theorems can be found in [1].

Consider the constrained optimization problem,

$$\min_{\mathbf{x} \in X} f(\mathbf{x}), \tag{2.15}$$

where $f, h_1, \dots, h_M \colon \mathbb{R}^N \to \mathbb{R}$ are continuously differentiable functions and the set X is defined as

$$X = \{\mathbf{x} \in \mathbb{R}^N \mid h_m(\mathbf{x}) = 0,\ m = 1, \dots, M\}. \tag{2.16}$$

The constrained minimizer $\hat{\mathbf{x}}$ is called *regular* if $\{\nabla h_1(\hat{\mathbf{x}}), \dots, \nabla h_m(\hat{\mathbf{x}})\}$ is a linearly independent set. We introduce the following important result called *Lagrange Multiplier Theorem.*

Theorem 2.3 *Assume that $\hat{\mathbf{x}}$ is a regular minimizer of (2.15). There exists a unique $\hat{\lambda} \in \mathbb{R}^M$, called Lagrange multiplier vector, such that*

$$0 \in \partial f(\hat{\mathbf{x}}) + \sum_{1 \leq m \leq M} \hat{\lambda}_m \partial h_m(\hat{\mathbf{x}}). \tag{2.17}$$

We can generalize Lagrange multipliers if we also allow inequality constraints for the minimization problem (2.15). Thus, now we define the set X as

$$X = \{\mathbf{x} \in \mathbb{R}^N \mid h_m(\mathbf{x}) = 0,\ g_l(\mathbf{x}) \leq 0,\ \forall m, l\}, \tag{2.18}$$

where $g_1, \dots, g_L \colon \mathbb{R}^N \to \mathbb{R}$.

Denote

$$A(\hat{\mathbf{x}}) = \{l \mid g_l(\hat{\mathbf{x}}) = 0\}, \tag{2.19}$$

then we have the following theorem.

Theorem 2.4 (KKT conditions) *Assume that $\hat{\mathbf{x}}$ is a regular minimizer of (2.15) with the set X defined as in (2.18). Then there exists a unique Lagrange multiplier $\hat{\lambda} \in \mathbb{R}^M$ and a multiplier $\hat{\mu} \in \mathbb{R}^L_+$, called a KKT multiplier, such that*

$$0 \in \partial f(\hat{\mathbf{x}}) + \sum_{1 \leq m \leq M} \hat{\lambda}_m \partial h_m(\hat{\mathbf{x}}) + \sum_{1 \leq l \leq L} \hat{\mu}_l \partial g_l(\hat{\mathbf{x}}), \tag{2.20}$$

and $\hat{\mu}_l = 0$ if $l \notin A(\hat{\mathbf{x}})$. In addition, we have for all $m \in \{1, \dots, M\}$ and $l \in \{1, \dots, L\}$, the following:

$$h_m(\hat{\mathbf{x}}) = 0,\ g_l(\hat{\mathbf{x}}) \leq 0,\ \hat{\mu}_l \geq 0,\ \hat{\mu}_l g_l(\hat{\mathbf{x}}) = 0. \tag{2.21}$$

That is, primal feasibility, dual feasibility and complementarity.

2.2 Linear and quadratic programming problems

The reader can find more elaborately material on the following concepts in [18].

A LP, or *linear program*, is an optimization problem such that the objective function and the constraints are both made up of linear equations and/or inequalities. This means that the objective function is a linear combination of the variables being optimized, and the constraints are represented by linear equalities and inequalities involving those variables. Any LP problem can be written in the *standard form* as

$$\begin{cases} \text{minimize } \mathbf{c}^\intercal \mathbf{x} \\ \text{such that } \mathcal{A}\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}, \end{cases} \tag{2.22}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathcal{A}$ is a $m \times n$ matrix.

One popular way to solve LP problems is to use the simplex algorithm [6].

Similarly, a QP or *quadratic programming* problem is an optimization problem where the objective function is quadratic in the unknowns and the constraints consist of linear equalities and inequalities. Written in its standard form,

$$\begin{cases} \text{minimize } \frac{1}{2}\mathbf{x}^\intercal \mathcal{P} \mathbf{x} + \mathbf{q}^\intercal \mathbf{x} \\ \text{such that } \mathbf{l} \leq \mathcal{A}\mathbf{x} \leq \mathbf{u}, \end{cases} \tag{2.23}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{q} \in \mathbb{R}^n$, $\mathcal{P}$ is a symmetric, positive definite $n \times n$ matrix, $\mathbf{l}, \mathbf{u} \in \mathbb{R}^m$ and the $m \times n$ matrix $\mathcal{A}$ holds the coefficients for the inequality constraints.

There are many popular methods for solving QP problems. For instance, the conjugate gradient descent [12] or operator-splitting [27] can be used.

3 Bundle algorithms for Non-Smooth Optimization (NSO)

Throughout this section, the General Bundle Method (GBM), the Proximal Bundle Method (PBM), and two variants of the Level Bundle Method (LBM) are described. For a general overview of Bundle Methods we refer to [19], for an overview of Proximal and Level Bundle Methods we refer to [21] and for a mathematically rigorous resource on Bundle Methods we refer to [2].

Remark 1 For simplicity, we will write $\mathbf{x}$ as x from now on. By $\|\cdot\|$ we mean the standard Euclidean 2-norm $\|\cdot\|_2$. We will emphasize that if it is not clear from the context.

Let us now formulate our main problem more precisely. We have the following non-smooth optimization (NSO) problem

$$\begin{cases} \min f(x) \\ \text{such that } x \in X. \end{cases} \tag{3.1}$$

We call the nonempty closed convex set $X \subset \mathbb{R}^n$ the *feasible set*. The objective function f is allowed to be non-differentiable, however, we require it to be (locally) Lipschitz continuous on the feasible set X. Suppose that at any $x \in \mathbb{R}^n$ we can evaluate the functional value $f(x)$ and at least one subgradient $\xi \in \partial f(x)$[†].

[†]This is in optimization known as the black-box method. See [2].

3.1 General Bundle Method (GBM)

One of the main issues is that, in general, we might now know the whole subdifferential ∂f of our objective function f. We wish to approximate the subdifferential ∂f using a *bundle* consisting of some information from previous iterations. The computed subgradients are used to approximate f with piecewise-affine functions. This approximation is used to determine the descent direction. If the descent is not sufficient, we add more information to the bundle.

In particular, we want to construct a sequence of *stability centres* $\{x_k\}_{k=1}^{\infty}$ converging to the global minimizer of f. In addition to the points x_k, we also have the support sequence or the *candidate points* $y_j \in \mathbb{R}^n$ and subgradients $\xi_j \in \partial f(y_j)$, $j \in J_k \subset \{1, \dots, k\}$. At each iteration k and the iteration point x_k, we define the bundle as the set

$$\mathcal{B}_k = \{(y_j, f(y_j), \xi_j) \mid j \in J_k\}. \tag{3.2}$$

So $\mathcal{B}_k$ carries the information based on the indices from the index set $J_k \subset \{1, \dots, k\}$. We wish to control the size of the index set J_k and hence control the storage in the bundle $\mathcal{B}_k$. This is one of the most crucial properties of Bundle Methods.

Bundle methods are an improved version of the *Cutting-Planes Method* (see [11]). The main idea is to build a polyhedral model of f, i.e., approximate f from below with piecewise-affine functions

$$\hat{f}_k(x) = \max_{j \in J_k}\{f(y_j) + \langle \xi_j, x - y_j \rangle\}, \tag{3.3}$$

for all $x \in \mathbb{R}^n$. As f is assumed to be convex and $\xi_j \in \partial f(y_j)$, we indeed have from (2.5) that for every $x \in \mathbb{R}^n$,

$$\hat{f}_k(x) = \max_{j \in J_k}\{f(y_j) + \langle \xi_j, x - y_j \rangle\} \leq f(x), \tag{3.4}$$

which shows that the polyhedral model bounds f from below.

Denote the *linearization error* by

$$\alpha_j^k := f(x_k) - f(y_j) - \langle \xi_j, x_k - y_j \rangle \geq 0, \tag{3.5}$$

for every $j \in J_k$. Hence, we can also write the polyhedral model as

$$\hat{f}_k(x) = \max_{j \in J_k}\{f(x_k) + \langle \xi_j, x - x_k \rangle - \alpha_j^k\} \leq f(x). \tag{3.6}$$

We define the next iterate as

$$y_{k+1} := x_k + d_k, \tag{3.7}$$

where the *descent direction* d_k is the solution of

$$\arg\min_{d \in X}\{\hat{f}_k(x_k + d) + \frac{1}{2} d^{\intercal} \mathbf{M}_k d\}, \tag{3.8}$$

$\mathbf{M}_k$ is a symmetric $n \times n$ matrix with $\det(\mathbf{M}_k) \neq 0$ for every k. The role of the matrix $\mathbf{M}_k$ is to accumulate the information about the curvature of f around x_k. The term $\frac{1}{2} d^{\intercal} \mathbf{M}_k d$ acts as a stabilizer, i.e., it makes the approximation local enough by preventing big oscillations and it also guarantees the existence of the solution d_k [19].

Remark 2 In order to simplify the notation in the algorithmic schemes, we often try to express the updating rule for the next iterate with as few letters as possible, often omitting the direction d_k as we shall see in later parts. Likewise, the matrix $\mathbf{M}_k$ is often taken to be just the identity matrix I and we often equip the *stabilizing subproblem* (3.8) with the Euclidean 2-norm $\|\cdot\|$.

We define the *predicted descent* as

$$v_k = \hat{f}_k(y_{k+1}) - f(x_k) \tag{3.9}$$

and pick a line search parameter $\kappa \in (0, \frac{1}{2})$. We stop the algorithm when

$$v_k \geq -\epsilon, \tag{3.10}$$

for some $\epsilon > 0$ which presents the desired tolerance set at the initialization of the algorithm.

The next iterations are computed as follows: we perform a *serious step*,

$$x_{k+1} \coloneqq y_{k+1} \tag{3.11}$$

if

$$f(y_{k+1}) - f(x_k) \leq \kappa v_k, \tag{3.12}$$

that means if the found descent is adequate enough. Otherwise we make a *null step*,

$$x_{k+1} \coloneqq x_k. \tag{3.13}$$

This does not mean that by performing the null step we make no progress, however. In both cases, we improve the cutting plane model of f by adding $\{(y_{k+1}, f(y_{k+1}), \xi_{k+1})\}$ to the bundle $\mathcal{B}$.

Notice that one natural way how to update the index set J_k is to set $J_{k+1} = J_k \cup \{k+1\}$ at each iteration. This strategy, however, is very inefficient in terms of memory storage and computational speed because the size of the index set J_k is unbounded in this case. Therefore, we typically employ more refined updating schemes for J_k (and thus also for the bundle $\mathcal{B}_k$), called *aggregation mechanisms*, one of which we will introduce in the next section.

Lastly, notice that in order to find the descent direction d_k for solving the stabilizing subproblem (3.8), we need to solve the following (smooth and convex) QP problem,

$$\begin{cases} \min v + \frac{1}{2} d^{\intercal} \mathbf{M}_k d \\ \text{such that } \langle \xi_j, d \rangle - \alpha_j^k \leq \tau, \ \forall j \in J_k, \end{cases} \tag{3.14}$$

where α_j^k is the linearization error and $\tau = \max_{j \in J_k} \{f(x_k) + \langle \xi_j, x - x_k \rangle - \alpha_j^k\}$. This is the cost we have to pay in order to achieve the stability of our algorithm.

Algorithm GBM

```
Step 0: Pick $x_1, y_1 \in \mathbb{R}^n$, $\epsilon > 0$, $J_1 = \{1\}$, $k = 1$, $\mathbf{M}_1 \in \mathbb{R}^{n \times n}$ with $\det(M_1) \neq 0$. Set $v_k = -\infty$.
Step 1: Stop when $v_k \geq -\epsilon$.
Step 2: Find $\xi_k \in \partial f(y_k)$.
Step 3: Solve (3.14) and set $y_{k+1} = x_k + d_k$.
Step 4: Pick $\kappa \in (0, \frac{1}{2})$. Set $v_k = \hat{f}_k(y_{k+1}) - f(x_k)$.
Step 5: If $f(y_{k+1}) - f(x_k) \leq \kappa v_k$, make a serious step $x_{k+1} = y_{k+1}$. Otherwise, make a null step $x_{k+1} = x_k$.
Step 6: Upgrade $\mathbf{M}_k = \widetilde{\mathbf{M}}_k$. Bundle aggregation: $J_k = \hat{J}_k$ and pick $J_{k+1} \supset J_k \cup \{k+1\}$.
Step 7: Increment $k$ to $k+1$ and go to Step 1.
```

3.2 Proximal Bundle Method (PBM)

In this section we present the *Proximal Bundle Method* (PBM) as described by Kiwiel in [13] and later improved by Kiwiel et al. in [15] and [9].

Consider the problem (3.1). We may treat this problem as the unconstrained problem $f^* \coloneqq \min f_X$ using the indicator function (2.1), i.e.,

$$f_X \coloneqq f(x) + \mathbb{1}_X. \tag{3.15}$$

To keep the notation simple, we may write $f \coloneqq f_X$. We again assume that it is possible to evaluate $f(x)$, $\xi \in \partial f(x)$ at any $x \in X$. Suppose that f has a global minimum.

PBM produces a sequence $\{x_k\}_{k=1}^{\infty} \subset X$ and points $y_j \in \mathbb{X}$ at each iteration. Define also $\xi_j \in \partial f(y_j)$, $j \in J_k \subset \{1, \dots, k\}$. PBM works with the linearizations of f,

$$f_k(\cdot) \coloneqq f(y_k) + \langle \xi_k, \cdot - y_k \rangle \leq f(\cdot) \tag{3.16}$$

similarly as in (3.4).

We also consider the polyhedral model of f at the iteration k

$$\hat{f}_k(x) := \max_{j \in J_k} f_j, \tag{3.17}$$

for $j \in J_k \subset \{1, \dots, k\}$.

The algorithm starts with picking $x_1 = y_1 \in X$. At each iteration, we compute the next candidate point y_{k+1} by solving for the proximal operator with the *proximity parameter* $u_k > 0$,

$$y_{k+1} = \mathbf{prox}_{u_k \hat{f}_k}(x_k) = \arg\min_x \left(\hat{f}_k(x) + \frac{1}{2} u_k \|x - x_k\|^2 + \mathbb{1}_X \right). \tag{3.18}$$

Next, the predicted descent is defined as

$$v_k := \hat{f}_k(y_{k+1}) - f(x_k). \tag{3.19}$$

The algorithm terminates if $v_k \geq -\epsilon$ for some given tolerance $\epsilon > 0$.

The serious step $x_{k+1} := y_{k+1}$ is made when

$$f(y_{k+1}) - f(x_k) \leq \kappa v_k, \tag{3.20}$$

where $\kappa \in (0, 1)$ is a fixed search parameter. Otherwise, we make the null step $x_{k+1} := x_k$.

After solving (3.18), we update the proximity parameter u_k. At each $k \geq 1$, we pick

$$u_k \in [u_{min}, u_{max}], \text{ for fixed } 0 < u_{min} \leq u_{max} < +\infty. \tag{3.21}$$

If the null step is made, we update with $u_{k+1} \geq u_k$.

We use the following aggregation technique to update the index set J_{k+1}. The optimality condition for (3.18) is

$$0 \in \partial \hat{f}_k(y_{k+1}) + u_k \|y_{k+1} - x_k\| + \partial \mathbb{1}_X(y_{k+1}). \tag{3.22}$$

Thus, there exists

$$p_f^k \in \partial \hat{f}_k(y_{k+1}), \tag{3.23}$$

such that

$$p_X^k := -u_k (y_{k+1} - x_k) - p_f^k \in \mathcal{N}_X(y_{k+1}) := \partial \mathbb{1}_X(y_{k+1}). \tag{3.24}$$

By (3.23, 3.16, 3.17) there exist Lagrange (KKT) multipliers $\lambda_j^k,\ j \in J_k$ such that

$$p_f^k = \sum_{j \in J_k} \lambda_j^k \xi_j, \text{ with } \sum_{j \in J_k} \lambda_j^k = 1 \text{ and } \lambda_j^k [\hat{f}_k(y_{k+1}) - f_j(y_{k+1})] = 0,\ j \in J_k. \tag{3.25}$$

Consider the active set of indices

$$\hat{J}_k := \{j \in J_k \mid \lambda_j^k > 0\}. \tag{3.26}$$

We pick J_k^s such that $\hat{J}_k \subset J_k^s \subset J_k$ and update the index set with $J_{k+1} = J_k^s \cup \{k+1\}$. In other words, we drop the linearizations f_j for which $\lambda_j^k = 0$. Indeed, it follows from the KKT conditions that such multipliers have no effect on p_f^k in (3.25) and thus they do not affect y_{k+1}. Therefore, this mechanism helps us to compress the size of J_k without affecting the convergence of the algorithm [9].

We can set $M_\xi \geq n + 2$ to be the maximum allowed size of the index set J_k and incorporate it in the algorithm as a termination condition (sentinel value). There exist advanced versions of PBM where the index set J_k can be compressed down to just 2 elements ([9], [25]). However, we work with the simplified version of PBM and M_ξ is taken to be a larger number during the implementation.

Lastly, notice that solving for the proximal operator means solving a QP problem. Namely, y_{k+1} and $\tau_k := \hat{f}_k(y_{k+1})$ are the solutions of

$$\begin{cases} \min \frac{1}{2}u_k\|x - x_k\|^2 + \tau, \text{ over all } (x,\tau) \in X \times \mathbb{R} \\ \text{such that } f_j(x_k) + \langle \xi_j, x - x_k\rangle \leq \tau, \ \forall j \in J_k. \end{cases} \tag{3.27}$$

Algorithm PBM

```
Step 0: Pick $x_1 = y_1 \in X$, $\epsilon > 0$, $J_1 = \{1\}$, $k = 1$, $M_\xi \geq n+2$, $\kappa \in (0,1)$, $u_1 > 0$, $u_{max} \geq u_{min} > 0$.
        Set $v_k = -\infty$.
Step 1: Stop when $v_k \geq -\epsilon$.
Step 2: Find $\xi_k \in \partial f(y_k)$.
Step 3: Solve (3.27) and find the multipliers $\lambda_j^k$ (3.25).
Step 4: Set $v_k = \hat{f}_k(y_{k+1}) - f(x_k)$.
Step 5: If $f(y_{k+1}) - f(x_k) \leq m v_k$, make a serious step $x_{k+1} = y_{k+1}$. Otherwise, make a null step
        $x_{k+1} = x_k$.
Step 6: Bundle aggregation: take the set of active indices $\hat{J}_k$ and pick $J_k^s$ such that $\hat{J}_k \subset J_k^s \subset J_k$ and
        $|J_k^s| \leq M_\xi - 1$ and set $J_{k+1} = J_k^s \cup \{k+1\}$.
Step 7: Upgrade $u_{k+1} \in [u_{min}, u_{max}]$.
Step 8: Increment $k$ to $k+1$ and go to Step 1.
```

3.3 Level Bundle Method (LBM)

In this section, we describe the *Level Bundle Method* (LBM). More specifically, we present two types of LBM - the Proximal and Non-proximal variant[‡]. The original LBM was introduced by Lemaréchal, Nemirovskii, and Nesterov in [17] and we refer to it as the Non-Proximal LBM. This method was later improved by many mathematicians, in particular, the core of this section is built on the work by Oliveira and Sagastizábal in [7], which generalizes the original LBM by introducing the Proximal LBM.

The initial problem is similar as in the previous sections. We seek to minimize a Lipschitz continuous convex function f over a non-empty, compact convex set $X \subset \mathbb{R}^n$. Notice that we require the feasible set X to be bounded in addition to the closedness. Suppose that we can evaluate $f(x)$ and at least one subgradient $\xi \in \partial f(x)$ at any $x \in X$.

LBM resembles PBM, but there are a few differences. Instead of solving for the proximal operator, we are solving a *projection problem*[§] which is also a QP problem. Let us remind that in PBM we compute the next iterate (serious step) as

$$x_{k+1} = \arg\min\{\hat{f}_k(x) + \frac{1}{2}u_k\|x - x_k\|^2 \mid x \in X\}. \tag{3.28}$$

The QP problem in LBM is slightly different,

$$x_{k+1} = \arg\min\{\frac{1}{2}\|x - \hat{x}_k\|^2 \mid \hat{f}_k(x) \leq f_k^{lev}, \, x \in X\}. \tag{3.29}$$

LBM generates the sequence $\{x_k\}_{k=1}^\infty \subset X$ converging to the global minimizer of f, if it exists. Unlike the PBM, the LBM does not require any support sequence $\{y_j\}_{j=1}^\infty$.

We again use the linearizations

$$f(x_k) + \langle \xi_k, \cdot - x_k\rangle \leq f(\cdot) \tag{3.30}$$

and the polyhedral model of f

$$\hat{f}_k(\cdot) := \max_{j \in J_k}\{f(x_j) + \langle \xi_j, \cdot - x_j\rangle\}, \text{ where } J_k \subset \{1, \ldots, k\}. \tag{3.31}$$

By solving the LP problem

$$f_k^{low} := \min_{x \in X} \hat{f}_k(x), \tag{3.32}$$

[‡]The word "proximal" is used here due to the historical context and conceptual similarities with the proximal operator and the PBM.

[§]Proximal operator is actually a generalized projection, see [22] for more details.

we obtain a lower bound for the optimal value $f^* := \inf_{x\in X} f$. In other words, f_k^{low} is the lowest level value and the rest of the polyhedral model lies above it.

Similarly, we denote

$$f_k^{up} := f(x_k) \tag{3.33}$$

to be the upper bound for the optimal value f^*. Indeed, as $x_k \in X$ and $f^* := \inf_{x\in X} f$, we have $f(x_k) \geq f^*$. Thus, f_k^{up} can be seen as the upper level value.

We define the *optimality gap* $\Delta_k := f_k^{up} - f_k^{low}$. Next, we define

$$f_k^{lev} := f_k^{up} - (1-\kappa)\Delta_k = f_k^{low} + \kappa\Delta_k, \tag{3.34}$$

with a level parameter $\kappa \in (0, 1)$. Finally, we define the *level set* as

$$\mathbb{X}_k := \{x \in X \mid \hat{f}_k(x) \leq f_k^{lev}\}. \tag{3.35}$$

One of the advantages of the Proximal LBM is that it suffices to solve the LP problem (3.32) only once, at the initialization of the algorithm. On the other hand, when using the Non-Proximal LBM, we need to solve (3.32) at each iteration, which adds extra complexity to the algorithm.

We start by picking the center parameter $\Theta \in \{0, 1\}$, where $\Theta = 0$ denotes the Non-Proximal LBM and $\Theta = 1$ denotes the Proximal LBM. Given the tolerance $\epsilon > 0$, we stop the algorithm when $\Delta_k \leq \epsilon$.

Another difference between the Proximal and Non-Proximal variants of LBM is in the meaning of the *projection points* $\hat{x}_k$. The points $\hat{x}_k$ can be seen as stability centers (Proximal LBM) or just the last iterates (Non-Proximal LBM).

The notion of serious and null steps is also slightly different for this class of Bundle Methods. Starting from $\hat{x}_0 = x_1$, if the optimality gap Δ_k is "thin" enough,

$$\Delta_k \leq (1-\kappa)\,\Delta_{k(l)}, \text{ where } k(l) \in \mathbb{Z}, \tag{3.36}$$

we select $\hat{x}_k \in \{x_j \mid j \in J_k\}$ ("serious step"). Otherwise ("null step"), we set

$$\hat{x}_k = \Theta\hat{x}_{k-1} + (1-\Theta)x_k. \tag{3.37}$$

In both variants, the next iterate is computed by solving the projection problem,

$$x_{k+1} := \arg\min_{x\in\mathbb{X}_k} \frac{1}{2}\|x - \hat{x}_k\|^2, \tag{3.38}$$

where $\mathbb{X}_k$ is the level set as defined in (3.35). In other words, x_{k+1} is obtained by projecting $\hat{x}_k$ onto the level set $\mathbb{X}_k$. It might happen that the problem (3.38) is infeasible because $\mathbb{X}_k$ might be empty. In that case, before solving for x_{k+1}, we pick a new $\tilde{f}_k^{low}$ such that $\tilde{f}_k^{low} \in (f_k^{low}, f_k^{lev}]$ and we start the algorithm again with a new optimality gap $\tilde{\Delta}$.

For both the Non-Proximal and Proximal variants of LBM, we update the upper bound f_{k+1}^{up} as

$$f_{k+1}^{up} := \min\{f_k^{up}, f(x_{k+1})\}. \tag{3.39}$$

As it was noted before, the updating rule for the lower level bound differs based on the value of the center parameter Θ. If $\Theta = 1$, i.e., the Proximal LBM, we solve (3.32) only once, namely for $k = 1$ and then set

$$f_{k+1}^{low} := f_k^{low}, \text{ for } k > 1. \tag{3.40}$$

On the other hand, if $\Theta = 0$, that is, the Non-Proximal LBM, we need to solve (3.32) at each iteration k.

Lastly, we discuss the aggregation mechanism. When $\Theta = 1$, we can use the same approach as we used for PBM. Namely, we drop the inactive indices j from the index set J_k. When $\Theta = 0$, we set $J_{k+1} = J_k \cup \{k+1\}$ after each iteration. While some aggregation strategies can be applied here as well

([7], [15]), to demonstrate the consequences of the absence of storage mechanism on the CPU speed during implementation, we decided to allow $|J_k|$ to be unbounded[¶].

For the sake of completeness, we present the complete formulation of the QP problem for LBM,

$$\begin{cases} \min \frac{1}{2}\|x-\hat{x}_k\|^2, \text{ over all } x \in X \\ \text{such that } f_j(x_k) + \langle \xi_j, x - x_k\rangle - f_k^{lev} \leq 0, \ \forall j \in J_k. \end{cases} \tag{3.41}$$

.

Algorithm LBM

```
Step 0: Pick $\hat{x}_0 = x_1 \in X$, $\epsilon > 0$, $k = 1$, $l = 0$, $k(l) = 1$, $J_1 = \{1\}$, $k = 1$, $\kappa \in (0,1)$, $\Theta \in \{0,1\}$. Find $\xi_1 \in \partial f(x_1)$. Solve (3.32) for $f_1^{low}$. Set $f_1^{up} = f(x_1)$.
Step 1: Stop when $\Delta_k = f_k^{up} - f_k^{low} \leq \epsilon$.
Step 2: If $\Delta_k \leq (1-\kappa)\Delta_{k(l)}$, make a serious step $\hat{x}_k \in \{x_j \mid j \in J_k\}$. Set $l = l+1$, $k(l) = k$. Else, make a null step $\hat{x}_k = \Theta\hat{x}_{k-1} + (1-\Theta)x_k$.
Step 3: Compute $\mathbb{X}_k$. If $\mathbb{X}_k = \emptyset$, adjust $f_k^{low} = \tilde{f}_k^{low}$ and go back to Step 2.[‖]
Step 4: Solve (3.41). Compute $\xi_{k+1} \in \partial f(x_{k+1})$. Find the multipliers $\lambda_j^k$.
Step 5: Set $f_{k+1}^{up} = \min\{f_k^{up}, f(x_{k+1})\}$. If $\Theta = 1$, set $f_{k+1}^{low} = f_{low}^k$. If $\Theta = 0$, solve (3.32) for $f_{LP}$ and set $f_{k+1}^{low} = \max\{f_{LP}, f_{low}^k\}$.
Step 6: Bundle aggregation. If $\Theta = 1$, take the set of active indices $\hat{J}_k$ and pick $J_k^s$ such that $\hat{J}_k \subset J_k^s \subset J_k$ and $|J_k^s| \leq M_\xi - 1$ and set $J_{k+1} = J_k^s \cup \{k+1\}$. If $\Theta = 0$, set $J_{k+1} = J_k \cup \{k+1\}$.
Step 7: Increment $k$ to $k+1$ and go to Step 1.
```

4 Convergence analysis

This section provides a comprehensive convergence analysis of the PBM and the Proximal LBM. Section 4.1 offers complete proof of convergence of the PBM. Similarly, the full proof of convergence of the Proximal LBM can be found in Section 4.2.

For further insights into the Non-Proximal LBM, we refer to [17]. Additionally, [2] serves as a valuable resource for a more in-depth analysis of the General Bundle Method (GBM) and its various variants.

4.1 PBM

This section is mainly based on [8, 13, 14]. These references have provided valuable insights, theoretical foundations, and methodologies that form the basis of our analysis and investigation in this section.

Recall the optimality condition for (3.18),

$$0 \in \partial\hat{f}_k(y_{k+1}) + u_k\|y_{k+1} - x_k\| + \partial\mathbb{1}_X(y_{k+1}). \tag{4.1}$$

We can pick subgradients $p_f^k \in \partial\hat{f}(y_{k+1})$, and $p_X^k \in \partial\mathbb{1}_X(y_{k+1})$ to set the aggregate linearizations

$$\bar{f}_k(\cdot) = \hat{f}_k(y_{k+1}) + \langle p_f^k, \cdot - y_{k+1}\rangle, \ \bar{\mathbb{1}}_X^k(\cdot) = \langle p_X^k, \cdot - y_{k+1}\rangle. \tag{4.2}$$

Denoting the descent direction by $d_k = y_{k+1} - x_k$, we may write

$$p_f^k + p_X^k + u_k\|d_k\| = 0. \tag{4.3}$$

Let us denote

$$p_k := p_f^k + p_X^k = -u_k d_k, \ \bar{\alpha}_{agg}^k := f(x_k) - \bar{f}_k(x_k) \geq 0, \ \bar{\alpha}_X^k := -\bar{\mathbb{1}}_X^k(x_k) = \langle p_X^k, d_k\rangle \geq 0. \tag{4.4}$$

The last two inequalities come from the fact that the linearization of a convex function bounds the function from below.

Lemma 1 *Denote $\bar{\alpha}_p^k = \bar{\alpha}_{agg}^k - \bar{\alpha}_X^k$. Then we have*

$$-v_k = u_k\|d_k\|^2 + \bar{\alpha}_p^k = \frac{1}{u_k}\|p_k\|^2 + \bar{\alpha}_p^k. \tag{4.5}$$

[¶]In fact, in the original paper [17] the authors do not present any aggregation mechanism.

[‖]By setting $f_k^{low} = \tilde{f}_k^{low}$ we get a new $\tilde{f}_k^{lev}$ and in particular a new optimality gap $\tilde{\Delta}$.

Proof: Using (4.3, 4.4, 3.19) and the linearity of the inner product $\langle \cdot, \cdot \rangle$, we have

$$\begin{aligned}
-v_k &= -\hat{f}_k(y_{k+1}) + f(x_k) = -\hat{f}_k(y_{k+1}) + f(x_k) + \langle p_f^k + p_X^k + u_k d_k, d_k \rangle \\
&= -\hat{f}_k(y_{k+1}) + f(x_k) + u_k \|d_k\|^2 + \langle p_f^k + p_X^k, d_k \rangle \\
&= -\hat{f}_k(y_{k+1}) + f(x_k) + u_k \|d_k\|^2 - \langle p_f^k, x_k - y_{k+1} \rangle + \langle p_X^k, d_k \rangle \\
&= u_k \|d_k\|^2 + f(x_k) - \bar{f}_k(x_k) + \langle p_X^k, d_k \rangle \\
&= u_k \|d_k\|^2 + \bar{\alpha}_p^k \\
&= \frac{1}{u_k} \|p_k\|^2 + \bar{\alpha}_p^k.
\end{aligned}$$

Which concludes the proof. □

We have

$$f(x) \geq f(x_k) + \langle p_k, x - x_k \rangle - \bar{\alpha}_p^k, \tag{4.6}$$

if we consider (3.6) with the aggregate linearization. Lemma 1 implies that

$$v_k \leq -\bar{\alpha}_p^k, \text{ and } \|p_k\| \leq \| - u_k v_k \|^{\frac{1}{2}}, \tag{4.7}$$

so

$$f(x) \geq f(x_k) - \langle (u_k v_k)^{\frac{1}{2}}, x - x_k \rangle + v_k \geq f(x_k) - \|u_k v_k\|^{\frac{1}{2}} \|x - x_k\| + v_k, \tag{4.8}$$

what follows from applying Cauchy-Schwarz inequality.

We may now start with proving that the sequence $\{x_k\}$ generated by PBM converges to some global minimizer of f, i.e.,

$$x_k \to \bar{x} \in \Omega = \arg\min_X f, \tag{4.9}$$

assuming the set of global minimizers Ω is nonempty. Hence, take $\epsilon = 0$. Notice that (4.8) holds for all $x \in X$, therefore we have that $x_k \in \Omega$ after termination, because in that case (4.8) reduces to $f(x) \geq f(x_k)$. Therefore, in order to prove (4.9), it is sufficient to assume that the PBM algorithm does not terminate.

Denote $f^\infty := \lim_{k \to \infty} f(x_k)$ and suppose that for some $\tilde{x} \in X$ and for all $k \in \mathbb{N}$ we have

$$f(x_k) \geq f(\tilde{x}). \tag{4.10}$$

Since f is assumed to be Lipschitz continuous on the closed set X, and $\{x_k\} \subset X$, the assumption above means that $\tilde{x}$ is a limit point of the sequence $\{x_k\}$. In what follows, it will be useful to introduce some new notation when talking about serious/null steps, thus we present a few remarks.

Remark 3 If we make a serious step at iteration k, set $t_L^k = 1$ and similarly, $t_L^k = 0$ if we make a null step. Notice that then we may simply write

$$x_{k+1} = t_L^k y_{k+1} + (1 - t_L^k) x_k. \tag{4.11}$$

Remark 4 Let $l \in \mathbb{N} \cup \{0\}$ be the counter of serious steps and let $k(l)$ be a function $k(\cdot)\colon \mathbb{N} \cup \{0\} \to \mathbb{N}$ such that when a serious step is made, we increment l to $l + 1$ and set

$$k(l + 1) = k + 1, \tag{4.12}$$

where k is the iteration counter. We initialize as $l = 0$, $k(0) = 1$, $k = 1$. So $k(l)$ will denote the iteration number of the lth serious step.

Lemma 2 *If $f^\infty > -\infty$, then*

$$\sum_{k=1}^{\infty} t_L^k (-\kappa v_k) \leq \sum_{k=1}^{\infty} t_L^k |v_k| \leq +\infty. \tag{4.13}$$

Moreover, if $K = \{k \mid t_L^k = 1\}$ is infinite then $v_k \xrightarrow{K} 0$.

Proof: We have $v_k = \hat{f}_k(y_{k+1}) - f(x_k) \le 0$. As $\kappa \in (0,1)$, we have

$$0 \le -\kappa v_k \le -v_k \le f(x_k) - f(x_{k+1}).$$

Summing over $k = 1, 2, \dots$, we have

$$\sum_{k=1}^{\infty} t_L^k(-\kappa v_k) \le \sum_{k=1}^{\infty} t_L^k |v_k| \le \sum_{k=1}^{\infty} (f(x_k) - f(x_{k+1})) \le +\infty,$$

as the last sum is telescopic and $f^\infty > -\infty$. Consequently, we must have $v_k \overset{K}{\to} 0$ if infinite number of serious steps were made. □

Lemma 3 *If (4.10) holds, then $x_k \to \bar{x}$, for some $\bar{x} \in X$.*

Proof: Set $x = \tilde{x}$. Then (4.10) and (4.6) implies that $\langle p_k, \tilde{x} - x_k \rangle \le \bar{\alpha}_p^k$. Next, we have

$$x_{k+1} - x_k = t_L^k d_k = -\frac{1}{u_k} p_k t_L^k \tag{4.14}$$

since $d_k = -\frac{1}{u_k} p_k$ and t_L^k is either 0 or 1. Then,

$$\begin{aligned} \|\tilde{x} - x_k\|^2 = \|\tilde{x} - x_k + x_k - x_{k+1}\|^2 &= \|\tilde{x} - x_k\|^2 + 2\langle \tilde{x} - x_k, x_k - x_{k+1} \rangle + \|x_k - x_{k+1}\|^2 \\ &\le \|\tilde{x} - x_k\|^2 + 2t_L^k \bar{\alpha}_p^k \frac{1}{u_k} + t_L^k \left(\frac{\|p_k\|}{u_k} \right)^2. \end{aligned} \tag{4.15}$$

By the Lemmas above and by taking $u_k \ge u_{min}$,

$$\|\tilde{x} - x_n\|^2 \le \|\tilde{x} - x_k\|^2 + \sum_{i=k}^{\infty} 2t_L^i \frac{|v_i|}{u_{min}} < +\infty, \text{ if } n > k, \tag{4.16}$$

which shows that $\{x_k\} \subset X$ is bounded. As X is closed, there exists some $\bar{x} \in X$ such that $x_k \to \bar{x}$. Take $\bar{x} = \tilde{x}$, then by taking $\epsilon > 0$ there exists some index k for which,

$$\|\bar{x} - x_k\|^2 \le \frac{\epsilon}{2}, \text{ and } \sum_{i=k}^{\infty} \frac{|v_i|}{u_{min}} \le \frac{\epsilon}{2}.$$

Hence, $\|\bar{x} - x_k\|^2 \le \epsilon$ whenever $n > k$. We conclude that $x_k \to \bar{x}$. □

Lemma 4 *Let*

$$w_k := \frac{u_k \|d_k\|^2}{2} + \bar{\alpha}_p^k = \frac{1}{2u_k} \|p_k\|^2 + \bar{\alpha}_p^k. \tag{4.17}$$

Then $v_k \le -w_k \le \frac{v_k}{2}$. Moreover, if $x_{k+1} = x_k$ and $u_{k+1} \ge u_k$, then

$$0 \le w_{k+1} \le w_k - \frac{u_k \|d_{k+1} - d_k\|^2}{2}. \tag{4.18}$$

Proof: The first claim follows from Lemma 1. To prove the second claim, consider $\hat{J}_k$ as described in (3.26) and let

$$f_k^{\hat{J}}(x) = \max_{j \in \hat{J}_k} f_j \tag{4.19}$$

with f_j defined as in (3.16). Denote

$$\Psi_1^k = \hat{f}_k + \frac{u_k \| \cdot - x_k\|^2}{2} + \mathbb{1}_X(\cdot), \text{ and } \Psi_2^k = f_k^{\hat{J}} + \frac{u_k \| \cdot - x_k\|^2}{2} + \mathbb{1}_X(\cdot).$$

Suppose that y_{k+1} is the solution of the proximal proximal operator $\mathbf{prox}_{u_k \hat{f}_k}(x_k)$. Denote

$$\rho_k = \min \Psi_1^k = \hat{f}_k(y_{k+1}) + \frac{u_k \|y_{k+1} - x_k\|^2}{2}. \tag{4.20}$$

Notice that we also have

$$y_{k+1} = \mathbf{prox}_{u_k f_k^j}(x_k) = \arg\min \Psi_2^k,$$

thus, $f_k^{\hat{J}}(y_{k+1}) = \hat{f}_k(y_{k+1})$ and consequently, $\rho_k = \min \Psi_2^k$.

Each f_j in (4.19) is strongly convex, thus also $f_k^{\hat{J}}(x)$ is strongly convex. The indicator function over the convex set X is a convex function and the term $\frac{u_k\|y_{k+1}-x_k\|^2}{2}$ is strongly convex. Therefore, Ψ_2^k is also strongly convex with modulus of convexity $u_k > 0$**. This implies that

$$\Psi_2^k \geq \rho_k + \frac{u_k\|y_{k+1} - x_k\|^2}{2}. \tag{4.21}$$

If $x_{k+1} = x_k$, then $\hat{f}_{k+1} \geq f_k^{\hat{J}}$ (as $J_{k+1} \supseteq \hat{J}_k$) and $u_{k+1} \geq u_k$ (as $x_{k+1} = x_k$ means "null step"). Hence,

$$\Psi_1^{k+1} \geq \Psi_2^k, \text{ and } \rho_{k+1} \geq \rho_k + \frac{u_k\|y_{k+2} - y_{k+1}\|^2}{2}. \tag{4.22}$$

Notice that $y_{k+2} - y_{k+1} = d_{k+1} - d_k$ and

$$f(x_k) - \rho_k = f(x_k) - \hat{f}_k(y_{k+1}) - \frac{u_k\|d_k\|^2}{2} = -v_k - \frac{u_k\|d_k\|^2}{2} = \frac{u_k\|d_k\|^2}{2} + \bar{\alpha}_p^k = w_k \tag{4.23}$$

by Lemma 1, (4.20) and the definition of w_k. Hence,

$$-w_k = v_k + \frac{u_k\|d_k\|^2}{2} \tag{4.24}$$

this shows that

$$0 \leq w_{k+1} \leq w_k - \frac{u_k\|d_{k+1} - d_k\|^2}{2}. \tag{4.25}$$

□

Lemma 5 *If $u_{k+1} \geq u_k$, then $w_{k+1} \leq -\frac{3v_k}{2}$.*

Proof: If a null step is made, i.e., $x_{k+1} = x_k$, then $w_{k+1} \leq w_k \leq -v_k$ by Lemma 4. If a serious step is made, i.e., $x_{k+1} = y_{k+1}$, then

$$\hat{f}_k(y_{k+1}) + \langle p_k, \cdot - y_{k+1}\rangle = \bar{f}_k(\cdot) + \bar{\mathbb{1}}_X^k \leq \hat{f}_k(\cdot) + \mathbb{1}_X(\cdot) \tag{4.26}$$

and we also have

$$\hat{f}_{k+1}(\cdot) \geq f_k^{\hat{J}}(\cdot). \tag{4.27}$$

The two expressions above together with (4.20) can be used to obtain

$$\rho_{k+1} \geq \min_x \ \hat{f}_k(y_{k+1}) + \langle p_k, x - x_{k+1}\rangle + \frac{u_k\|x - x_{k+1}\|^2}{2} = \hat{f}_k(y_{k+1}) - \frac{\|p_k\|^2}{2u_{k+1}}. \tag{4.28}$$

Therefore, by (4.23) we have

$$w_{k+1} = f(x_{k+1}) - \rho_{k+1} \leq f(x_{k+1}) - \hat{f}_k(y_{k+1}) + \frac{\|p_k\|^2}{2u_{k+1}}. \tag{4.29}$$

We have $f(x_{k+1}) \leq f(x_k)$ since we assume that $x_{k+1} = y_{k+1}$. The defiition of the descent direction $v_k = \hat{f}_k(y_{k+1}) - f(x_k)$ together with

$$\frac{\|p_k\|^2}{2u_{k+1}} \leq \frac{\|p_k\|^2}{2u_k} \leq -\frac{v_k}{2}, \tag{4.30}$$

which comes from the assumption $u_{k+1} \geq u_k$ and Lemma 1, yield the desired result

$$w_{k+1} \leq -v_k - \frac{v_k}{2} = -\frac{3v_k}{2}. \tag{4.31}$$

□

Lemma 6 *If $f^\infty > -\infty$, then v_k, w_k and $\|d_k\|$ all tend to 0 as $k \to \infty$.*

**Reader can find more detailed proof in [23].

Proof: Assume that the number of serious steps $l \to \infty$. By Lemma 4 and Lemma 5, we have

$$0 \leq -\frac{v_k}{2} \leq w_k \leq -\frac{3v_{k(l)-1}}{2} \text{ if } k(l) \leq k < k(l+1). \tag{4.32}$$

But when $l \to \infty$, then Lemma 2 implies that $v_{k(l)-1} \to \infty$. Hence, both $v_k \to 0$ and $w_k \to 0$. Next, fix l and assume that for all $k \geq k(l)$ we have

$$x_k = x_{k(l)} = \bar{x}. \tag{4.33}$$

So we have $u_{k+1} \geq u_k$ and $w_{k+1} \leq w_k$ for all $k \geq k(l)$ by Lemma 4. As $k \to \infty$, we also have

$$\|d_{k+1} - d_k\| \to 0, \tag{4.34}$$

and boundedness of v_k and $\|d_k\|$ by Lemma 4. Moreover, since f is assumed to be Lipschitz continuous on X, we have for all $k \geq k(l)$,

$$\|\xi_k^\intercal\| \leq L \tag{4.35}$$

by combining the inequalities (2.1) and (2.5) with $\xi_k \in f(y_k)$. Therefore, $\|\xi_k\|$ is bounded as well. Boundedness of v_k allows us to set $\bar{v} = \limsup_{k\to\infty} v_k$. Since $x_k = x_{k(l)} = \bar{x}$ for all $k \geq k(l)$, i.e., null steps, we have

$$f(y_{k+1}) - f(x_k) > \kappa v_k \tag{4.36}$$

by negating (3.20). Next,

$$\begin{aligned}
0 = \bar{v} - \bar{v} &= \limsup_{k\to\infty}(v_{k+1} - v_k + \|\xi_{k+1}\|\|d_{k+1} - d_k\|) \geq v_{k+1} - v_k + \|\xi_{k+1}\|\|d_{k+1} - d_k\| \\
&\geq \hat{f}_{k+1}(y_{k+2}) - \hat{f}_k(y_{k+1}) - \langle \xi_{k+1}, y_{k+2} - y_{k+1}\rangle \\
&= \hat{f}_{k+1}(y_{k+1}) - \hat{f}_k(y_{k+1}) \\
&= f(y_{k+1}) - \hat{f}_k(y_{k+1}) \\
&= f(y_{k+1}) - v_k - f(x_k) \\
&> (\kappa - 1)v_k = (1-\kappa)|v_k|,
\end{aligned} \tag{4.37}$$

so $v_k \to 0$ and $w_k \to 0$. By Lemma 1, also $\|d_k\| \to 0$. □

Lemma 7 *If (4.10) holds, then $x_k \to \bar{x} \in \Omega$ and also $f(x_k) \downarrow f(\bar{x})$*[††].

Proof: We have already shown in Lemma 3 that $\bar{x} \in X$, it is left to show that in fact, $\bar{x} \in \Omega$. By Lemma 6 we have $v_k \to 0$. The sequence $\{u_k\}$ is bounded by the construction of the algorithm ($u_{max} \geq u_k$ for all k). In view of Definition 2.2, by taking $k \to \infty$ in (4.8), we obtain our result. □

Theorem 4.1 *The sequence $\{x_k\}$ generated by PBM either converges to some $\bar{x} \in \Omega = \arg\min_X f$, or $\Omega = \emptyset$ and $\|x_k\| \to \infty$. In both cases we have $f(x_k) \downarrow f^* := \inf_X f$.*

Proof: If the condition (4.10) holds, i.e., $\Omega \neq \emptyset$, then by Lemma 7

$$f(x_k) \downarrow f(\bar{x}) = f(\tilde{x}). \tag{4.38}$$

Hence, $\tilde{x} \in \Omega$ and $f(\tilde{x}) = \inf_X f$. If (4.10) does not hold, then $\Omega = \emptyset$. In view of previous Lemmas, we conclude that PBM must terminate if $f^* > -\infty$ and the tolerance $\epsilon > 0$. □

4.2 LBM

Denote by $x_k^{\text{rec}} \in \{x_1, \ldots, x_k\}$ the point for which we upgrade the upper level bound $f(x_k^{\text{rec}}) = f_k^{up}$ in (3.39). We will prove that the sequence $\{x_k^{\text{rec}}\}$ generated by the proximal variant of LBM converges to some point in $\Omega = \arg\min_X f$, assuming $\Omega \neq \emptyset$.

While it is possible, with tedious technical modifications, to re-use the proof from Section 4.1 (see [4]), we present proof from [15]. In addition to convergence, we also provide an efficiency estimate for the algorithm.

[††]The notation $\downarrow$ means convergence from above

Similarly, as in the previous section, $k(l)$ will denote the iteration number of the lth serious step. Moreover, throughout this section, we denote by L the Lipschitz constant associated with f and

$$P_{\mathbb{X}_k}(\hat{x}_k) := \arg\min_{x\in\mathbb{X}_k} \frac{1}{2}\|x-\hat{x}_k\|^2 \tag{4.39}$$

will denote the projection of x on the level set $\mathbb{X}_k$.

Lemma 8 *If $k > k(l)$, then*

$$\|x_{k+1}-x_k\| \geq \frac{(1-\kappa)\Delta_k}{L}. \tag{4.40}$$

If $k = k(l)$, then

$$\|x_{k+1}-\hat{x}_k\| \geq \frac{(1-\kappa)\Delta_k}{L}. \tag{4.41}$$

Proof: By construction of the method, $f_k^{up} := f(x_k) \geq \min_{j=1,\dots,k} f(x_j)$. As $x_{k+1}\in\mathbb{X}_k$, then

$$f_j(x_{k+1}) = f(x_j) + \langle \xi_j, x_{k+1}-x_j\rangle \leq \hat{f}_k(x_{k+1}) \leq f_k^{lev}. \tag{4.42}$$

Thus, by the Lipschitz-continuity of f and (4.35),

$$L\|x_{k+1}-x_j\| \geq \|\xi_j\|\|x_{k+1}-x_j\| \geq f(x_k) - f_k^{lev} = f_k^{up} - (f^{up} - (1-\kappa)\Delta_k) = (1-\kappa)\Delta_k. \tag{4.43}$$

If $k = k(l)$, pick $j\in J_k$ such that $x_j = \hat{x}_k$ and if $k > k(l)$, put $j = k$. $\square$

Recall that our aggregation mechanism allows us to write

$$x_{k+1} = P_{\mathbb{X}_k}(\hat{x}_k) = P_{\mathbb{X}_k^{\hat{J}}}, \tag{4.44}$$

where

$$\mathbb{X}_k^{\hat{J}} = \{x\in X \mid f_k^{\hat{J}}(x) \leq f_k^{lev}\}, \tag{4.45}$$

and $f_k^{\hat{J}}$ is defined similarly as in (4.19).

Lemma 9 *If $k > k(l)$, then*

$$\|x_{k+1}-\hat{x}_k\|^2 \geq \|x_k-\hat{x}_k\|^2 + \|x_{k+1}-x_k\|^2, \tag{4.46}$$

and $\hat{x}_k = \hat{x}_{k-1}$.

Proof: Notice that $k > k(l)$ means $\hat{x} = \hat{x}_{k-1}$ by the construction of the algorithm. Moreover, we have

$$f_{k(l)}^{lev} \geq f_j^{lev} \geq f_k^{lev} \text{ if } k(l) < j \leq k \leq k(l+1), \tag{4.47}$$

because $f_{k-1}^{low} = f_k^{low}$ and $\Delta_{k-1} \leq \Delta_k$ due to (3.39). We also have $\hat{J}_{k-1}\subset J_k$ and $f_{k-1}^{\hat{J}}(\cdot) \geq f_k(\cdot)$ so $\mathbb{X}_k^{\hat{J}} \subset \mathbb{X}_k$. Notice that $x_{k+1}\in\mathbb{X}_k^{\hat{J}}$, $x_k = P_{\mathbb{X}_k}(\hat{x}_{k-1}) = P_{\mathbb{X}_k^{\hat{J}}}(\hat{x}_{k-1})$. Notice that then $x_{k+1}, \hat{x}_k, x_k$ all lie in $\mathbb{X}_k^{\hat{J}}$, so the result (4.46) follows simply from applying the triangle inequality. $\square$

Lemma 10 *If $k(l) \leq k < k(l+1)$ and $\Delta_k > 0$, then*

$$k - k(l) + 1 \leq \left(\frac{\operatorname{diam}(X)L}{(1-\kappa)\Delta_k}\right)^2. \tag{4.48}$$

Proof: We can apply Lemmas 8 and 9 with $\hat{x}_j = \hat{x}_{k(l)}$, for $j = k(l),\dots,k$, and $x_{k+1}, \hat{x}_k \in X$ to obtain

$$\begin{aligned} \operatorname{diam}(X)^2 = (\sup\{\|x-y\| \mid x,y\in X\})^2 \geq \|x_{k+1}-\hat{x}_k\|^2 + \sum_{j=k(l)+1}^{k} \|x_{j+1}-x_j\|^2 \geq \sum_{j=k(l)}^{k}\left(\frac{(1-\kappa\Delta_j)}{L}\right)^2 \\ \geq \left(\frac{(1-\kappa\Delta_k)}{L}\right)^2 (k-k(l)+1) \end{aligned} \tag{4.49}$$

as $\Delta_j \geq \Delta_k$ for all $j = 1, \ldots, k$ by construction. □

We can partition the iterations k into distinct groups as

$$\mathcal{K}_l = \{k(l), \ldots, k(l+1) - 1\}. \tag{4.50}$$

Recall, that if at the first iteration the problem (3.41) is infeasible, we choose a new $\tilde{f}_k^{low} \in (f_k^{low}, f_k^{lev}]$. So we get the relations

$$\tilde{\Delta}_k \geq \Delta_k, \text{ and } f_k^{up} \geq f_{k+1}^{up} \geq f^* := \inf_X f \geq f_{k+1}^{low} \geq \tilde{f}_k^{low} \geq f_k^{low}. \tag{4.51}$$

Lemma 11 *If $\tilde{\Delta}_{k_\epsilon} \geq \epsilon > 0$ for some k_ϵ, then*

$$k_\epsilon \leq \frac{(\operatorname{diam}(X)L)^2}{\epsilon^2(1-\kappa)^2(1-(1-\kappa)^2)}. \tag{4.52}$$

Proof: Let $\mathcal{K}(\epsilon) = \{1, \ldots, k_\epsilon\}$ so that

$$\mathcal{K}(\epsilon) \subset \bigcup_{l=0}^{m} \mathcal{K}_l, \tag{4.53}$$

for appropriate m. The fact that $\Delta_{k+1} \leq \tilde{\Delta}_k \leq \Delta_k$ for all k and (3.36) imply that

$$\tilde{\Delta}_k \leq (1-\kappa)\Delta_{k(l)}, \tag{4.54}$$

what means that $k \leq k(l)$ by the construction. Hence,

$$\Delta_k \geq \tilde{\Delta}_k \geq \frac{\tilde{\Delta}_{k(l+1)}}{1-\kappa} \text{ if } k \in \mathcal{K}_l,\ l \geq 0. \tag{4.55}$$

We claim that

$$\Delta_k \geq \frac{\epsilon}{(1-\kappa)^{m-l}}, \tag{4.56}$$

for all $k \in \mathcal{K}_l \cap \mathcal{K}(\epsilon)$, $l = 0, \ldots, m$. We can prove this by induction as follows.

We have $k(l+1) > k_\epsilon$ so $\tilde{\Delta}_{k_\epsilon} \geq \tilde{\Delta}_{k(l+1)}$. Let $\epsilon = \tilde{\Delta}_{k(l+1)}$ and pick $k_\epsilon \in \mathcal{K}(\epsilon) \cap \mathcal{K}_l$. Then the base case $m = 1$ follows from (4.55) ($l = 0$) and from the assumption in our lemma ($l = 1$).

Suppose now that, for $l = 0, \ldots, m$, it holds

$$\Delta_k \geq \tilde{\Delta}_{k_\epsilon} \geq \frac{\tilde{\Delta}_{k(l+1)}}{(1-\kappa)^{m-l}} = \frac{\epsilon}{(1-\kappa)^{m-l}}. \tag{4.57}$$

Then,

$$\begin{aligned}\frac{\epsilon}{(1-\kappa)^{(m+1)-l}} &= \frac{\epsilon}{(1-\kappa)(1-\kappa)^{m-l}} = \frac{1}{1-\kappa}\left(\frac{\epsilon}{(1-\kappa)^{m-l}}\right) \leq \frac{1}{1-\kappa}\tilde{\Delta}_{k_\epsilon} \leq (1-\kappa)\left(\frac{\Delta_k}{1-\kappa}\right) \\ &= \Delta_k.\end{aligned} \tag{4.58}$$

if $k \in \mathcal{K}(\epsilon) \cap \mathcal{K}_l$ by (3.36). This proves our claim.
Next, let

$$q := \left(\frac{\operatorname{diam}(X)L}{(1-\kappa)\epsilon}\right)^2. \tag{4.59}$$

Then by (4.56) and Lemma 10 we have,

$$|\mathcal{K}_l \cap \mathcal{K}(\epsilon)| \leq k - k(l) + 1 \leq \left(\frac{\operatorname{diam}(X)L}{(1-\kappa)\Delta_k}\right)^2 \leq \frac{(1-\kappa)^{2(m-l)}}{\epsilon^2}\left(\frac{\operatorname{diam}(X)L}{(1-\kappa)}\right)^2 = q(1-\kappa)^{2(m-l)}. \tag{4.60}$$

Using the summing formula for a geometric series,

$$k_\epsilon = \sum_{l=0}^{m} |\mathcal{K}_l \cap \mathcal{K}(\epsilon)| \leq \sum_{l=0}^{m} q(1-\kappa)^{2(m-l)} = q(1-\kappa)^2 m \frac{\frac{(1-\kappa)^{2(m+1)}-1}{(1-\kappa)^{2(m+1)}}}{\frac{(1-\kappa)^2-1}{(1-\kappa)^2}} = q\frac{(1-\kappa)^{2(m+1)}-1}{(1-\kappa)^2-1} \\ \leq \frac{q}{1-(1-\kappa)^2}, \tag{4.61}$$

which gives the result. □

Corollary 1 (Efficiency estimate) *If*

$$k > \frac{(\mathrm{diam}(X)L)^2}{\epsilon^2(1-\kappa)^2(1-(1-\kappa)^2)}, \tag{4.62}$$

then for any $\epsilon > 0$,

$$f_k^{up} - f^* \leq \Delta_k < \epsilon. \tag{4.63}$$

We can now present the main result of this section.

Theorem 4.2 *If the Proximal LBM algorithm does not terminate, then as $k \to \infty$ we have*

$$f_k^{up} \to f^*, \ f_k^{low} \to f^*, \ f_k^{lev} \to f^*, \ \Delta_k \to 0, \ \tilde{\Delta}_k \to 0. \tag{4.64}$$

Moreover, $x_k^{rec} \to \bar{x}$, for some $\bar{x} \in \Omega = \arg\min_X f$.

Proof: As we have seen, Δ_k, $\tilde{\Delta}_k$ are non-increasing so by Lemma 11 and the fact that $\Delta_{k+1} \leq \tilde{\Delta}_k \leq \Delta_k$, we have $\Delta_k \to 0$, $\tilde{\Delta}_k \to 0$. Next, we have

$$\Delta_k \geq \max\{|f_k^{low} - f^*|, |f_k^{up} - f^*|, |f_k^{lev} - f^*|\} \tag{4.65}$$

so all f_k^{low}, f_k^{up}, f_k^{lev} tend to 0 as $k \to \infty$. Lastly, we have $f(x_k^{\mathrm{rec}}) \downarrow f^*$, X is assumed to be compact and f Lipschitz continuous, so indeed

$$x_k^{\mathrm{rec}} \to \bar{x}, \text{ for some } \bar{x} \in \Omega. \tag{4.66}$$

□

5 Numerical experiments: Regularized Support Vector Machine

This section is dedicated to the numerical testing and empirical evaluation of the performance of PBM, Proximal LBM, and Non-Proximal LBM - in the context of solving the Support Vector Machine (SVM) problem. SVMs are commonly used for classification tasks in machine learning [5]. The aim is to find an optimal decision boundary between classes by maximizing the margin or distance between the decision boundary and the data points (x_i, y_i), where $x_i \in \mathbb{R}^N$ and $y_i \in \{-1, 1\}$, for $i = 1, \ldots, n$.

By measuring and analyzing factors such as CPU speed and accuracy, we aim to gain insights into the performance characteristics and capabilities of each method when applied to SVM. Through this evaluation, we aim to provide valuable empirical evidence regarding the suitability and practical applicability of bundle methods for solving SVM problems. For a more complex evaluation of the Bundle Methods applied to SVM and other machine learning models, we refer to [16].

Problem formulation
Consider the NSO problem

$$\min_w R_{emp}(w) + \lambda\Omega(w), \tag{5.1}$$

with the *empirical risk*

$$R_{emp}(w) := \frac{1}{m} \sum_{1 \leq i \leq m} l(x_i, y_i, w), \tag{5.2}$$

and the *Hinge loss function*

$$l(x, y, w) = \max\{0, 1 - y\langle x, w\rangle\}. \tag{5.3}$$

In binary classification, the Hinge loss penalizes misclassifications. It assigns a higher loss when the predicted value is on the wrong side of the decision boundary. The loss is zero when the predicted value is correctly classified. Its distinctive "hinge" shape renders the Hinge loss both non-differentiable and convex in nature [5]. Therefore, the SVM problem is well-suited to demonstrate the application of Bundle Methods in this context.

SVM typically employs the L_2 regularizer

$$\Omega_2(w) = \frac{1}{2}\|w\|_2^2, \tag{5.4}$$

which is known as Tikhonov regularization. Tikhonov regularization encourages the model coefficients to be small but non-zero, leading to a more evenly distributed impact across all features [20].

While it is less common, it is still possible to use a different norm for the regularization. We also consider the L_1 regularizer (SVML-1)

$$\Omega_1(w) = \frac{1}{2}\|w\|_1 = \frac{1}{2}\sum_{j=1}^{N}|w_j|. \tag{5.5}$$

This is known as LASSO regularization. LASSO regularization promotes sparsity by driving some coefficients to zero, resulting in a sparse model that selects only the most relevant features.

Notice that we do not need to compute the whole subdifferential of $J_k(w) := R_{emp}(w) + \lambda\Omega_k(w)$, $k \in \{1, 2\}$, but it suffices to get a one subgradient at each $w \in dom(J_k)$ as follows,

$$\partial_w J_k = \Gamma + \lambda\partial_w \Omega_k, \tag{5.6}$$

where

$$\Gamma = \frac{1}{m}\sum_{1\leq i\leq m} \partial_w \max\{0, 1 - y_i\langle x_i, w\rangle\}, \tag{5.7}$$

and

$$\partial_w \max\{0, 1 - y\langle x, w\rangle\} = \begin{cases} -yx, \text{ if } 1 \geq y\langle x, w\rangle \\ 0, \text{ otherwise.} \end{cases} \tag{5.8}$$

For the squared L_2 norm, at any w, the subgradient is just $2w$. In the case of L_1 norm, $q \in \mathbb{R}^N$ is a subgradient of $\|w\|_1$ at $w = (w_1, \dots, w_N)^\intercal$ if

$$q_j = \begin{cases} 1, \text{ if } w_j > 0 \\ -1, \text{ if } w_j < 0 \\ r \in [-1, 1], \text{ if } w_j = 0, \end{cases} \tag{5.9}$$

for $j \in \{1, \dots, N\}$. It is often preferred to set $r = 0$ to promote the sparsity in the model [20].

We say that the step x_k is critical if $t_L^k = 1$, that means, a serious step was made, and if a descent step was made with respect to the "true" error. It is convenient to plot the critical steps against the "true" error as both the Proximal and Non-Proximal LBM does not necessarily make the descent step at each iteration.

To compute the "true" solutions and therefore to make a comparison with our algorithms, we use the built-in function `minimize` from the module `scipy.optimize` from the Python library `SciPy`. To compute the stabilizing QP subproblems (3.27) and (3.41) as well as the LP problem (3.32), we use the Python library `cvxpy` which is well-suited for convex optimization.

Parameters choice, assumptions, and data setup
We set $\kappa = 0.5$ for the PBM and $\kappa = 0.75$ for both the Proximal and Non-Proximal LBM. The tolerance is taken to be $\epsilon = 10^{-6}$. We pick $M_\xi = 10^4$ as the maximum allowed bundle size. This is indeed a big number, at least bigger than what is normally used in practice. However, it suffices for the scope of this testing. The constant λ in front of the regularization term $\Omega(\cdot)$ is taken to be $\lambda = 0.0001$. We use `random` to generate various data samples. The code is provided in Appendix B.

Remark 5 For the PBM, we may see the problem as the unconstrained one ($X = \mathbb{R}^N$) for simplicity. For the Proximal and Non-Proximal LBM, we need to bound the feasible set X to satisfy the compactness. By leveraging heuristics, we can infer that the solution to our problem is in close proximity to $\mathbf{0} \in \mathbb{R}^N$. This enables us to select a small radius r and define X as the closed ball $X = \mathcal{B}_r^N(\mathbf{0})$. In our experiments, we set $r = 0.01$.

Remark 6 We use the following simple updating technique for the proximity parameter u_k. We fix $u_{min} = u_1 = 2$ and $u_{max} = 10000$. We pick $u_{k+1} \in [u_{min}, \lceil u_k \rceil]$ at random after the serious step and $u_{k+1} \coloneqq \min\{u_{max}, 2.5u_k\}$ after the null step. There exists far more refined updating techniques ([13], [10]), but in order to achieve convergence, it is sufficient for the sequence $\{u_k\}$ to be bounded[‡‡] and non-decreasing after the null step is made. This can be seen from the convergence analysis we provide in Section 4.1.

Remark 7 Due to the nature of the objective function $J(\cdot)$, and especially the hinge loss, both LBMs methods seem to make very few serious steps. However, each time they do, the resulting descent is often significant. We decided to make the following adjustment. In addition to (3.36), we force the serious step if there were `ForcedSerStep` $= 50$ consecutive null steps made.

Remark 8 The bundle aggregation technique is implemented in practice as follows. We drop those multipliers λ_j from the index set J_k, which are in the ball $\mathcal{B}_\delta(0)$ for a small $\delta > 0$. We take $\delta \in [10^{-30}, 10^{-7}]$ depending on the dimension size N.

Remark 9 To avoid some unexpected behaviour that can result in prolonged run-time (for instance, if ϵ is not small enough), a sentinel value is set at the iteration $k = 1000$ forcing the algorithms to terminate if this value is reached.

Numerical experiments
We fix $n = 1000$, which denotes the number of data points (x_i, y_i). The testing was performed in two phases.

First, we vary the dimensions $N \in \{2, 5, 30, 60, 100, 200\}$ and we test the performance of each of the three Bundle Methods discussed. The resulting table with results Table 1 together with selected plots can be found in Appendix A.

Next, we vary the dimensions $N \in \{10^3, 10^4\}$, but we only use the PBM and the Proximal LBM for this test. The main aim is to test the performance in terms of CPU time. We perform 15 tests with $N = 10^3$, 2 tests with $N = 10^4$ for the PBM and 4 tests with $N = 10^4$ for the Proximal LBM. The resulting table with results together with selected plots can be found in Table 2 in Appendix A.

The stopping rules $v_k \geq -\epsilon$ and $\Delta_k \leq \epsilon$ did not always project to the "true" accuracy, causing unsatisfactory results even after the stopping rule was triggered. This was the issue, especially for the Proximal LBM. Thus, we decided to add an extra, artificial stopping rule (with respect to the "true" error) to both algorithms for the tests with $N = 10^4$. This significantly improved the speed of the Proximal LBM. The effect on the PBM was less significant.

6 Conclusion

The primary objective of this paper is to study Bundle Algorithms for solving convex non-smooth optimization problems. We start by exploring various concepts from the mathematical optimization theory. The central focus of this paper revolves around providing a formal description and conducting a convergence analysis of two distinct classes of Bundle Methods. Specifically, our attention is directed toward the Proximal Bundle Method (PBM) and two variations of the Level Bundle Method (LBM): the Proximal LBM and the Non-Proximal LBM.

In addition to the primary objective, this paper also aims to gain a comprehensive understanding of the historical development surrounding the PBM and LBM. Therefore, a lot of effort is spent on studying the proximal and projection operators, polyhedral models, Cutting-Planes Method, optimality conditions for non-smooth optimization problems as well as techniques for managing

[‡‡]In fact, after some subtle changes to the algorithm are made, $\{u_k\}$ can even be unbounded, see [13].

optimization constraints through the Lagrange Multiplier Theorem and the KKT conditions. The convergence analysis in Section 4 is conducted with a direct, explicit, and mathematically rigorous approach.

Lastly, we examine the suitability of PBM, Proximal LBM, and Non-Proximal LBM to the Regularized Support Vector Machine (SVM) problem. Our extensive analysis reveals that all three algorithms show commendable accuracy and reliability. For the SVM problem, the performance of these methods is significantly influenced by the CPU speed and the ability to handle information storage. These factors play a crucial role in determining the overall suitability of the algorithms for the given application of Bundle Methods.

6.1 Results of the numerical experiments

Despite its stability and reliability, the Non-Proximal LBM exhibits the slowest CPU time among the three methods. Indeed, this should not be a big surprise since the Non-Proximal LBM needs to solve both QP and LP problems at each iteration, without utilizing any aggregation technique for efficient storage. Consequently, the computation of each iterate is considerably expensive, resulting in longer CPU times. As a result, this method may not be particularly suitable for datasets with large dimensions. However, it performs well with smaller datasets, as the computed solution closely approximates the exact solution.

In contrast, the Proximal LBM demonstrates enhanced efficiency in terms of memory storage and CPU time. However, it does have a few drawbacks. It requires careful tuning and is highly sensitive to the specific characteristics of the objective function, constraints, and data. When appropriately configured, this method can be exceptionally efficient, particularly for datasets with large dimensions. When compared to the PBM, the Proximal LBM exhibits superior results in both accuracy and CPU time for datasets with large dimensions. Perhaps the biggest issue with this method is that the stopping rule $\Delta_k \leq \epsilon$ might not be always triggered at the right time, causing either premature or delayed termination. We suppose that this is because of the fact that the produced sequence $\{x_k\}_{k\in\mathbb{N}}$ is not necessarily non-increasing. We show in Section 4.2 that only the critical points x_k^{rec} are converging to the minimizer $\bar{x} \in \Omega = \arg\min_X f$. Hence, the algorithm can accumulate a lot of non-critical iterations until the condition $\Delta_k \leq (1-\kappa)\,\Delta_{k(l)}$ is satisfied. At the same time, the optimality gap Δ_k can slowly continue getting thinner, potentially triggering the stopping rule before the next critical iteration is computed. This, however, can be avoided, or at least mitigated, through precise tuning of the algorithm. In general, the Proximal LBM is capable of producing better results when provided with more comprehensive information regarding the objective function, constraints, and underlying data.

Lastly, the PBM seems to be overall the most reliable method in terms of accuracy, CPU time, and memory usage. Unlike the Proximal LBM, it requires minimal tuning while maintaining impressive speed, particularly for smaller datasets. However, its performance can worsen when handling larger datasets. This limitation can be mitigated by implementing a more refined updating scheme for the proximity parameter u_k and by having more comprehensive information about the nature of the specific optimization problem.

Appendix A

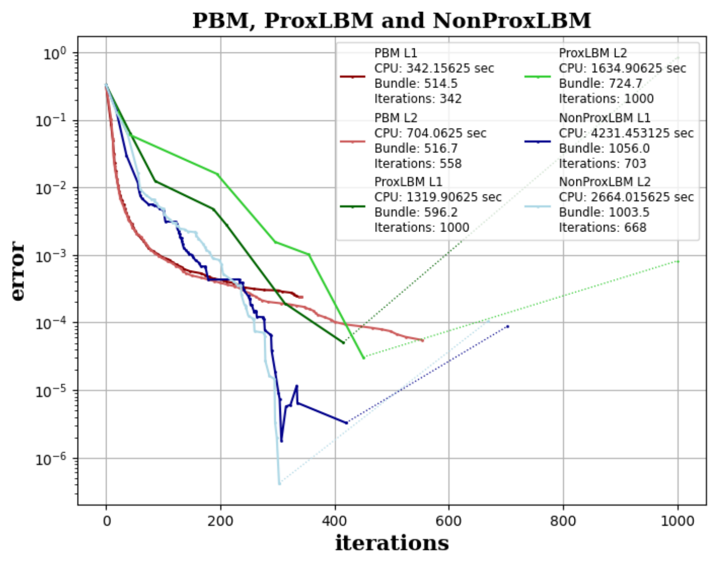

Figure .1: Error plot with all three Bundle Methods used with $N = 200$. The dotted lines connect the last critical iteration with the iteration for which the stopping rule was triggered.

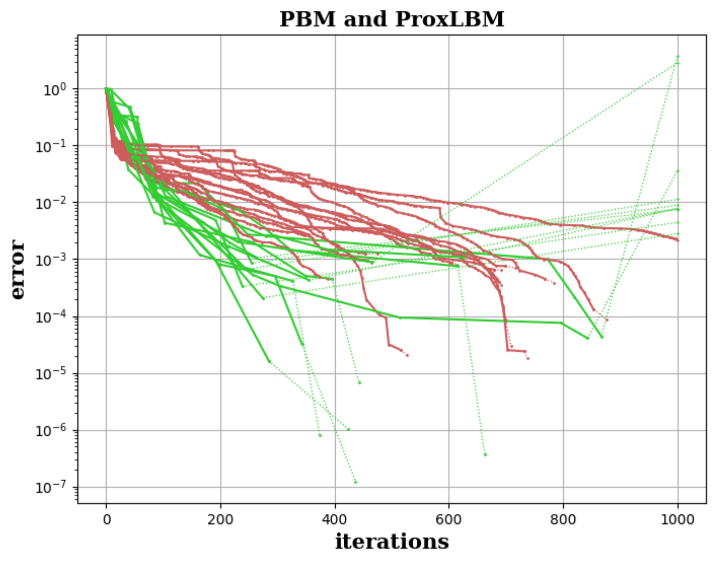

Figure .2: Error plot showing the 15 experiments with $N = 1000$. Only the PBM and the Proximal LBM were used. The green curves denote the Proximal LBM, and the red curves denote the PBM.

Lasso L_1 PBM				
N	CPU	k_{total}	ϵ_{acc}	$\bar{B}$
2	0.66sec	16	5.13×10^{-6}	17.6
5	1.31sec	23	7.81×10^{-6}	26.5
30	22.84sec	93	8.34×10^{-6}	97.9
60	28.77sec	96	6.1×10^{-5}	121.9
100	1min 33sec	380	5.25×10^{-5}	161.9
200	5min 42sec	342	4.72×10^{-4}	514.5

Tikhonov L_2 PBM				
N	CPU	k_{total}	ϵ_{acc}	$\bar{B}$
2	0.61sec	15	8.83×10^{-7}	16.4
5	1.45sec	23	2.69×10^{-7}	26.5
30	21.44sec	88	1.97×10^{-5}	99.1
60	1min 15sec	152	9.98×10^{-6}	205.4
100	1min 21sec	341	4.87×10^{-5}	159.0
200	11min 44sec	558	8.37×10^{-5}	516.7

Lasso L_1 Proximal LBM				
N	CPU	k_{total}	ϵ_{acc}	$\bar{B}$
2	1.24sec	26	3.79×10^{-8}	11.8
5	1.98sec	41	7.30×10^{-6}	14.9
30	46.61sec	238	4.04×10^{-5}	76.4
60	1min 22sec	243	8.12×10^{-5}	139.5
100	10min 12sec	1000	8.90×10^{-6}	451.3
200	22min	1000	8.14×10^{-5}	596.2

Tikhonov L_2 Proximal LBM				
N	CPU	k_{total}	ϵ_{acc}	$\bar{B}$
2	0.95sec	20	9.82×10^{-8}	13.7
5	1.70sec	39	1.97×10^{-6}	14.2
30	42.52sec	237	1.01×10^{-7}	67.9
60	3min 9sec	430	9.39×10^{-6}	190.4
100	6min 20sec	1000	9.79×10^{-6}	276.5
200	27min 15sec	1000	6.10×10^{-5}	724.7

Lasso L_1 Non-Proximal LBM				
N	CPU	k_{total}	ϵ_{acc}	$\bar{B}$
2	1.50sec	16	1.01×10^{-6}	25.5
5	4.64sec	30	8.67×10^{-6}	46.5
30	53.73sec	108	4.01×10^{-5}	163.5
60	3min 58sec	191	2.57×10^{-5}	288.0
100	7min 57sec	347	7.83×10^{-8}	522.0
200	1h 10min 31sec	703	6.75×10^{-6}	1056.0

Tikhonov L_2 Non-Proximal LBM				
N	CPU	k_{total}	ϵ_{acc}	$\bar{B}$
2	2.25sec	20	1.01×10^{-6}	31.5
5	3.92sec	28	2.17×10^{-6}	43.5
30	52.83sec	110	6.37×10^{-7}	166.5
60	3min 46sec	185	8.87×10^{-7}	279.0
100	6min 35sec	316	9.07×10^{-6}	475.5
200	44min 24sec	668	4.91×10^{-7}	1003.5

Table .1: N | CPU time | Number of iterations | Error accuracy | Average bundle size at each iteration

Tikhonov L_2 regularization		
Method	N	Average CPU
PBM	1000	5min 2sec
ProxLBM	1000	9min 42sec

Tikhonov L_2 regularization		
Method	N	CPU
PBM	10000	3h 59min 37sec
PBM	10000	2h 36min 33sec
ProxLBM	10000	17.78sec
ProxLBM	10000	10min 31sec
ProxLBM	10000	29.57sec
ProxLBM	10000	1h 22min 18sec

Table .2: Method used | N | CPU time

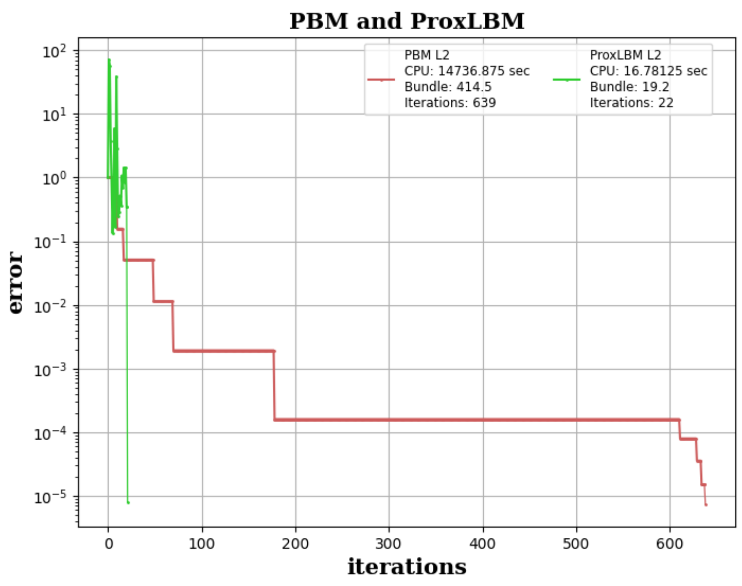

Figure .3: This oddly-looking plot shows both the critical and the non-critical iterations used with $N = 10000$ for the PBM and the Proximal LBM. Notice that significant descent step in the Proximal LBM made and the rather slow, but stable convergence of the PBM. Tikhonov L_2 regularization was used.

Appendix B

Defining the objective function and its subgradients

```
def omega(w, lamb):  # Tikhonov L2
    return (lamb / 2) * (LA.norm(w) ** 2)

def lassoregularizer(w, lamb):  # LASSO L1
    return (lamb / 2) * np.sum(np.abs(w))

def subgrl1(w):  # subgradients of L1
    subgrad = np.zeros_like(w)

    for i, p in enumerate(w):
        if p < 0:
            subgrad[i] = -1
        elif p > 0:
            subgrad[i] = 1
        else:
            # For p = 0, any value between -1 and 1 is a valid subgradient, but the value
                0 is preferred
            subgrad[i] = 0

    return subgrad

def hinge_loss(w, x, y):  # Hinge loss
    return np.maximum(0, 1 - y * np.dot(x, w))

def R(w):  # empirical risk term sum
    term = 0
    for x, y in data:
        term = term + hinge_loss(w, x, y)
    return (1 / len(data)) * term

def Jobj(w, lamb):  # Tikhonov L2 SVM
    return omega(w, lamb) + R(w)

def Jobjl1(w, lamb):  # LASSO L1 SVM
    return lassoregularizer(w, lamb) + R(w)

def subgr_hinge_loss(w, x, y):  # subgradients of the hinge loss
    if 1 >= y * np.dot(x, w):
        return -y * x
    else:
        return 0

def gamma(w):  # support term for the subgradients SVM
    term = 0
    for x, y in data:
        term = term + subgr_hinge_loss(w, x, y)
    return (1 / len(data)) * term

def subgr_J(w, lamb):  # SVM subgradients Tikhonov L2
    return lamb * w + gamma(w)

def subgrl1_J(w, lamb):  # SVM subgradients LASSO L1
    return (lamb / 2) * subgrl1(w) + gamma(w)
```

PBM

```
def PBM(w, yT, kappa, test, M, u_max, u, tol, func, subgr, trueres):  # PBM
    # initialization
    J = [1]
    yLIST = [yT]
    subgrLIST = [subgr(yT, test)]
    v_descent = -1000000  # the predicted descent set to a large negative number at the
        beginning
    total_size = 3  # size of the bundle = |J|+|yLIST|+|subgrLIST| = 3*|J|
    total_information = [3]
    errors = [np.linalg.norm(trueres.fun - func(w, test))]  # comparing with the "true"
        solution
    k = 1
    l = 0
    kl = 0
    crit_iterations = [kl]
    # PBM's stopping rule and artificial stopping rule (used in the second phase of the
        testing)
    while v_descent < -tol and errors[-1] > 0.00001:
        old_l = l
        # solving the QP problem using cvxpy
        x = cp.Variable(len(yT))
        tau = cp.Variable(1)
        prox = cp.Minimize((0.5 * u) * (cp.norm(x - w) ** 2) + tau)  # the proximal
            operator
        constraints = []
        for j in J:  # adding the constraints
            constraints.append(func(yLIST[j - 1], test) + cp.sum(cp.multiply(subgrLIST[j
                - 1], w - yLIST[j - 1])) +
                                   cp.sum(cp.multiply(subgrLIST[j - 1], x - w)) <= tau)
        problem = cp.Problem(prox, constraints)
        try:
            # Solve the problem using the 'ECOS' solver
            problem.solve(solver='ECOS')
        except cp.SolverError:
            # If 'ECOS' fails, try using the 'SCS' solver
            try:
                problem.solve(solver='SCS')
            except cp.SolverError:
                print("Both 'ECOS' and 'SCS' solvers failed to solve the problem.")
                break
        yT = x.value  # solution y_{k+1}
        tauSolved = tau.value  # solution \hat{f}_{k}(y_{k+1})
        KKTmultipliers = []
        for constrain in constraints:
            KKTmultipliers.append(constrain.dual_value)
        v_descent = tauSolved - func(w, test)  # predicted descent
        if func(yT, test) - func(w, test) <= kappa * v_descent:
            l += 1
            w = yT  # serious step
            # updating scheme for the proximity parameter
            u_new = random.randint(2, math.ceil(u))
            u = u_new
        else:
            w = w  # null step
            # updating scheme for the proximity parameter
            u = 2.5 * u
            while u > u_max:
                u = u_max
            pass
        k += 1
        # bundle (index set) aggregation
        J_hat = [J[j] for j in range(len(J)) if np.abs(KKTmultipliers[j]) >
            0.000000000000000000000001]
        J = J_hat + [k]
        if old_l < l and errors[-1] > np.linalg.norm(res.fun - func(w, test)):
            kl = k
            errors.append(np.linalg.norm(trueres.fun - func(w, test)))
            crit_iterations.append(kl)
        total_information.append(3 * len(J))
        yLIST.append(yT)
        xi_subgr = subgr(yT, test)
        subgrLIST.append(xi_subgr)
        total_size = total_size + 3 * len(J)
        if len(J) <= M:  # we can bound the size of the index set
            pass
        else:
            bundle = total_size / k
            if kl != k:
                crit_iterations.append(k)
                errors.append(np.linalg.norm(trueres.fun - func(w, test)))
            return w, total_information, crit_iterations, errors, bundle, k
        if k > 999:
            # sentinel set at k=1000
            bundle = total_size / k
            if kl != k:
                crit_iterations.append(k)
                errors.append(np.linalg.norm(res.fun - func(w, test)))
            return w, total_information, crit_iterations, errors, bundle, k
    bundle = total_size / k
    if kl != k:
        crit_iterations.append(k)
        errors.append(np.linalg.norm(trueres.fun - func(w, test)))
    return w, total_information, crit_iterations, errors, bundle, k
```

Non-proximal LBM

```
def nonproxLBM(w, kappa, tol, test, func, subgr, trueres):  # NonProx LBM
    global f_low
    # initialization
    k = 1
    J = [1]
    wLIST = [w]
    total_size = 3
    l = 0
    kl = 0
    crit_iterations = [kl]
    total_information = [3]
    errors = [np.linalg.norm(trueres.fun - func(w, test))]
    check = True  # flag for checking whether serious or null step was made
    xi_subgr = subgr(w, test)
    subgrLIST = [xi_subgr]
    support = -0.1 * np.ones(len(w))  # the feasible set for (non)prox LBM has to be
        compact in general
    # solving the LP problem at the first iteration
    x = cp.Variable(len(w))
    y = cp.Variable(1)
    objective = cp.Minimize(y)
    constraints = [func(w, test) + cp.sum(cp.multiply(xi_subgr, x - w)) <= y, x >=
        support, x <= -support]
    LPproblem = cp.Problem(objective, constraints)
    try:
        # Solve the problem using the 'ECOS' solver
        f_low = LPproblem.solve(solver='ECOS')
    except cp.SolverError:
        # If 'ECOS' fails, try using the 'SCS' solver
        try:
            f_low = LPproblem.solve(solver='SCS')
        except cp.SolverError:
            print("Both 'ECOS' and 'SCS' solvers failed to solve the problem.")
    f_up = func(w, test)
    opt_gap = f_up - f_low
    opt_gapSS = opt_gap
    forcedSerStep = 0  # to avoid accumulation of null steps
    while opt_gap > tol:
        old_l = l
        if (opt_gap <= ((1 - kappa) * opt_gapSS)) or (forcedSerStep > 50):  # serious
            step
            l += 1
            opt_gapSS = opt_gap
            if len(J) > 1:
                if check:
                    j = J[-1]
                else:
                    j = random.choice(J[-2:])
            else:
                j = random.choice(J)
            w_proj = wLIST[j - 1]  # critical step
        else:  # null step
            w_proj = w
            forcedSerStep += 1
        while True:
            f_lev = f_low + kappa * opt_gap
            z = cp.Variable(len(w))
            QPproj = cp.Minimize(0.5 * (cp.norm(z - w_proj) ** 2))
            constraints = []
            for j in J:
                constraints.append(func(wLIST[j - 1], test) + cp.sum(cp.multiply(
                    subgrLIST[j - 1], z - wLIST[j - 1]))
                                - f_lev <= 0)
            constraints.append(cp.norm(z) <= cp.norm(support))
            constraints.append(z <= cp.Constant(0.01))
            constraints.append(z >= cp.Constant(-0.01))
            # controlling the stability by preventing big oscillations in the "true"
                errors
            problem = cp.Problem(QPproj, constraints)
            try:
                # Solve the problem using the 'ECOS' solver
                problem.solve(solver='ECOS')
            except cp.SolverError:
                # If 'ECOS' fails, try using the 'SCS' solver
                try:
                    problem.solve(solver='SCS')
                except cp.SolverError:
                    print("Both 'ECOS' and 'SCS' solvers failed to solve the problem.")
            if problem.status == 'optimal' or problem.status == 'optimal_inaccurate':
                break
            elif problem.status == 'infeasible' or problem.status == '
                infeasible_inaccurate':
                f_low = f_lev  # f_lev  is certainly bigger than f_low
                opt_gap = f_up - f_low
                print("infeasible solution, updating f_low")
            else:
                print("QP problem ", problem.status)
                return w, total_information, k, errors

        w = z.value
        wLIST.append(w)
        if old_l < l and errors[-1] > np.linalg.norm(res.fun - func(w, test)):
            kl = k
            errors.append(np.linalg.norm(trueres.fun - func(w, test)))
            crit_iterations.append(kl)
        xi_subgr = subgr(w, test)
        subgrLIST.append(xi_subgr)
        if f_up < func(w, test):
            check = False
        else:
            check = True
        f_up = np.minimum(f_up, func(w, test))
        J = J + [k + 1]  # no aggregation method is used for NonProx LBM
        # in the nonproxLBM we need to solve that LP problem at each iteration
        x2 = cp.Variable(len(w))
        y2 = cp.Variable(1)
        LPobj = cp.Minimize(y2)
        constraints = []
        for j in J:
            constraints.append(func(wLIST[j - 1], test) + cp.sum(cp.multiply(subgrLIST[j
                - 1], x2 - wLIST[j - 1]))
                              <= y2)
        constraints.append(x2 >= support)
        constraints.append(x2 <= -support)
        problemLP = cp.Problem(LPobj, constraints)
        try:
            # Solve the problem using the 'ECOS' solver
            problemLP.solve(solver='ECOS')
        except cp.SolverError:
            # If 'ECOS' fails, try using the 'SCS' solver
            try:
                problemLP.solve(solver='SCS')
            except cp.SolverError:
                print("Both 'ECOS' and 'SCS' solvers failed to solve the problem.")
        # upgrading f_low when necessary
        f_new_low = y2.value
        f_low = np.maximum(f_low, f_new_low)
        opt_gap = f_up - f_low
        k += 1
        total_information.append(3 * len(J))
        total_size = total_size + 3 * len(J)
        if k > 999:
            # sentinel set at k=1000
            bundle = total_size / k
            if kl != k:
                crit_iterations.append(k)
                errors.append(np.linalg.norm(res.fun - func(w, test)))
            return w, total_information, crit_iterations, errors, bundle, k
    bundle = total_size / k
    if kl != k:
        crit_iterations.append(k)
        errors.append(np.linalg.norm(trueres.fun - func(w, test)))
    return w, total_information, crit_iterations, errors, bundle, k
```

Proximal LBM

```
def proxLBM(w, kappa, tol, M, test, func, subgr, trueres):  # Prox LBM
    global f_low
    # initialization
    k = 1
    J = [1]
    l = 0
    kl = 0
    crit_iterations = [kl]
    wLIST = [w]
    w_proj = w
    xi_subgr = subgr(w, test)
    subgrLIST = [xi_subgr]
    total_size = 3
    total_information = [3]
    errors = [np.linalg.norm(trueres.fun - func(w, test))]
    support = -0.1 * np.ones(len(w))  # feasible set needs to be bounded (compact) in
        general
    # solving the LP problem at the first iteration
    x = cp.Variable(len(w))
    y = cp.Variable(1)
    objective = cp.Minimize(y)
    constraints = [func(w, test) + cp.sum(cp.multiply(xi_subgr, x - w)) <= y, x >=
        support, x <= -support]
    LPproblem = cp.Problem(objective, constraints)
    try:
        # Solve the problem using the 'ECOS' solver
        f_low = LPproblem.solve(solver='ECOS')
    except cp.SolverError:
        # If 'ECOS' fails, try using the 'SCS' solver
        try:
            f_low = LPproblem.solve(solver='SCS')
        except cp.SolverError:
            print("Both 'ECOS' and 'SCS' solvers failed to solve the problem.")
    f_up = func(w, test)
    opt_gap = f_up - f_low
    check = True
    opt_gapSS = opt_gap
    forcedSerStep = 0
    # LBM's stopping rule and artificial stopping rule (used in the second phase of the
        testing)
    while opt_gap > tol and errors[-1] > 0.00001:
        old_l = l
        # selecting the projection point
        if (opt_gap <= ((1 - kappa) * opt_gapSS)) or forcedSerStep > 45:  # serious step
            forcedSerStep = 0
            l += 1
            opt_gapSS = opt_gap
            if len(J) > 1:
                if check:
                    j = J[-1]  # correct
                else:
                    j = random.choice(J[-2:])
            else:
                j = random.choice(J)
            w_proj = wLIST[j - 1]
        else:  # null step
            forcedSerStep += 1
            pass
        # QP problem
        while True:
            f_lev = f_low + kappa * opt_gap
            z = cp.Variable(len(w))
            QPproj = cp.Minimize(0.5 * (cp.norm(z - w_proj) ** 2))
            constraints = []
            for j in J:
                constraints.append(func(wLIST[j - 1], test) + cp.sum(cp.multiply(
                    subgrLIST[j - 1], z - wLIST[j - 1]))
                                   - f_lev <= 0)
            constraints.append(cp.norm(z) <= cp.norm(support))
            constraints.append(z <= cp.Constant(0.01))
            constraints.append(z >= cp.Constant(-0.01))
            # avoiding big oscillations
            problem = cp.Problem(QPproj, constraints)
            try:
                # Solve the problem using the 'ECOS' solver
                problem.solve(solver='ECOS')
            except cp.SolverError:
                # If 'ECOS' fails, try using the 'SCS' solver
                try:
                    problem.solve(solver='SCS')
                except cp.SolverError:
                    print("Both 'ECOS' and 'SCS' solvers failed to solve the problem.")
            if problem.status == 'optimal' or problem.status == 'optimal_inaccurate':
                break
            elif problem.status == 'infeasible' or problem.status == '
                infeasible_inaccurate':
                f_low = f_lev  # f_lev is certainly bigger than f_low
                opt_gap = f_up - f_low
            else:
                return w, total_information, k, errors
        KKTmultipliers = []
        for constrain in constraints:
        KKTmultipliers.append(constrain.dual_value)
        w = z.value
        if old_l < l and errors[-1] > np.linalg.norm(res.fun - func(w, test)):
            kl = k
            errors.append(np.linalg.norm(trueres.fun - func(w, test)))
            crit_iterations.append(kl)
        wLIST.append(w)
        xi_subgr = subgr(w, test)
        subgrLIST.append(xi_subgr)
        if f_up < func(w, test):
            check = False
        else:
            check = True
        f_up = np.minimum(f_up, func(w, test))
        opt_gap = f_up - f_low
        k += 1
        # bundle (index set) aggregation
        J_hat = [J[j] for j in range(len(J)) if np.abs(KKTmultipliers[j]) >
            0.000000000001]
        J = J_hat + [k]
        total_size = total_size + 3 * len(J)
        total_information.append(3 * len(J))
        if len(J) <= M:
            pass
        else:
            bundle = total_size / k
            if kl != k:
                crit_iterations.append(k)
                errors.append(np.linalg.norm(res.fun - func(w, test)))
            return w, total_information, crit_iterations, errors, bundle, k
        if k > 999:
            # sentinel set at k=1000
            bundle = total_size / k

            if kl != k:
                crit_iterations.append(k)
                errors.append(np.linalg.norm(res.fun - func(w, test)))
            return w, total_information, crit_iterations, errors, bundle, k
    bundle = total_size / k
    if kl != k:
        crit_iterations.append(k)
        errors.append(np.linalg.norm(trueres.fun - func(w, test)))
    return w, total_information, crit_iterations, errors, bundle, k
```

References

[1] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3): 334–334, 1997.

[2] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A Sagastizábal. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006.

[3] Jonathan Borwein and Adrian Lewis. *Convex Analysis*. Springer, 2006.

[4] Ulf Brännlund, Krzysztof C Kiwiel, and Per Olof Lindberg. A descent proximal level bundle method for convex nondifferentiable optimization. *Operations Research Letters*, 17(3):121–126, 1995.

[5] Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 2020.

[6] George B Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 1990.

[7] Welington de Oliveira and CLAUDIA Sagastizábal. Level bundle methods for oracles with on-demand accuracy. *Optimization Methods and Software*, 29(6):1180–1209, 2014.

[8] Mateo Díaz and Benjamin Grimmer. Optimal convergence rates for the proximal bundle method. *arXiv preprint arXiv:2105.07874*, 2021.

[9] Stefan Feltenmark and Krzysztof C Kiwiel. Dual applications of proximal bundle methods, including lagrangian relaxation of nonconvex problems. *SIAM Journal on Optimization*, 10(3):697–721, 2000.

[10] A. Fuduli and M. Gaudioso. Tuning strategy for the proximity parameter in convex minimization. *Journal of Optimization Theory and Applications*, 130(1):95–112, 2006.

[11] Ralph Gomory. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960.

[12] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

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

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

[15] Krzysztof C Kiwiel. Proximal level bundle methods for convex nondifferentiable optimization, saddle-point problems and variational inequalities. *Mathematical Programming*, 69(1-3):89–109, 1995.

[16] Quoc Le, Alex Smola, and Svn Vishwanathan. Bundle methods for machine learning. *Advances in neural information processing systems*, 20, 2007.

[17] Claude Lemaréchal, Arkadii Nemirovskii, and Yurii Nesterov. New variants of bundle methods. *Mathematical programming*, 69:111–147, 1995.

[18] David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*, volume 2. Springer, 1984.

[19] Marko Mäkelä. Survey of bundle methods for nonsmooth optimization. *Optimization methods and software*, 17(1):1–29, 2002.

[20] Jianyu Miao and Lingfeng Niu. A survey on feature selection. *Procedia computer science*, 91:919–926, 2016.

[21] Welington de Oliveira and Claudia Sagastizábal. Bundle methods in the xxist century: A bird's-eye view. *Pesquisa Operacional*, 34:647–670, 2014.

[22] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1 (3):127–239, 2014.

[23] R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

[24] R Tyrrell Rockafellar. *Convex analysis*, volume 11. Princeton university press, 1997.

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

[26] Mícheál Ó Searcóid. *Metric spaces.* Springer-Verlag Berlin Heidelberg, 2007.

[27] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4): 637–672, 2020.

[28] Stephen J Wright and Benjamin Recht. *Optimization for data analysis.* Cambridge University Press, 2022.