IPIANO: INERTIAL PROXIMAL ALGORITHM FOR NON-CONVEX OPTIMIZATION

DAVID STUTZ*

Abstract. This paper studies the minimization of non-convex and non-smooth composite functionsIn particular, we discuss the algorithm proposed by Ochs et al. in [15], called iPiano. Following [15], we present a global convergence result for functions satisfying the Kurdyka-Lojasiewicz property [12, 11] which is based on the work by Attouch et al. [4]. Furthermore, we discuss the implementation of iPiano and apply the algorithm to image denoising and image segmentation. In contrast to [15], we use simple denoising functionals instead of Fields of Expert [19, 7] to demonstrate that simple functionals may benefit from non-smooth and non-convex terms. Finally, we demonstrate the applicability of the algorithm to image segmentation based on a 2-phase fields approximation of the Mumford-Shah functional [20].

1. Introduction. Many image processing and computer vision problems involve complex, possibly non-smooth and/or non-convex optimization problems. Solving these optimization problems efficiently is as important as attaining globally optimal solutions. On the one hand, this requires efficient algorithms and implementations, while on the other hand proving global convergence may require rather involved mathematics and stricter assumptions on the objective to be optimized. In the simplest case, i.e. involving a convex and smooth objective, gradient-based approaches and proximal algorithms offer global convergence, see [6] and [16] for details. Proximal algorithms are also applicable to non-smooth objectives, however, computing the proximal map may not necessary be trivial. More recently, splitting algorithms for non-smooth composite objectives have been studied, see [8]. For non-convex objectives, establishing global convergence is – in general – much harder. Recent efforts focussed on objectives satisfying the Kurdyika-Lojasiewicz property [12, 11] for which certain algorithms have been shown to converge [4]. In this paper, we study the algorithm proposed by Ochs et al. [15] named iPiano meant for non-smooth and non-convex composite functions of the form introduced in Section 1.2.

1.1. Outline. We first present the general problem statement and the necessary mathematical background in Sections 1.2 and 1.3, respectively. Subsequently we briefly discuss relevant related work in Section 2. In the main part, Section 3, we present the iPiano algorithm and several variants, present convergence analysis and discuss convergence rate as well as implementation details. Finally, in Section 4, we discuss applications before concluding in Section 5.

1.2. Problem Statement. In this paper we focus on so-called composite functions (also called C^1-perturbations of convex functions) of the form

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

where $f : \mathbb{R}^n \to \mathbb{R}$ is required to be in C^1 but might be non-convex, while $g : \text{dom}(g) \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is required to be convex but might be non-smooth. We further require f to have L-Lipschitz continuous gradient ∇f. For the minimization problem in Equation (1) to be well-posed, we require h to be bounded below by some value

$$-\infty < h_{\min} := \inf_{x\in\mathbb{R}^n} h(x)$$

Furthermore, h is assumed to be coercive, i.e. $h(x) \to \infty$ for $\|x\|_2 \to \infty$. With $\text{dom}(g) := \{x \in \mathbb{R}^n | g(x) < \infty\}$ being the effective domain of g and $\text{dom}(g) \subset \text{dom}(f)$, it follows that $\text{dom}(h) = \text{dom}(g)$ has to be convex.

1.3. Preliminaries. Before considering the first-order condition for Equation (1) we formally restate the requirement on h to be coercive.

DEFINITION 1. *A function $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is called coercive if there exist constants $p, C > 0$*

*Submitted as part of the seminar "Ausgewählte Themen der Bildverarbeitung", WS 2015/2016, and advised by Prof. Berkels, AICES, RWTH Aachen University.

and $q \geq 0$ such that

$$h(x) \geq C\|x\|_2^p - q \quad \forall x \in dom(h).$$

Considering the minimization problem in Equation (1), the first-order condition for $x^* \in \mathbb{R}^n$ to be a critical point of h is

$$0 \in \partial h(x^*) \Leftrightarrow 0 \in \nabla f(x^*) + \partial g(x^*) \Leftrightarrow -\nabla f(x^*) \in \partial g(x^*)$$

where $\nabla f(x^*) + \partial g(x^*) := \{\nabla f(x^*) + y | y \in \partial g(x^*)\}$ and $\partial g(x^*)$ is the subdifferential of g at position x^*. In the following, we consider g to be proper, closed and convex:

DEFINITION 2. *The function $g : dom(g) \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is*
(a) convex if

$$g(tx + (1-t)\hat{x}) \leq tg(x) + (1-t)g(\hat{x}) \quad \forall x, \hat{x} \in dom(g), t \in [0,1];$$

(b) proper if $dom(g) \neq \emptyset$;
(c) and closed if $graph(g) := \{(x, \xi) \in \mathbb{R}^n \times \mathbb{R} | \xi \leq h(x)\}$ is closed.

Furthermore, we expect g to be lower semicontinuous, i.e. for all $\hat{x} \in \mathrm{dom}(g)$ it holds

$$\lim \inf_{x^{(n)} \to x} f(x^{(n)}) \geq f(x).$$

The first-order condition then follows directly from the following definition and the subsequent lemmata:

DEFINITION 3. *Let $g : dom(g) \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a proper convex and lower semi continuous function. Then, the subdifferential $\partial g(x)$ of g at position $x \in dom(g)$ is defined as*

$$\partial g(x) := \{w \in \mathbb{R}^n | g(x) + w^T(\hat{x} - x) \leq g(\hat{x}) \forall \hat{x} \in \mathbb{R}^n\}. \tag{2}$$

The above definition requires g to be proper convex and lower semi continuous. In contrast, Ochs et al. originally introduce (following [18, Def. 8.3]) the Fréchet subdifferential, i.e.

$$\hat{\partial} g(x) := \{w \in \mathbb{R}^n | \lim \inf_{x^{(n)} \to x, x^{(n)} \neq x} \frac{g(x^{(n)}) - g(x) - w^T(x^{(n)} - x)}{|x^{(n)} - x|} \geq 0\},$$

and the limiting subdifferential, i.e.

$$\begin{aligned}\partial g(x) :=& \{w \in \mathbb{R}^n | \exists (x^{(n)})_{n \in \mathbb{N}} \subset \mathbb{R}^n, (w^{(n)})_{n \in \mathbb{N}} \subset \mathbb{R}^n \\ & \text{such that } x^{(n)} \to x, g(x^{(n)}) \to g(x), w^{(n)} \to w \text{ and } w^{(n)} \in \hat{\partial} g(x^{(n)})\}.\end{aligned}$$

However, following [18, Prop. 8.12], the equivalence of Equations (1.3) and (1.3) follows if g is proper convex.

Following [18, Thm. 8.6], we first present a basic property of the subdifferential and subsequently establish the first-order condition mentioned before:

LEMMA 1. *For $g : dom(g) \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ and $x \in \mathbb{R}^n$ with $g(x) < \infty$, the set $\partial g(x)$ is closed.*

LEMMA 2. *Let $h = f + g$ with $f : \mathbb{R}^n \to \mathbb{R}$ in C^1 and $g : dom(g) \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ convex. Then, for $x \in dom(h)$ it holds*

$$\partial h(x) = \nabla f(x) + \partial g(x).$$

Now, we have a closer look at f; the following lemma proves a useful property of f with L-Lipschitz continuous gradient ∇f:

LEMMA 3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be in C^1 with L-Lipschitz continuous gradient ∇f, i.e. it exists $L > 0$ is such that*

$$\|\nabla f(x) - \nabla f(\hat{x})\|_2 \leq L\|x - \hat{x}\|_2, \quad \forall x, \hat{x} \in \mathbb{R}^n.$$

Then it holds:

$$f(x) \leq f(\hat{x}) + \nabla f(\hat{x})^T(x - \hat{x}) + \frac{L}{2}\|x - \hat{x}\|_2^2, \quad \forall x, \hat{x} \in \mathbb{R}^n. \tag{3}$$

Note that Ochs et al. require f to have L-Lipschitz continuous gradient ∇f on $\mathrm{dom}(g)$ only; this implies that Equation (3) holds for all $x, \hat{x} \in \mathrm{dom}(g)$.

Turning to g, we first define the proximal mapping $\mathrm{prox}_{\alpha g}$, which is the basis of proximal algorithms. We note that further discussion and properties can be found in [16] and [8].

DEFINITION 4. *For $\hat{x} \in \mathbb{R}^n$ and $g : dom(g) \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ being a proper closed convex and lower semicontinuous function, the proximal mapping is defined as*

$$prox_{\alpha g}(\hat{x}) := \arg\min_{x \in \mathbb{R}^n} \frac{\|x - \hat{x}\|_2^2}{2} + \alpha g(x) \tag{4}$$

where $\alpha > 0$ is a given parameter.

Note that within the literature, the proximal mapping is often written as

$$(I + \alpha \partial g)^{-1}(\hat{x}) := \mathrm{prox}_{\alpha g}(\hat{x})$$

and called resolvent of the subdifferential. Here, I refers to the identity mapping. The requirements on g stated in Definition 4 are necessary for $\mathrm{prox}_{\alpha g}$ to be well-defined, i.e. there exist a unique minimizer of Equation (4).

2. Related Work. The work by Ochs et al. combines two different lines of research: splitting algorithms and multistep algorithms. Splitting algorithms originate from the proximal point algorithm. These splitting algorithms usually address problems of the form

$$\min_{x \in \mathbb{R}^n} f_1(x) + \ldots + f_k(x)$$

for convex functions $f_i : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ which may not be smooth. Different algorithms have been proposed; we refer to the work by Combettes and Pesquet [8] providing a thorough overview of the different approaches. Furthermore, they show several useful properties of the proximal mapping as given by Definition 4 including an extensive list of known proximal mappings for a wide range of functions. Ochs et al. partly built upon [4] where an abstract convergence theorem for functions satisfying the Kurdyika-Lojasiewicz property is shown (see [12] and [11]). Multistep algorithms were first considered by Polyak [17] and use a so-called inertial force (also called momentum term) to improve convergence. In practice, the inertial force is usually computed as the difference of two preceding iterates. Ochs et al. extend the work by Attouch et al. [4] by combining a backward-forward splitting scheme with an inertial force.

3. iPiano. Following [15] we present different variants of the proposed algorithm, called iPiano, and discuss convergence for functions satisfying the Kurdyika-Lojasiewicz property (formally introduced later) as well as the corresponding convergence rate.

3.1. Algorithm. Essentially, iPiano combines forward-backward splitting as discussed in [8] with a inertial force (also termed momentum term or multistep term). Given an initial starting point $x^{(0)} \in \mathbb{R}^n$ and setting $x^{(-1)} = x^{(0)}$, iterates are computed as

$$x^{(n+1)} = \operatorname{prox}_{\alpha_n g}\left(x^{(n)} - \alpha_n \nabla f(x^{(n)}) + \beta_n (x^{(n)} - x^{(n-1)})\right) \tag{5}$$

for step size parameters $(\alpha_n)_{n\in\mathbb{N}}$ and momentum parameters $(\beta_n)_{n\in\mathbb{N}}$. Obviously, the critical aspect of this generic scheme is choosing these parameters appropriately. In the following we discuss different means of choosing α_n and β_n.

3.1.1. ciPiano – Algorithm 1. Of course, both the step size parameter and the momentum parameter can be chosen apriori based on knowledge about f. In particular, we need to estimate the global Lipschitz constant L of ∇f in order to choose $\alpha < \frac{2(1-\beta)}{L}$.

Algorithm 1 ciPiano: iPiano with constant stepsize and momentum parameter.

1: choose $\beta \in [0, 1)$
2: choose $\alpha < \frac{2(1-\beta)}{L}$
3: choose $x^{(0)}$
4: $x^{(-1)} := x^{(0)}$
5: // Fixed number of iterations; or ϵ-criterion.
6: **for** $n = 1, \ldots$ **do**
7: $\quad x^{(n+1)} = \operatorname{prox}_{\alpha g}\left(x^{(n)} - \alpha \nabla f(x^{(n)}) + \beta(x^{(n)} - x^{(n-1)})\right)$
8: **end for**

3.1.2. miPiano – Algorithm 2. Estimating L automatically can be accomplished via backtracking: given $x^{(n)}$ and $x^{(n+1)}$, the estimate of the Lipschitz constant L_n in iteration $n \geq 0$ is required to satisfy

$$f(x^{(n+1)}) \leq f(x^{(n)}) + \nabla f(x^{(n)})^T (x^{(n+1)} - x^{(n)}) + \frac{L_n}{2} \|x^{(n+1)} - x^{(n)}\|_2^2 \tag{6}$$

which follows from Lemma 3. Usually, $x^{(n+1)}$ is re-estimated until a

$$L_n \in \{L_{n-1}, \eta L_{n-1}, \eta^2 L_{n-1}, \ldots\}$$

is found satisfying Equation (6). Here, $\eta > 1$ implicitly defines the step size of backtracking. Note that L_n merely represents a local estimate of the Lipschitz constant. After estimating L_n, Algorithm 2 adapts the step size parameter α_n in each iteration while keeping β fixed.

3.1.3. biPiano – Algorithm 3. In contrast, Algorithm 3 adapts both α_n and β_n based on the latest estimate of L_n. Based on auxiliary constants $\delta \geq c_2 > 0$ the updates of α_n and β_n are governed by

$$\beta_n = \frac{(b-1)}{(b-\frac{1}{2})}, \quad \text{with} \quad b = \frac{(\delta + \frac{L_n}{2})}{(c_2 + \frac{L_n}{2})};$$
$$\alpha_n = \frac{2(1-\beta_n)}{(2c_2 + L_n)}.$$

The relationship between δ and c_2 determines how large L_n has to be in order for β_n to tend towards zero.

3.1.4. iPiano – Algorithm 4. The convergence statements derived in the following sections are based on the general update rules in Algorithm 4 which subsumes Algorithms 1 - 3 as special cases. However, Algorithm 4 introduces nondeterminism, see Lines 12 and 13, which needs to be resolved in an actual implementation. In the following Lemmata we discuss existence and derive some basic properties and bounds of the parameters α_n, β_n as well as δ_n and γ_n. These insights will be used both for convergence analysis and the implementation.

Algorithm 2 nmiPiano: iPiano with backtracking to determine the step size parameter.

choose $\beta \in [0, 1)$
choose $L_{-1} > 0$
choose $\eta > 1$
choose $x^{(0)}$
$x^{(-1)} := x^{(0)}$
// Fixed number of iterations; or ϵ-criterion.
for $n = 1, \ldots$ **do**
  $L_n := \frac{1}{\eta} L_{n-1}$ // Initial estimate of L_n is L_{n-1} or alternatively use (27).
  repeat
    $L_n := \eta L_n$ // Next estimate of L_n.
    choose $\alpha_n < \frac{2(1-\beta)}{L_n}$
    $\tilde{x}^{(n+1)} = \operatorname{prox}_{\alpha_n g}\left(x^{(n)} - \alpha_n \nabla f(x^{(n)}) + \beta(x^{(n)} - x^{(n-1)})\right)$ // Re-estimate $\tilde{x}^{(n+1)}$.
  until (6) is satisifed for $\tilde{x}^{(n+1)}$
  //Estimated L_n satisfies Equation (6) and $\tilde{x}^{(n+1)}$ is next iterate:
  $x^{(n+1)} := \tilde{x}^{(n+1)}$
end for

Algorithm 3 biPiano: iPiano with backtracking for both the step size parameter and the momentum parameter.

choose $\delta \geq c_2 > 0$ // With c_2 close to zero.
choose $L_{-1} > 0$
choose $\eta > 1$
choose $x^{(0)}$
$x^{(-1)} := x^{(0)}$
// Fixed number of iterations; or ϵ-criterion.
for $n = 1, \ldots$ **do**
  $L_n := \frac{1}{\eta} L_{n-1}$ // Initial estimate of L_n is L_{n-1} or alternatively use (27).
  repeat
    $L_n := \eta L_n$ // Next estimate of L_n.
    $\beta_n := \frac{(b-1)}{(b-\frac{1}{2})}$ with $b := \frac{(\delta + \frac{L_n}{2})}{(c_2 + \frac{L_n}{2})}$
    $\alpha_n := \frac{2(1-\beta_n)}{(2c_2 + L_n)}$
    $\tilde{x}^{(n+1)} = \operatorname{prox}_{\alpha_n g}\left(x^{(n)} - \alpha_n \nabla f(x^{(n)}) + \beta_n(x^{(n)} - x^{(n-1)})\right)$ // Re-estimate $\tilde{x}^{(n+1)}$.
  until (6) is satisifed for $\tilde{x}^{(n+1)}$
  //Estimated L_n satisfies Equation (6) and $\tilde{x}^{(n+1)}$ is next iterate:
  $x^{(n+1)} := \tilde{x}^{(n+1)}$
end for

LEMMA 4. *For each $n \in \mathbb{N}$, there exist $\alpha_n < 2(1-\beta_n)/L_n$, $0 \leq \beta_n < 1$ such that $c_2 \leq \gamma_n \leq \delta_n$.*

Proof. By the definition of δ_n and γ_n in Algorithm 4 it directly follows that $0 < \gamma_n \leq \delta_n$. Now, rearranging

$$\begin{aligned} & \gamma_n := \frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{\alpha_n} \geq c_2 \\ \Leftrightarrow \quad & \alpha_n \leq \frac{1-\beta_n}{c_2 + \frac{L_n}{2}} < \frac{2(1-\beta_n)}{L_n} \end{aligned} \tag{7}$$

the upper bound for α_n follows and for $\beta_n < 1$ and c_1 small enough the existence of an appropriate α_n follows. ∎

Unfortunately, Ochs et al. do not prove the existence of an $\alpha_n \geq c_1$ for fixed but arbitrary c_1. The following result will be useful for both the subsequent convergence analysis and for choosing β_n in practice:

Algorithm 4 iPiano.

1: choose $c_1, c_2 > 0$ // With c_1, c_2 close to zero.
2: choose $L_{-1} > 0$
3: choose $\eta > 1$
4: choose $x^{(0)}$
5: $x^{(-1)} := x^{(0)}$
6: // Fixed number of iterations; or ϵ-criterion.
7: **for** $n = 1, \ldots$ **do**
8: $L_n := \frac{1}{\eta} L_{n-1}$ // Initial estimate of L_n is L_{n-1} or alternatively use (27).
9: **repeat**
10: $L_n := \eta L_n$ // Next estimate of L_n.
11: **repeat**
12: choose $\alpha_n \geq c_1$
13: choose $\beta_n \geq 0$
14: $\delta_n := \frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{2\alpha_n}$
15: $\gamma_n := \frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{\alpha_n}$
16: **until** $\delta_n \geq \gamma_n \geq c_2$
17: $\tilde{x}^{(n+1)} = \operatorname{prox}_{\alpha_n g}\left(x^{(n)} - \alpha_n \nabla f(x^{(n)}) + \beta_n (x^{(n)} - x^{(n-1)})\right)$ // Re-estimate $\tilde{x}^{(n+1)}$.
18: **until** (6) is satisifed for $\tilde{x}^{(n+1)}$
19: //Estimated L_n satisfies Equation (6) and $\tilde{x}^{(n+1)}$ is next iterate:
20: $x^{(n+1)} := \tilde{x}^{(n+1)}$
21: **end for**

LEMMA 5. *For each $n \in \mathbb{N}$, given $L_n > 0$, there exist α_n and β_n as in Lemma 4 such that $(\delta_n)_{n\in\mathbb{N}}$ is monotonically decreasing.*

Proof. Consider the following inequalities, equivalent to the descent property of $(\delta_n)_{n\in\mathbb{N}}$:

$$\begin{aligned} \delta_{n-1} \geq \delta_n &:= \frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{2\alpha_n} \\ \Leftrightarrow \quad \alpha_n &\geq \frac{1 - \frac{\beta_n}{2}}{\delta_{n-1} + \frac{L_n}{2}}. \end{aligned} \tag{8}$$

Requiring the existence of an α_n fulfilling both Equations (7) and (8) is equivalent to

$$\frac{1 - \frac{\beta_n}{2}}{\delta_{n-1} + \frac{L_n}{2}} \leq \frac{1 - \beta_n}{c_2 + \frac{L_n}{2}} \quad \Leftrightarrow \quad \frac{1 - \frac{\beta_n}{2}}{1 - \beta_n} \leq \frac{\delta_{n-1} + \frac{L_n}{2}}{c_2 + \frac{L_n}{2}}.$$

We define

$$b_n := \frac{\delta_{n-1} + \frac{L_n}{2}}{c_2 + \frac{L_n}{2}} \tag{9}$$

such that Equation (9) can be written as

$$\beta_n \leq \frac{b_n - 1}{b_n - \frac{1}{2}}$$

and for $b_n \geq 1$ the existence of an appropriate $\beta_n \in [0, 1)$ follows. Note that for $\beta_n < 1$, the existence of an appropriate α_n follows. ∎

From the above lemmata, the rules for choosing α_n and/or β_n in Algorithms 2 and 3 are explained and can directly be used to avoid the non-determinism in Lines 12 and Lines 13 of Algorithm 4 as done in Section 3.3. However, we first discuss convergence.

3.2. Convergence Analysis. In [15], Ochs et al. state an abstract convergence result (based in large parts on [4]) which is in turn applied to

$$H_{\delta_n}(x) := h(x) + \delta_n \|x - y\|_2^2 \tag{10}$$

where $h(x)$ is the composite function from Section 1.2. The convergence analysis is based on the following three conditions which have been adapted from [4] and are required to be fulfilled by the sequence $(z^{(n)})_{n\in\mathbb{N}} := (x^{(n)}, x^{(n-1)})_{n\in\mathbb{N}} \subset \mathbb{R}^n \times \mathbb{R}^n$ generated by Algorithm 4 and the function H_{δ_n}:

DEFINITION 5. *A sequence $(z^{(n)})_{n\in\mathbb{N}} := (x^{(n)}, x^{(n-1)})_{n\in\mathbb{N}} \subset \mathbb{R}^n \times \mathbb{R}^n$ and a proper lower semi-continuous function $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ satisfy Conditions (H1) - (H3) for fixed $a, b > 0$ and $\Delta_n := \|x^{(n)} - x^{(n-1)}\|_2$ if:*

(H1) for each $n \in \mathbb{N}$, it holds

$$H(z^{(n+1)}) + a\Delta_n^2 \leq H(z^{(n)});$$

(H2) for each $n \in \mathbb{N}$, there exists $w^{(n+1)} \in \partial H(z^{(n+1)})$ with

$$\|w^{(n+1)}\|_2 \leq \frac{b}{2}(\Delta_n + \Delta_{n+1});$$

(H3) there exists a subsequence $(z^{(n_j)})_{j\in\mathbb{N}}$ with

$$z^{(n_j)} \to \tilde{z} = (\tilde{x}, \tilde{x}) \quad \text{and} \quad H(z^{(n_j)}) \to H(\tilde{z}), \quad j \to \infty.$$

In the following, we show that H_{δ_n} satifies Conditions (H1) - (H3), starting with Condition (H1):

LEMMA 6. *H_{δ_n} satisfies (H1), in particular for each $n \in \mathbb{N}$*

$$H_{\delta_{n+1}}(z^{(n+1)}) + \gamma_n \Delta_n^2 \leq H_{\delta_n}(z^{(n)}) \tag{11}$$

with H_{δ_n} as in Equation (10) and δ_n, γ_n as in Algorithm 4.

Proof. From Equation (5), Definition 4 and the form of g, i.e. proper and convex, it follows:

$$\frac{x^{(n)} - x^{(n+1)}}{\alpha_n} - \nabla f(x^{(n)}) + \frac{\beta_n}{\alpha_n}(x^{(n)} - x^{(n-1)}) \in \partial g(x^{(n+1)}).$$

With Lemma 3 and Equation (2) (setting $\hat{x} = x^{(n)}$, $x = x^{(n+1)}$ and $L = L_n$) we have

$$f(x^{(n+1)}) \leq f(x^{(n)}) + \nabla f(x^{(n)})^T(x^{(n+1)} - x^{(n)}) + \frac{L_n}{2}\|x^{(n)} - x^{(n+1)}\|_2^2$$

and

$$g(x^{(n+1)}) \leq g(x^{(n)}) - w^T(x^{(n)} - x^{(n+1)}) = g(x^{(n)}) + w^T(x^{(n+1)} - x^{(n)}) \tag{12}$$

for $w \in \partial g(x^{(n+1)})$. If we choose

$$\begin{aligned} w &:= \frac{x^{(n)} - x^{(n+1)}}{\alpha_n} - \nabla f(x^{(n)}) + \frac{\beta_n}{\alpha_n}(x^{(n)} - x^{(n-1)}) \\ &= -\frac{x^{(n+1)} - x^{(n)}}{\alpha_n} - \nabla f(x^{(n)}) + \frac{\beta_n}{\alpha_n}(x^{(n)} - x^{(n-1)}) \end{aligned} \tag{13}$$

and consider $h(x^{(n+1)}) = f(x^{(n+1)}) + g(x^{(n+1)})$ we get

$$\begin{aligned} h(x^{(n+1)}) &\leq h(x^{(n)}) + \left(\nabla f(x^{(n)}) + w\right)^T \left(x^{(n+1)} - x^{(n)}\right) + \frac{L_n}{2} \underbrace{\|x^{(n)} - x^{(n+1)}\|_2^2}_{=\Delta_{n+1}^2} \\ &\overset{(13)}{=} h(x^{(n)}) - \left(\frac{1}{\alpha_n} - \frac{L_n}{2}\right) \Delta_{n+1}^2 + \frac{\beta_n}{\alpha_n} \underbrace{(x^{(n+1)} - x^{(n)})^T (x^{(n)} - x^{(n-1)})}_{\leq \frac{1}{2}\|x^{(n+1)} - x^{(n)}\|_2^2 + \frac{1}{2}\|x^{(n)} - x^{(n-1)}\|_2^2} \\ &\leq h(x^{(n)}) - \left(\frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{2\alpha_n}\right) \Delta_{n+1}^2 + \frac{\beta_n}{2\alpha_n} \Delta_n^2. \end{aligned} \tag{14}$$

Note that Equations (12) and (13) have explicitly been rewritten in order to match the signs used in the above derivation. Using the definition of δ_n and γ_n in Algorithm 4, i.e.

$$\delta_n := \frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{2\alpha_n} \quad \text{and} \quad \gamma_n := \frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{\alpha_n},$$

it easily follows that

$$\delta_n - \gamma_n = \frac{\beta_n}{2\alpha_n}.$$

Substituting this into Equation (14), we get

$$\begin{aligned} & h(x^{(n+1)}) \leq h(x^{(n)}) - \delta_n \Delta_{n+1}^2 + \delta_n \Delta_n^2 - \gamma_n \Delta_n^2 \\ \Leftrightarrow \quad & h(x^{(n+1)}) + \delta_n \Delta_{n+1}^2 \leq h(x^{(n)}) + \delta_n \Delta_n^2 - \gamma_n \Delta_n^2 \\ \Leftrightarrow \quad & H_{\delta_{n+1}}(x^{(n+1)}, x^{(n)}) + \gamma_n \Delta_n^2 \leq H_{\delta_n}(x^{(n)}, x^{(n-1)}) \end{aligned}$$

where the latter equivalence follows from the monotonicity of δ_n established in Lemma 5. Furthermore, as $\gamma_n > 0$ by Algorithm 4, the sequence $(H(z^{(n+1)}))_{n\in\mathbb{N}} = (H(x^{(n+1)}, x^{(n)}))_{n\in\mathbb{N}}$ is monotonically decreasing and converges as h is bounded below by the requirements made in Section 1.2. ∎

Note that the above lemma also justifies the requirement of $(\delta_n)_{n\in\mathbb{N}}$ being monotonically decreasing as discussed in Lemma 5. We continue with requirement (H2):

LEMMA 7. *H_{δ_n} satisfies (H2), i.e. for each $n \in \mathbb{N}$ and for a fixed $b > 0$, there exists $w^{(n+1)} \in \partial H_{\delta_{n+1}}(z^{(n+1)})$ with*

$$\|w^{(n+1)}\|_2 \leq \frac{b}{2}(\Delta_n + \Delta_{n+1}).$$

Proof. Considering the definition of H_{δ_n} as

$$H_{\delta_n}(z^{(n+1)}) = H_{\delta_n}(x^{(n+1)}, x^{(n)}) = h(x^{(n+1)}) + \delta_n \|x^{(n+1)} - x^{(n)}\|_2^2$$

we have $w^{(n+1)} := (w_1^{(n+1)}, w_2^{(n+1)})$ with

$$\begin{aligned} w_1^{(n+1)} &\in \underbrace{\partial g(x^{(n+1)}) + \nabla f(x^{(n+1)})}_{=\partial h(x^{(n+1)})} + 2\delta_n (x^{(n+1)} - x^{(n)}); \\ w_2^{(n+1)} &= -2\delta_n (x^{(n+1)} - x^{(n)}) \end{aligned}$$

following directly from Lemma 2 and basic rules of differentiation. Again, considering

$$\frac{x^{(n)} - x^{(n+1)}}{\alpha_n} - \nabla f(x^{(n)}) + \frac{\beta_n}{\alpha_n}(x^{(n)} - x^{(n-1)}) \in \partial g(x^{(n+1)}),$$

as done in the proof of Lemma 6, we get

$$\begin{aligned}
\|w^{(n+1)}\|_2 \leq& \|w_1^{(n+1)}\|_2 + \|w_2^{(n+1)}\|_2 \\
\leq& \frac{\|x^{(n)} - x^{(n+1)}\|_2}{\alpha_n} + \|\nabla f(x^{(n+1)}) - \nabla f(x^{(n)})\|_2 \\
&+ \frac{\beta_n}{\alpha_n}\|x^{(n)} - x^{(n-1)}\|_2 + 4\delta_n\|x^{(n+1)} - x^{(n)}\|_2 \\
=& \left(\frac{1}{\alpha_n} + 4\delta_n\right)\|x^{(n+1)} - x^{(n)}\|_2 + \|\nabla f(x^{(n+1)}) - \nabla f(x^{(n)})\|_2 \\
&+ \frac{\beta_n}{\alpha_n}\|x^{(n)} - x^{(n-1)}\|_2 \\
\leq& \left(\frac{1}{\alpha_n} + 4\delta_n\right)\|x^{(n+1)} - x^{(n)}\|_2 + L_n\|x^{(n+1)} - x^{(n)}\|_2 \\
&+ \frac{\beta_n}{\alpha_n}\|x^{(n)} - x^{(n-1)}\|_2 \\
=& \left(\frac{1}{\alpha_n} + 4\delta_n + L_n\right)\Delta_{n+1} + \frac{\beta_n}{\alpha_n}\Delta_n
\end{aligned}$$

where we used the L_n-Lipschitz continuity of ∇f. Using the definition of δ_n in Algorithm 4,

$$\delta_n := \frac{1}{\alpha_n} - \frac{L_n}{2} - \frac{\beta_n}{2\alpha_n},$$

and the upper bound on α_n proved in Lemma 4,

$$\alpha_n < 2(1-\beta_n)L_n,$$

we get

$$\begin{aligned}
L_n\alpha_n < 2(1-\beta_n) &\overset{\beta_n\in[0,1)}{\leq} 2; \\
\delta_n\alpha_n = 1 - \frac{\alpha_n L_n}{2} - \frac{\beta_n}{2} &\leq 1.
\end{aligned}$$

Overall, with $\alpha_n > c_1$ it follows that

$$\|w^{(n+1)}\|_2 \leq \left(\frac{1}{\alpha_n} + 4\delta_n + L_n\right)\Delta_{n+1} + \frac{\beta_n}{\alpha_n}\Delta_n \leq \underbrace{\frac{7}{c_1}}_{=:b}(\Delta_{n+1} + \Delta_n). \qquad \square$$

For (H3), we need to do some more work:

LEMMA 8. *It holds*

$$\sum_{n=0}^{\infty} \Delta_n^2 < \infty$$

and, thus, it must also hold $\lim_{n\to\infty}\Delta_n = 0$.

Proof. Summing up Condition (H1), i.e. Equation (11), for $n = 0, \ldots, N$ yields

$$\begin{aligned}
\sum_{n=0}^{N}\gamma_n\Delta_n^2 \leq& \sum_{n=0}^{N} H_{\delta_n}(x^{(n)}, x^{(n-1)}) - H_{\delta_{n+1}}(x^{(n+1)}, x^{(n)}) \\
=& h(x^{(0)}) - H_{\delta_{N+1}}(x^{(N+1)}, x^{(N)}) \leq h(x^{(0)}) - h_{\min} < \infty
\end{aligned}$$

where $H_\delta(x^{(0)}, h^{(-1)}) = h(x^{(0)})$ and $h_{\min}$ is the lower bound of h. As $\gamma_n \geq c_2 > 0$ in the requirements of Algorithm 4, the claim follows. $\square$

LEMMA 9. *The sequence $(h(x^{(n)}))_{n\in\mathbb{N}}$ converges and it exists a converging subsequence $(x^{(n_j)})_{n\in\mathbb{N}}$ such that any limit point $x^* := \lim_{j\to\infty} x^{(n_j)}$ is a critical point of h and $h(x^{(n_j)}) \to h(x^*)$ for $j \to \infty$.*

Proof. We start by proving that $(h(x^{(n)}))_{n\in\mathbb{N}}$ converges. Therefore, we note that

$$H_{-\delta_n}(x^{(n)}, x^{(n-1)}) = H_{\delta_n}(x^{(n)}, x^{(n-1)}) - 2\delta_n\Delta_n^2$$

and by Lemma 8 we have

$$\lim_{n\to\infty} H_{-\delta_n}(x^{(n)}, x^{(n-1)}) = \lim_{n\to\infty} H_{\delta_n}(x^{(n)}, x^{(n-1)}) - \underbrace{2\delta_n\Delta_n^2}_{\to 0} = \lim_{n\to\infty} H_{\delta_n}(x^{(n)}, x^{(n-1)}).$$

From the squeeze theorem, i.e.

$$H_{-\delta_n}(x^{(n)}, x^{(n-1)}) \leq h(x^{(n)}) \leq H_{\delta_n}(x^{(n)}, x^{(n-1)})$$

and the convergence of $(H(z^{(n)}))_{n\in\mathbb{N}}$ the claim follows.

Now, we prove that there is a converging subsequence $(x^{(n_j)})_{n\in\mathbb{N}}$. As $H(x^{(n)}, x^{(n-1)})$ is monotonically decreasing, we have

$$h_{\min} \leq h(x^{(n)}) \leq h(x^{(0)}) = H_{\delta_0}(x^{(0)}, x^{(-1)}) \quad \forall n \in \mathbb{N} \tag{15}$$

for all $x^{(n)}$. Due to the coercivity of h, the set of all such x satisfying Equation (15) is bounded (as with $\|x\| \to \infty$, we also have $h(x) \to \infty$, contradicting the bounds in Equation (15)). The claim follows from the Bolzano-Weierstrass theorem.

Finally, we show that each $x^* := \lim_{j\to\infty} x^{(n_j)}$ is a critical point of h. Consider

$$\xi^{(j)} := \underbrace{\frac{x^{(n_j)} - x^{(n_j+1)}}{\alpha_{n_j}} - \nabla f(x^{(n_j)}) + \frac{\beta_{n_j}}{\alpha_{n_j}}(x^{(n_j)} - x^{(n_j-1)})}_{\in\partial g(x^{(n_j+1)})} + \nabla f(x^{(n_j+1)});$$

then, the sequence $(x^{(n_j)}, \xi^{(j)}) \in \operatorname{graph}(\partial h) := \{(x,\xi) \in \mathbb{R}^n \times \mathbb{R}^n | \xi \in \partial h(x)\}$. Using Lemma 8 and the L-Lipschitz continuity of ∇f, it follows

$$\|\xi^{(j)} - 0\|_2 \leq \frac{1}{\alpha_{n_j}}\Delta_{n_j+1} + \frac{\beta_{n_j}}{\alpha_{n_j}}\Delta_{n_j} + \underbrace{\|\nabla f(x^{(n_j+1)}) - \nabla f(x^{(n_j)})\|_2}_{\leq L\|x^{(n_j+1)} - x^{(n_j)}\|_2 = L\Delta_{n_j+1}} \to 0.$$

By Lemma 1, $(x^*, 0) \in \operatorname{graph}(\partial h)$ implies that x^* is a critical point of h. The last statement, i.e. $\lim_{j\to\infty} h(x^{(n_j)}) = h(x^*)$, follows from the following considerations: First, as f is in C^1 it follows $\lim_{j\to\infty} f(x^{(n_j)}) = f(x^*)$. For g, in contrast, we use the lower semicontinuity in combination with $\lim_{j\to\infty} \xi^{(j)} = 0$ and the subadditivity of lim sup which becomes an equality if one of the two summed sequences converges:

$$\limsup_{j\to\infty} g(x^{(n_j)}) = \limsup_{j\to\infty} g(x^{(n_j)}) + (\xi^{(j)})^T(x^* - x^{(n_j)}) \leq g(x^*) \leq \liminf_{j\to\infty} g(x^{(n_j)}).$$

The middle inequality follows from the convexity of g. Overall, we showed $\lim_{j\to\infty} h(x^{(n_j)}) = h(x^*)$. ∎

Now, we can show that H_{δ_n} satisfies (H3):

LEMMA 10. *H_{δ_n} satisfies (H3), i.e. there exists a subsequence $(x^{(n_j)})_{n\in\mathbb{N}}$ such that*

$$z^{(n_j)} \to \tilde{z} = (\tilde{x}, \tilde{x}) \quad \text{and} \quad H(z^{(n_j)}) \to H(\tilde{z}), \quad j \to \infty.$$

Proof. From Lemmata 8 and 9 we know that $\Delta_n \to 0$ and that there exists a subsequence $(x^{(n_j)})_{j\in\mathbb{N}}$ such that $h(x^{(n_j)}) \to h(x^*)$ with $x^* := \lim_{j\to\infty} x^{(n_j)}$. As the term $\delta_n\|x^{(n_j)} - x^{(n_j-1)}\|_2$ is continuous in both $x^{(n_j+1)}$ and $x^{(n_j)}$, we have

$$\lim_{j\to\infty} H_{\delta_n}(x^{(n_j)}, x^{(n_j-1)}) = \lim_{j\to\infty} h(x^{(n_j)}) + \underbrace{\delta_n\|x^{(n_j)} - x^{(n_j-1)}\|_2}_{\to 0} = H(x^*, x^*) = h(x^*).$$ ∎

The above lemma concludes that the defined H_{δ_n} satisfies (H1) - (H3). At this point, Ochs et al. use an abstract convergence result is based on [4]. We first introduce the required mathematical background, i.e. the Kurdyka-Lojasiewicz property (introduced in [12] and [11] and our definition based on [3]):

DEFINITION 6. *A function $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ has the Kurdyka-Lojasiewicz property at point $z^* \in dom(\partial H)$ there exist $\eta \in (0, \infty]$, a neighborhood U of z^*, and a continuous concave function $\phi : [0, \eta) \to \mathbb{R}_+$ such that*

- *$\phi(0) = 0$,*
- *$\phi \in C^1((0, \eta))$,*
- *for all $s \in (0, \eta)$, $\phi'(s) > 0$,*
- *and for all $z \in U \cap \{z \in \mathbb{R}^n \times \mathbb{R}^n | H(z^*) < H(z) < H(z^*) + \eta\}$ the Kurdyka-Lojasiewicz inequality holds:*

$$\phi'(H(z) - H(z^*)) \inf_{\hat{z} \in \partial H(z)} \|\hat{z}\|_2 \geq 1. \tag{16}$$

Here, $\inf_{\hat{z} \in \partial H(z)} \|\hat{z}\|_2$ represents the distance of 0 to the nearest element in $\partial H(z)$. The set

$$\begin{aligned} &U \cap \{z \in \mathbb{R}^n \times \mathbb{R}^n | H(z^*) < H(z) < H(z^*) + \eta\} \\ =&\{z \in \mathbb{R}^n \times \mathbb{R}^n | \|z - z^*\|_2 < \delta, H(z^*) < H(z) < H(z^*) + \eta\} \end{aligned}$$

for some δ is also called the strict local upper level set [10].

It is difficult to find appropriate characterizations of functions satisfying the Kurdyka-Lojasiewicz property relevant to our discussion and the applications presented in Section 4. In [3] Attouch et al. show that all proper lower semicontinuous functions $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ satisfy the Kurdyka-Lojasiewicz property for all non-stationary point. In particular, they prove the existence of a $c > 0$ such that

$$\inf_{\hat{z} \in \partial H(z)} \|\hat{z}\|_2 \geq c$$

for all $z \in B_{\frac{c}{2}}(z^*) \cap \{z \in \mathbb{R}^n \times \mathbb{R}^n | H(z^*) < H(z) < H(z^*) + \eta\}$ where $z^* \in \text{dom}(\partial H)$ is a non-critical point and $B_{\frac{c}{2}}(z^*)$ the ball of radius $\frac{c}{2}$ around z^*. Then, $\phi(s) := c^{-1}s$ is a suitable concave function for Definition 6. As consequence, Definition 6 is usually extended as follows:

DEFINITION 7. *A proper lower semicontinuous function $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ satisfying the Kurdyka-Lojasiewicz property for each $z \in dom(\partial H)$ (including critical points) is called a Kurdyka-Lojasiewicz-function, i.e. a KL-function.*

Following [10], a simple interpretation of KL-functions can be obtained when assuming H to be continuously differentiable. Then, Equation (16) becomes

$$\phi'(H(z) - H(z^*)) \inf_{\hat{z} \in \partial H(z)} \|\hat{z}\|_2 = \phi'(H(z) - H(z^*)) \|\nabla H(z)\|_2 \geq 1$$

In other words, this means that whenever H is flat around a critical point z^*, ϕ has to be steep in order to compensate $\|\nabla H(z)\|_2$ being close to 0. Following Xu and Yin [22], we briefly discuss two important classes of functions satisfying the Kurdyka-Lojasiewicz property for any $z \in \text{dom}(\partial H)$:

- real analytic functions, i.e. infinitely differentiable functions $H : \mathbb{R} \to \mathbb{R}$ such that for each $\hat{z} \in \mathbb{R}$, the Taylor series around $\hat{z}$ converges to H; in particular, polynomials, the exponential function, the natural logarithm, trigonometric functions and all sums, products and compositions thereof are analytic;
- and semialgebraic functions, i.e. functions $H : \mathbb{R}^n \to \mathbb{R}$ where the associated graph(H) is a semialgebraic set (we refer to [9] for details); in particular, polynomials (i.e. polynomial in all coordinates), rational functions (i.e. fractions of polynomials where the denominator is non-zero) and compositions thereof.

Based on the Kurdyka-Lojasiewicz property, we follow Ochs et al. and give an abstract convergence result which can in turn be applied to H_{δ_n} to conclude the convergence analysis of Algorithm 4. The abstract convergence result is split into the following lemma, corollary and theorem:

LEMMA 11. *Let $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a proper lower semicontinuous function satisfying the Kurdyka-Lojasiewicz property at some point $z^* = (x^*, x^*) \in \mathbb{R}^n \times \mathbb{R}^n$. Let*

- *U, η, $\phi : [0, \eta) \to \mathbb{R}_+$ be the elements from Definition 6 for the point z^*;*
- *$\sigma, \rho > 0$ such that $B_\sigma(z^*) \subset U$ and $\rho \in (0, \sigma)$;*
- *$(z^{(n)})_{n\in\mathbb{N}} = (x^{(n)}, x^{(n-1)})_{n\in\mathbb{N}}$ be a sequence satisfying Conditions (H1) - (H3).*

Furthermore, we make the following assumptions:

(A1) for each $n \in \mathbb{N}$, the sequence $(z^{(n)})_{n\in\mathbb{N}}$ satisfies $H(z^{(n+1)}), H(z^{(n+2)}) \geq H(z^*)$ and

$$z^{(n)} \in B_\rho(z^*) \quad \Rightarrow \quad z^{(n+1)} \in B_\sigma(z^*)$$

(A2) $z^{(0)} = (x^{(0)}, x^{(-1)})$ satisfies $H(z^*) \leq H(z^{(0)}) \leq H(z^*) + \eta$ and

$$\|x^* - x^{(0)}\|_2 + \sqrt{\frac{H(z^{(0)}) - H(z^*)}{a}} + \frac{b}{a}\phi(H(z^{(0)}) - H(z^*)) < \frac{\rho}{2}.$$

where a and b are from Definition 5. Then, for each $n \in \mathbb{N}$, $z^{(n)} \in B_\rho(z^*)$ and

$$\sum_{i=0}^{\infty} \Delta_i < \infty \quad \text{and} \quad H(z^{(n)}) \to H(z^*)$$

and $z^{(n)} \to \bar{z} = (\bar{x}, \bar{x}) \in B_\sigma(z^*)$ with $H(\bar{z}) = H(z^*)$ and $0 \in \partial H(\bar{z})$.

Before presenting the proof of Lemma 11 we briefly recapitulate. Therefore, we consider Condition (H3), i.e. there exists a subsequence $(z^{(n_j)})_{n\in\mathbb{N}}$ such that

$$z^{(n_j)} \to \tilde{z} \quad \text{and} \quad H(z^{(n_j)}) \to H(\tilde{z}).$$

Note the notional difference between Condition (H3) and Lemma 11, i.e. $\bar{z}$ and $\tilde{z}$. In the following proof, Condition (H3) is crucial to show that $0 \in \partial H(\bar{z})$ and $H(\bar{z}) = H(\tilde{z})$. Letting $z^* = \tilde{z}$, i.e. letting z^* in Lemma 11 coincide with the cluster point of Condition (H3), we can subsequently proof convergence in Theorem 9.

Proof. We first note that due to Condition (H1), $(H(z^{(n)}))_{n\in\mathbb{N}}$ is non-increasing. Then, with Assumptions (A1) and (A2) it follows that

$$\begin{aligned} H(z^{(n+1)}) \geq H(z^*) \quad &\Rightarrow \quad H(z^{(n+1)}) - H(z^*) \geq 0 \\ H(z^{(n+1)}) \leq H(z^{(0)}) \leq H(z^*) + \eta \quad &\Rightarrow \quad H(z^{(n+1)}) - H(z^*) \leq \eta \end{aligned}$$

such that $\phi(H(z^{(n+1)}) - H(z^*))$ is well-defined.

The next paragraphs are concerned with proving the following statements: for each $n \in \mathbb{N}$

$$z^{(n)} \in B_\rho(z^*) \tag{17}$$

and

$$\sum_{i=1}^{n} \Delta_i \leq \frac{1}{2}(\Delta_0 - \Delta_n) + \frac{b}{a}[\phi(H(z^{(1)}) - H(z^*)) - \phi(H(z^{(n+1)}) - H(z^*))]. \tag{18}$$

We first define

$$D_n^\phi := \phi(H(z^{(n)}) - H(z^*)) - \phi(H(z^{(n+1)}) - H(z^*))$$

for convenience (and conformity with Ochs et al.). Then we show that for $H(z^{(n)}) < H(z^*) + \eta$ and $z^{(n)} \in B_\rho(z^*)$ it holds

$$2\Delta_n \leq \frac{b}{a} D_n^\phi + \frac{1}{2}(\Delta_n + \Delta_{n+1}). \tag{19}$$

Let us first consider the case $\Delta_n = 0$: From Condition (H1) we know $H(z^{(n+1)}) \leq H(z^{(n)})$ and by Assumption (A1) we have $H(z^{(n+1)}), H(z^{(n)}) \geq H(z^*)$ such that with $\phi'(s) > 0$ for $s \in [0, \eta)$ it follows

$$\begin{aligned} & \phi(H(z^{(n)}) - H(z^*)) \geq \phi(H(z^{(n+1)}) - H(z^*)) \\ \Leftrightarrow \quad & D_n^\phi = \phi(H(z^{(n)}) - H(z^*)) - \phi(H(z^{(n+1)}) - H(z^*)) \geq 0. \end{aligned}$$

Let us assume the case $\Delta_n \neq 0$. Then, with Condition (H1) and $\gamma_n \geq c_2 > 0$ it follows $H(z^{(n+1)}) < H(z^{(n)})$ and by Assumption (A1) we have $H(z^{(n+1)}) \geq H(z^*)$. We consider $w^{(n)} \in \partial H(z^{(n)})$ as in Condition (H2) and the KL-inequality

$$\underbrace{\phi'(H(z^{(n)}) - H(z^*))}_{>0} \|w^{(n)}\|_2 \geq \phi'(H(z^{(n)}) - H(z^*)) \inf_{\hat{z} \in \partial H(z^{(n)})} \|\hat{z}\|_2 \geq 1,$$

shows that $w^{(n)} \neq 0$ and, thus, $\Delta_n - \Delta_{n+1} > 0$. Then it follows that

$$\phi'(H(z^{(n)}) - H(z^*)) \geq \frac{1}{\|w^{(n)}\|_2} \overset{(H2)}{\geq} \frac{2}{b(\Delta_{n-1} + \Delta_n)}. \tag{20}$$

As ϕ is concave, it holds

$$\begin{aligned} \phi(H(z^{(n+1)}) - H(z^*)) \leq & \phi(H(z^{(n)}) - H(z^*)) \\ & + \phi'(H(z^{(n)}) - H(z^*))(H(z^{(n+1)}) - H(z^*) - H(z^{(n)}) + H(z^*)) \end{aligned}$$

which follows from the first-order condition for concave functions. Therefore, we can write

$$D_n^\phi \geq \phi'(H(z^{(n)}) - H(z^*))(H(z^{(n+1)}) - H(z^{(n)})) \geq \phi'(H(z^{(n)}) - H(z^*)) a \Delta_n^2 \tag{21}$$

where we used Condition (H1). Combining Equations (20) and (21) we get

$$\frac{b}{2a} D_n^\phi (\Delta_{n-1} + \Delta_n) \geq \Delta_n^2.$$

Then, the claim follows from the following observation:

$$\begin{aligned} 0 \leq p^2 + q^2 \quad & \Leftrightarrow \quad 2pq \leq (p+q)^2 = p^2 + 2pq + q^2 \\ & \Leftrightarrow \quad 2\sqrt{pq} \leq p + q; \end{aligned}$$

in particular, we have

$$2\Delta_n \leq 2(\underbrace{\frac{b}{a} D_n^\phi}_{=:p} \underbrace{\frac{1}{2}(\Delta_{n-1} + \Delta_n)}_{=:q})^{\frac{1}{2}} \leq \frac{b}{a} D_n^\phi + \frac{1}{2}(\Delta_n + \Delta_{n+1}).$$

Obviously, Equation (19) does only hold for $z^{(n)} \in B_\rho(z^*)$. However, by Assumption (A1) we can only expect $z^{(n)} \in B_\sigma(z^*)$ with $\rho \in (0, \sigma)$ to hold true. Nevertheless, we can show Equations (17) and (18) by induction over n. In the induction base, i.e. $n = 1$, from Assumption (A2) we know $z^{(0)} \in B_\rho(z^*)$ and from Assumption (A1) we have $z^{(1)} \in B_\rho(z^*)$. Furthermore, $H(z^{(2)}), H(z^{(1)}) \geq H(z^*)$. Using Condition (H1), we can write

$$H(z^{(2)}) + a\Delta_1^2 \leq H(z^{(1)}) \quad \Leftrightarrow \quad \Delta_1 \leq \sqrt{\frac{H(z^{(1)}) - H(z^{(2)})}{a}}$$

and with $H(z^{(2)}) \geq H(z^*)$, $H(z^{(1)}) \leq H(z^{(0)})$ it follows

$$\Delta_1 \leq \sqrt{\frac{H(z^{(1)}) - H(z^{(2)})}{a}} \leq \sqrt{\frac{H(z^{(0)}) - H(z^*)}{a}} \tag{22}$$

Using the inequality of Assumption (A2), we get

$$\|x^* - x^{(1)}\|_2 \leq \|x^* - x^{(0)}\|_2 + \underbrace{\|x^{(1)} - x^{(0)}\|_2}_{=\Delta_1} \leq \|x^{(0)} - x^*\|_2 + \sqrt{\frac{H(z^{(0)}) - H(z^*)}{a}} < \frac{\rho}{2}.$$

It follows that $z^{(1)} = (x^{(1)}, x^{(0)}) \in B_\rho(z^*)$. Applying Equation (19) (for $n = 1$) implies

$$2\Delta_1 \leq \frac{b}{a} D_n^\phi + \frac{1}{2}(\Delta_0 + \Delta_1) \quad \Leftrightarrow \quad \Delta_1 \leq \frac{b}{a} D_n^\phi + \frac{1}{2}(\Delta_0 + \Delta_1) - \Delta_1$$

which is Equation (18). In the induction step, we assume that Equations (17) and (18) hold for some $n \geq 1$ (i.e. the induction hypothesis). Then we write

$$\begin{aligned}
\|z^* - z^{(n+1)}\|_2 &\leq \|x^* - x^{(n+1)}\|_2 + \|x^* - x^{(n)}\|_2 \\
&\leq \|x^* - x^{(n)}\|_2 + \Delta_{n+1} + \|x^* - x^{(n-1)}\|_2 + \Delta_n \leq \dots \\
&\leq 2\|x^* - x^{(0)}\|_2 + 2\sum_{i=1}^{n} \Delta_i + \Delta_{n+1} \\
&\leq 2\|x^* - x^{(0)}\|_2 + (\Delta_0 - \Delta_n) + \Delta_{n+1} \\
&+ 2\frac{b}{a}[\phi(H(z^{(1)}) - H(z^*)) - \phi(H(z^{(n+1)}) - H(z^*))] \\
&\leq 2\|x^* - x^{(0)}\|_2 + \Delta_0 + \Delta_{n+1} + 2\frac{b}{a}\phi(H(z^{(0)}) - H(z^*)).
\end{aligned}$$

Using the same argument as in Equation (22), we use

$$\Delta_{n+1} \leq \sqrt{\frac{h(z^{(n+1)}) - H(z^{(n+2)})}{a}} \leq \sqrt{\frac{H(z^{(0)}) - H(z^*)}{a}}$$

and the Assumption (A2), to deduce

$$\|z^* - z^{(n+1)}\|_2 \leq 2\|x^* - x^{(0)}\|_2 + \Delta_0 + \sqrt{\frac{H(z^{(0)}) - H(z^*)}{a}} + 2\frac{b}{a}\phi(H(z^{(0)} - H(z^*)) < \rho;$$

which shows that $z^{(n+1)} \in B_\rho(z^*)$. To show Equation (18) we first note that

$$2\Delta_{n+1} \leq \frac{b}{a} D_{n+1}^\phi + \frac{1}{2}(\Delta_n + \Delta_{n+1}) \quad \Leftrightarrow \quad \Delta_{n+1} \leq \frac{b}{a} D_{n+1}^\phi + \frac{1}{2}(\Delta_n - \Delta_{n+1})$$

which follows from Equation (19). Then, adding this to Equation (18) we have

$$\begin{aligned}
\sum_{i=1}^{n} \Delta_i + \Delta_{n+1} \leq& \frac{1}{2}(\Delta_0 - \Delta_n) + \frac{b}{a}[\phi(H(z^{(1)}) - H(z^*)) - \phi(H^{(n+1)}) - H(z^*))] \\
&+ \frac{1}{2}(\Delta_n - \Delta_{n+1}) + \frac{b}{a}[\phi(H(z^{(n+1)}) - H(z^*)) - \phi(H(z^{(n+2)}) - H(z^*))]
\end{aligned}$$

which implies Equation (18).

To show $\sum_{i=0}^{\infty} \Delta_i < \infty$ we use Equation (18) to get

$$\begin{aligned}
\sum_{i=1}^{\infty} \Delta_i &= \lim_{n\to\infty} \sum_{i=1}^{n} \Delta_i \leq \lim_{n\to\infty} \frac{1}{2}(\Delta_0 - \Delta_n) + \frac{b}{a}[\phi(H(z^{(1)}) - H(z^*)) - \phi(H(z^{(n+1)}) - H(z^*))] \\
&\leq \frac{1}{2}\Delta_0 + \frac{b}{a}\phi(H(z^{(1)}) - H(z^*)) < \infty.
\end{aligned}$$

Therefore, $z^{(n)} \to \bar{z}$ and from Condition (H2) it follows that $w^{(n)} \to 0$. Furthermore, $H(z^{(n)})$ converges to some $H(\bar{z}) =: \xi \geq H(z^*)$ (as $H(z^{(n)}) \geq H(z^*)$ for all $n \in \mathbb{N}$). It remains to show that $\xi \leq H(z^*)$. For the sake of the argument we assume $\xi > H(z^*)$. Then, the KL-inequality gives

$$\phi'(\xi - H(z^*))\|w^{(n)}\|_2 \geq \phi'(\xi - H(z^*)) \inf_{\hat{z} \in \partial H(\bar{z})} \|\hat{z}\|_2 \geq 1$$

which contradicts the finding that $w^{(n)} \to 0$ and it follows $\xi = H(\bar{z}) \leq H(z^*)$. ∎

COROLLARY 1. *Given η, σ and ρ as in Lemma 11, Assumption (A1) can be replaced by (A1') for each $n \in \mathbb{N}$, the sequence $(z^{(n)})_{n\in\mathbb{N}}$ satisfies $H(z^{(n)}) \geq H(z^*)$ and it holds*

$$\eta < a\left(\frac{\sigma-\rho}{2}\right)^2. \tag{23}$$

Proof. Using Condition (H1) we have

$$\Delta_{n+1}^2 \leq \frac{H(z^{(n+1)}) - H(z^{(n+2)})}{a} \leq \frac{\eta}{a} < \left(\frac{\sigma-\rho}{2}\right)^2.$$

For $z^{(n)} \in B_\rho(z^*)$, using the triangle inequality we get

$$\begin{aligned}\|z^{(n+1)} - z^*\|_2 &\leq \|z^{(n+1)} - z^{(n)}\|_2 + \|z^{(n)} - z^*\|_2 \\ &< \|x^{(n+1)} - x^{(n)}\|_2 + \|x^{(n)} - x^{(n-1)}\|_2 + \rho \\ &\leq \frac{\sigma-\rho}{2} + \frac{\sigma-\rho}{2} + \rho = \sigma.\end{aligned}$$

□

Note that Ochs et al. originally use

$$\eta < a(\sigma-\rho)^2$$

instead of Equation (23). However, this would not work in the above proof. We assume that this was a minor mistake when adapting [4, Cor. 2.8] to the new setting.

THEOREM 1. *Let $H : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a proper lower semicontinuous function and $(z^{(n)})_{n\in\mathbb{N}} = (x^{(n)}, x^{(n-1)})_{n\in\mathbb{N}}$ be a sequence satisfying Conditions (H1) - (H3). Furthermore, let H satisfy the Kurdyka-Lojasiewicz property at the cluster point $\tilde{z}$ specified in Condition (H3). Then, the sequence $(x^{(n)})_{n\in\mathbb{N}}$ converges to $\tilde{x}$ such that $\tilde{z} = (\tilde{x}, \tilde{x})$ is a critical point of H.*

Proof. Following Condition (H3), there exists a subsequence $(z^{(n_j)})_{j\in\mathbb{N}}$ with $z^{(n_j)} \to \tilde{z}$. Because of the non-increasingness of H by Condition (H1) this implies that $H(z^{(n)}) \to H(\tilde{z})$ as $n \to \infty$ and $H(z^{(n)}) \geq H(\tilde{z})$. Due to the Kurdyka-Lojasiewicz property of H at point $\tilde{z}$, there exist $\eta \in (0, \infty]$, a neighborhood U around $\tilde{z}$ and a continuous, concave function $\phi : [0, \eta) \to \mathbb{R}_+$ satisfying the conditions in Definition 6. Let $\sigma, \rho > 0$ such that $B_\sigma(\tilde{z}) \subset U$, $\rho \in (0, \sigma)$ and

$$\eta < a\left(\frac{\sigma-\rho}{2}\right)^2. \tag{24}$$

In the worst case, η can be decreased until Equation (24) is satisfied. Then, there exist $n_0 \in \mathbb{N}$ such that $H(\tilde{z}) \leq H(z^{(n)}) < H(\tilde{z}) + \eta$ for $n \geq n_0$ and

$$\|\tilde{x} - x^{(n_0)}\|_2 + \sqrt{\frac{H(z^{(n_0)}) - H(\tilde{z})}{a}} + \frac{b}{a}\phi(H(z^{(n_0)}) - H(\tilde{z})) < \frac{\rho}{2}. \tag{25}$$

This can be seen as ϕ is required to be continuous and $H(z^{(n)}) \to H(\tilde{z})$ such that $\phi(H(z^{(n_0)}) - H(\tilde{z}))$ can be made arbitrarily small. Similar arguments can be applied to the remining terms of Equation (25). As result, both Assumptions (A1') and (A2) are fulfilled and the claim follows by Lemma 11 and Corollary 1. □

3.2.1. Convergence Rate. Ochs et al. first relate the Δ_n's, which were an important part of our discussion so far, to the proximal residual:

DEFINITION 8. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable and $g : dom(g) \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be proper closed convex. Then the proximal residual is defined as*

$$r(x) := x - prox_g(x - \nabla f(x)).$$

Obviously, the residual is directly related to the first-order condition as derived from Lemma 2 in the following sense:

$$\begin{aligned} r(x^*) = 0 \quad &\Leftrightarrow \quad x^* = \operatorname{prox}_g(x^* - \nabla f(x^*)) \\ &\Leftrightarrow \quad x^* - \nabla f(x^*) \in x^* + \partial g(x^*) \\ &\Leftrightarrow \quad -\nabla f(x^*) \in \partial g(x^*) \\ &\Leftrightarrow \quad 0 \in \partial g(x^*) + \nabla f(x^*). \end{aligned}$$

The relation between the residual $r(x)$ and the Δ_n's is then established by the following Lemma.

LEMMA 12. *Let $r(x)$ be the residual as given by Definition 8. Then, it holds*

$$\sum_{i=1}^{n} \|r(x^{(i)})\|_2 \leq \frac{2}{c_1} \sum_{i=0}^{n} \Delta_{i+1}$$

where $c_1 > 0$.

Proof. We omit the proof for brevity as several properties of the proximal mapping $\operatorname{prox}_{\alpha g}$ would be required. The proof can be found in [15]. ∎

To discuss the convergence rate, we first introduce the following definitions:

DEFINITION 9. *The residual $r(x)$ being as in Definition 8, we define the measurable error as*

$$\mu_{meas}^{(n)} := \min_{0 \leq i \leq n} \Delta_i^2$$

and the true error as

$$\mu_{true}^{(n)} := \min_{0 \leq i \leq n} \|r(x^{(i)})\|_2^2.$$

Note that the chosen names are to be taken literally: i.e. μ_{meas} is measurable during the execution of Algorithm 4 in terms of the Δ_n's; μ_{true} cannot be measured directly and represents the true error. Therefore, we intend to bound the true error by the measurable error to get a grasp of how close the algorithm is to a solution. This, in turn, may also be used as termination criterion for Algorithm 4.

THEOREM 2. *Let μ_{meas} and μ_{true} as in Definition 9. Then Algorithm 4 guarantees that for $n \in \mathbb{N}$:*

$$\mu_{true}^{(n)} \leq \frac{2}{c_1} \mu_{meas}^{(n)} \quad \text{and} \quad \mu_{meas}^{(n)} \leq \frac{h(x^{(0)}) - h_{\min}}{c_1(n+1)}$$

where $h_{\min}$ is a lower bound of $h(x)$ and $x^{(0)}$ an appropriate initial value.

Proof. We merely show the second inequality as it is of more practical relevance. Therefore, we consider Condition (H1), in particular as proved in Lemma 6:

$$H_{\delta_{n+1}}(x^{(n+1)}, x^{(n)}) \leq H_{\delta_n}(x^{(n)}, x^{(n-1)}) - \gamma_n \Delta_n^2.$$

Summing both sides for $i = 0, \dots, n$ we get

$$\sum_{i=0}^{n} H_{\delta_{i+1}}(x^{(i+1)}, x^{(i)}) \leq \sum_{i=0}^{n} H_{\delta_i}(x^{(i)}, x^{(i-1)}) - \sum_{i=0}^{n} \gamma_i \Delta_i^2$$

$$\Leftrightarrow \quad h(x^{(n+1)}) + \delta_{n+1}\Delta_{n+1}^2 \leq h(x^{(0)}) - \sum_{i=0}^{n} \gamma_i \Delta_i^2 \leq h(x^{(0)}) - \min_{0 \leq i \leq n} \gamma_i \Delta_i^2.$$

Rearrangement, using the lower bound $h_{\min}$ of $h(x)$ and the fact that $\delta_{n+1} \geq 0$ yields

$$h_{\min} \leq h(x^{(0)}) - \min_{0 \leq i \leq n} \gamma_i \Delta_i^2 \leq h(x^{(0)}) - (n+1) c_1 \mu_{\text{meas}}^{(n)}$$

$$\Leftrightarrow \quad \mu_{\text{meas}} \leq \frac{h(x^{(0)}) - h_{\min}}{c_2 (n+1)}$$

where we used $\gamma_n \geq c_2$ as required by Algorithm 4. ∎

Theorem 2 implies

$$\mu_{\text{true}}^{(n)} \leq \frac{2(h(x^{(0)}) - h_{\min})}{c_1 c_2 (n+1)}.$$

As $h(x^{(0)})$, $h_{\min}$, c_1 and c_2 can be considered constant, this is equivalent to stating a

$$\mu_{\text{true}}^{(n)} \in \mathcal{O}(c\frac{1}{n+1}) = \mathcal{O}(\frac{1}{n})$$

convergence rate of the residual in the squared L_2-norm. Overall, we deduce a convergence rate of $\mathcal{O}(\frac{1}{\sqrt{n}})$ for the residual in L_2-norm.

3.3. Implementation. In this section, we briefly discuss two important details concerning an actual implementation of Algorithm 4: Initializing α_0 and β_0 as well as choosing α_n and β_n for $n \geq 1$.

3.3.1. Initialization – Determining α_0 and β_0. We revisit Lemmata 4 and 5 providing the following inequalities:

$$\alpha_0 \leq \frac{2(1-\beta_0)}{(L_0 + 2c_2)} < \frac{2(1-\beta_0)}{L_0} \tag{26}$$

$$\beta_0 \leq \frac{b_0 - 1}{b_0 - \frac{1}{2}} \quad \text{with} \quad b_0 := \frac{\delta_{-1} + \frac{L_n}{2}}{c_2 + \frac{L_n}{2}}.$$

Obviously, this recursion can be broken when expecting β_0 to be given as parameter (as it is more difficult to make a good guess for δ_{-1} than β_0). Then, it remains to find L_0 in order to deduce α_0 as in Equation (26). Algorithm 4 depicts a simple backtracking scheme to estimate L_0 given a user provided estimate L_{-1}. Instead, we propose to estimate L_0 directly around $x^{(0)}$:

$$\|\nabla f(x^{(0)}) - \nabla f(\hat{x})\|_2 \leq L_0 \|x^{(0)} - \hat{x}\|_2$$

$$\Leftrightarrow \quad \frac{\|\nabla f(x^{(0)}) - \nabla f(\hat{x})\|_2}{\|x^{(0)} - \hat{x}\|_2} \leq L_0 \tag{27}$$

for an appropriate $\hat{x}$ which can be chosen according to

$$\hat{x} = \text{prox}_g(x^{(0)} - \nabla f(x^{(0)})).$$

Given β_0 and L_0, α_0 can be determined using the inequality in Equation (26) subject to the requirement

$$\delta_0 \geq \gamma_0 \geq c_2.$$

The remaining non-determinism can be resolved by considering a fixed number of discrete steps

$$\alpha_0^{(k)} := a_0 - k\frac{a_0 - c_1}{K} \quad \text{with} \quad a_0 := \frac{2(1-\beta_0)}{(L_0 + 2c_2)},\ K \gg 100 \quad \text{and} \quad k = 1, \dots, K.$$

until an $\alpha_0^{(k)}$ is found satisfying the requirements.

3.3.2. Iteration – Finding α_n and β_n. We need to resolve the nondeterminism in Algorithm 4, Lines 12 and 13. Given the current estimate of L_n, see Line 10, we follow a similar strategy as used for initialization. In particular, we evaluate the requirement

$$\delta_n \geq \gamma_n \geq c_2 \quad \text{and} \quad \delta_n \leq \delta_{n-1}$$

for discrete steps

$$\beta_n^{(j)} := \frac{b_n - 1}{b_n - \frac{1}{2}} - \frac{j}{J}\frac{b_n - 1}{b_n - \frac{1}{2}} \quad \text{with} \quad b_n := \frac{\delta_{n-1} + \frac{L_n}{2}}{c_2 + \frac{L_n}{2}},\ J \gg 100 \quad \text{and} \quad j = 0, \ldots, J,$$
$$\alpha_n^{(k)} := a_n - k\frac{a_n - c_1}{K} \quad \text{with} \quad a_n := \frac{2(1-\beta_n)}{(L_n + 2c_2)},\ K \gg 100 \quad \text{and} \quad k = 1, \ldots, K$$

and stop as soon as the requirements are satisfied. As alternative to Line 8, i.e. using L_{n-1} as starting point for backtracking, we re-estimate L_n using Equation (27) (adapted for $n \geq 0$). The disadvantage of using L_{n-1} is that backtracking guarantees $L_n \geq L_{n-1}$ which is problematic as high L_n induces low α_n for fixed β_n slowing down convergence. Re-estimating L_n, in contrast, also allows $L_n < L_{n-1}$.

4. Image Processing Applications. In this section, we apply Algorithm 4 to the following image processing tasks: denoising and segmentation. Therefore, we consider an image $u^{(0)} : \Omega \to [0, 1]$ where Ω is usually assumed to to be the unit square, i.e. $\Omega := [0, 1]^2$. We assume $u^{(0)}$ to be a grayscale image, however most of the below considerations can be extended to color images. In practice, the image is given by a regular grid of intensity values; we expect a discretization $\tilde{u}^{(0)}$ of $u^{(0)}$ to be represented by $\tilde{u}^{(0)} \in [0, 1]^{n\times m}$ such that $\tilde{\Omega} = \{(i, j) \in \mathbb{N} \times \mathbb{N} | 1 \leq i \leq n, 1 \leq j \leq m\}$.

4.1. Denoising. Ochs et al. use the Field of Experts [19] model to apply Algorithm 4 to image denoising. However, Algorithm 4 can only be used to denoise a given image $u^{(0)}$ if the Field of Experts is already trained, i.e. the used filters have been determined. As can be seen in [7], Algorithm 4 cannot be used to train a Field of Experts. Therefore, we focus on simpler models for image denoising. In particular, we use a simple Markov Random Field model:

$$h(u; u^{(0)}, \lambda) = \int_\Omega \rho_1(u(x) - u^{(0)}(x))dx + \lambda \int_\Omega \rho_2(\|\nabla u(x)\|_2)dx \tag{28}$$

where $\rho_1, \rho_2 : \mathbb{R} \to \mathbb{R}$ are appropriate functions. In the following we investigate the practical implementation of the following variant of the above model:

$$\rho_{1,\text{abs}}(x) = |x| \quad \text{and} \quad \rho_{1,\text{sqr}}(x) = x^2;$$
$$\rho_2(x; \sigma) = \log\left(1 + \frac{x^2}{\sigma^2}\right).$$

where the latter is also referred to as the Lorentzian function. To apply ρ_1 in Equation (28), the work by Combettes and Pesquet [8] and Parikh and Boyd [16] is useful in deriving $\text{prox}_{\alpha\rho_1}$:

LEMMA 13. *For $\rho_{1,abs}(x) = |x|$, the proximal mapping $prox_{\alpha\rho_{1,abs}}$ is given by*

$$prox_{\alpha\rho_{1,abs}}(x) = \begin{cases} x - \alpha & x \geq \alpha \\ 0 & |x| \leq \alpha \\ x + \alpha & x \leq -\alpha \end{cases}$$

As of Equation (28), we additionally need to consider the translation of the argument, i.e.

$$\rho_1(u(x) - u^{(0)}(x)).$$

We use the result as presented in [8]:

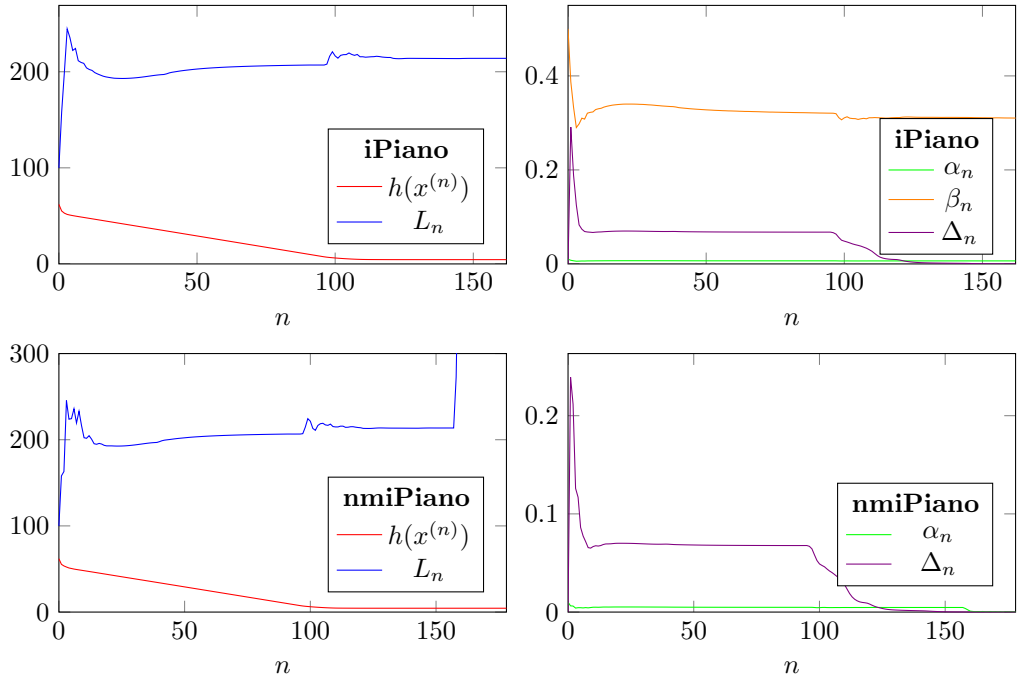

Fig. 1: Convergence of **iPiano**, i.e. Algorithm 4, and **nmiPiano**, i.e. Algorithm 2. Shown is the value of the objective function $h(x(n))$ for each iterate $x(n)$, $n \geq 0$, as well as the corresponding parameters α_n, β_n and L_n. For **nmiPiano** we set $\beta = 0.5$. Furthermore, $\Delta_n := \|x^{(n)} - x^{(n-1)}\|_2$ is shown.

LEMMA 14. *For any $\rho_1 : \mathbb{R}^n \to \mathbb{R}$ and $\rho_{1,trans} := \rho_1(x - x^{(0)})$ with $x^{(0)} \in \mathbb{R}^n$ it holds*

$$prox_{\alpha\rho_{1,trans}}(x) = -x^{(0)} + prox_{\alpha\rho_1}(x + x^{(0)}).$$

Combining Lemmata 13 and 14, $\text{prox}_{\alpha\rho_1}$ can be written as

$$\text{prox}_{\alpha\rho_{1,\text{trans}}}(x) = \max(0, |x| - \alpha) \cdot \text{sign}(x) - x^{(0)}$$

which can easily be implemented efficiently. With $\rho_{1,\text{sqr}}$ and ρ_2 being continuously differentiable, Algorithm 4 can be applied after discretization.

4.1.1. Discretization. Discretization of the Markov Random Field model of Equation (28) is straight forward:

$$h(\tilde{u}; u^{(0)}, \lambda) = \underbrace{\sum_{i=1}^{n}\sum_{j=1}^{m} \rho_1(\tilde{u}_{i,j} - \tilde{u}^{(0)}_{i,j})}_{=g(\tilde{u};u^{(0)})} + \underbrace{\lambda \left[\sum_{i=2}^{n}\sum_{j=2}^{m} \rho_2(\|\tilde{u}_{i,j} - \tilde{u}_{i-1,j}\|_2) + \rho_2(\|\tilde{u}_{i,j} - \tilde{u}_{i,j-1}\|_2)\right]}_{=f(\tilde{u})}.$$

While differentiation is linear, leading to an easy-to-compute gradient ∇f, we use the following lemma to compute $\text{prox}_{\alpha g}$ in practice:

LEMMA 15. *For $x \in \mathbb{R}^{n\times m}$ and*

$$g(x) = \sum_{i=1}^{n}\sum_{j=1}^{m} \rho(x_{i,j})$$

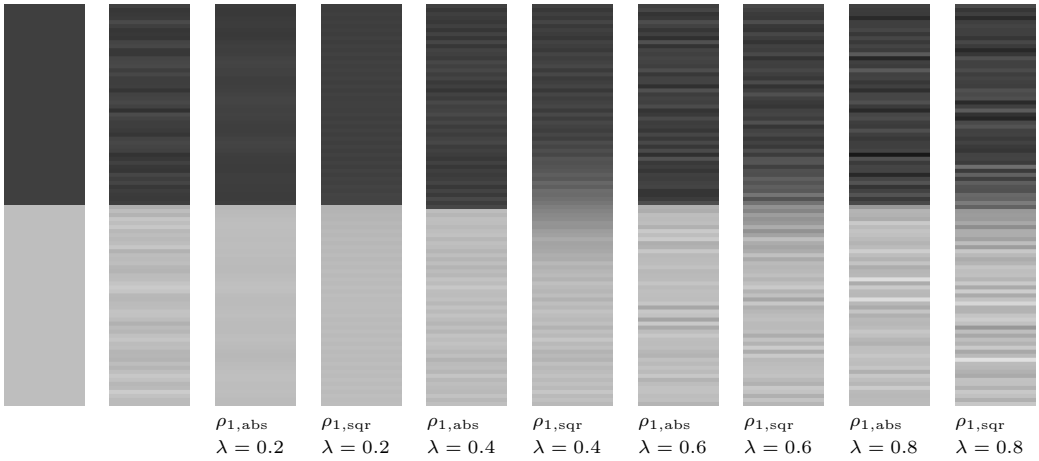

Fig. 2: Simple signal denoising experiment; input signal shown on the left with the perturbed/noisy signal on its right. The result of applying Algorithm 4 to the proposed denoising model using $\rho_{1,\text{abs}}$ and $\rho_{1,\text{sqr}}$ with $\lambda \in \{0.2, 0.4, 0.6, 0.8\}$ is shown.

it holds $prox_{\alpha g} \in \mathbb{R}^{n\times m}$ *and*

$$\big(prox_{\alpha g}\big)_{i,j} = prox_{\alpha\rho}(x_{i,j}).$$

4.1.2. Experiments. In order to discuss convergence as well as the advantage of using $\rho_{1,\text{abs}}$, we first discuss signal denoising (i.e. one-dimensional denoising). Figure 1 shows the convergence of Algorithms 4 and 2 for $\rho_{1,\text{abs}}$. Both algorithms converge to $h(x^*) := 4.4598$, however, Algorithm 4 converges faster, needing only 162 iterations compared to 178 iterations needed for Algrithm 2 to converge. Of course, this may be due to a suboptimal choice of $\beta = 0.5$ for Algorithm 2. On the other hand, this illustrates that automatically adapting both α_n and β_n is beneficial. Figure **??** shows a signal, the corresponding perturbed/noisy signal and results of using Algorithm 4 for denoising using $\rho_{1,\text{abs}}$ and $\rho_{1,\text{sqr}}$ with different values of λ. Gaussian additive noise with standard deviation $\sigma = 0.05$ has been used. The advantage of using $\rho_{1,\text{abs}}$ gets apparent for $\lambda > 0.2$ where the sharp edge is not preserved when using $\rho_{1,\text{sqr}}$.

Results of applying Algorithm 4 to denoise images from the Berkeley Segmentation Dataset [2] are shown in Figure 3. Gaussian additive noise with standard deviation $\sigma = 0.05$ has been used. Again, $\rho_{1,\text{sqr}}$ is not able to preserve edges.

4.2. Segmentation. Following Shen [20] we want to apply Algorithm 4 to image segmentation using a (2-)phase-field. Therefore, Shen considers the reduced Mumford-Shah[1] model [13]:

$$h(O; c_+, c_-, u^{(0)}) = \int_O (u^{(0)}(x) - c_+)^2 dx + \int_{\Omega\setminus O} (u^{(0)}(x) - c_-)^2 dx + \lambda \text{per}(O)$$

where λ is a regularization parameter, c_+ and c_- average intensities and $\text{per}(O)$ is the perimeter of $O \subset \Omega$ (also called the length in [20]):

$$\text{per}(O) := |\chi_O|_{BV(\Omega)},$$

i.e. the total variation of the characteristics function χ_O. Obviously, minimization over the set O is problematic. One of the most popular approaches to this problem may be the work by Chan and Vese [21] who apply the level set framework to minimize over a 3-dimensional curve instead of the set O. Instead, Shen – following Ambrosio and Tortorelli [1] – derives a Γ-convergence formulation of the

[1] In particular, we consider the 2-phase model of the Mumford-Shah model.

Fig. 3: Results of applying Algorithm 4 to image denoising using the proposed model with the noisy image in the top row, $\rho_{1,\text{abs}}$ in the middle row and $\rho_{1,\text{sqr}}$ in the bottom row. Gaussian additive noise with standard deviation $\sigma = 0.05$ has been used. The discrete image has been scaled such that $\tilde{u}_{i,j} \in [0, 1]$.

reduced Mumford-Shah model:

$$\begin{aligned} h_\epsilon(u; c_+, c_-, u^{(0)}, \lambda) = &\int_\Omega \left(9\epsilon \|\nabla u(x)\|_2^2 + \frac{(1 - u(x)^2)^2}{64\epsilon} \right) dx \\ &+ \lambda \int_\Omega \left(\frac{1 + u(x)}{2} \right)^2 (u^{(0)}(x) - c_+)^2 dx \\ &+ \lambda \int_\Omega \left(\frac{1 - u(x)}{2} \right)^2 (u^{(0)}(x) - c_-)^2 dx. \end{aligned} \tag{29}$$

We briefly give the intuition behind this model without covering the necessary mathematical background. Let $u : \Omega \to [-1, 1]$ be called phase-field with the corresponding phase field energy given by

$$f_\epsilon(u) = \int_\Omega \left(9\epsilon \|\nabla u\|_2^2 + \frac{(1 - u^2)^2}{64\epsilon} \right) dx. \tag{30}$$

For $\epsilon > 0$ but close to 0, this energy will force u to be 1 or -1 almost everywhere. Then, the perimeter $\text{per}(O)$ can be approximated by f_ϵ. In this sense Equation (30) represents an approximation to the reduced Mumford-Shah model. Shen proposes an alternating scheme to minimize h_ϵ, i.e. for fixed c_+ and c_- minimize h_ϵ and for fixed u compute

$$c_+ = \frac{\int_\Omega (1 + u(x))^2 u^{(0)}(x) dx}{\int_\Omega (1 + u(x))^2 dx}; \tag{31}$$

$$c_- = \frac{\int_\Omega (1 - u(x))^2 u^{(0)}(x) dx}{\int_\Omega (1 - u(x))^2 dx}. \tag{32}$$

As notation of Equations (29) and (30) suggests, Algorithm 4 can be used to minimize

$$h_\epsilon(u; c_+, c_-, u^{(0)}, \lambda) = f_\epsilon(u) + g_{\text{sqr}}(u; c_+, c_-, u^{(0)}, \lambda)$$

with

$$g_{\text{sqr}}(u; c_+, c_-, u^{(0)}, \lambda) = \lambda \int_\Omega \left(\frac{1 + u(x)}{2} \right)^2 (u^{(0)}(x) - c_+)^2 dx + \lambda \int_\Omega \left(\frac{1 - u(x)}{2} \right)^2 (u^{(0)}(x) - c_-)^2 dx.$$

As the integrands are continuously differentiable Algorithm 4 can easily be applied after discretization.

Before discussing discretization and experiments, we discuss an alternative g comprising absolute terms:

$$g_{\text{abs}}(u; c_+, c_-, u^{(0)}, \lambda) = \lambda \int_\Omega |1 + u(x)|(u^{(0)}(x) - c_+)^2 dx + \lambda \int_\Omega |1 - u(x)|(u^{(0)}(x) - c_-)dx.$$

Equations (31) and (32) can be adapted accordingly. As we will see, the proximal mapping needed after discretization cannot be derived using the rules presented in [8].

4.2.1. Discretization. It remains to discuss the discretization of Equation (29). Given an image $\tilde{u}^{(0)} \in \mathbb{R}^{n\times m}$, we discretize h_ϵ as follows

$$f_\epsilon(\tilde{u}) = \sum_{i=2}^{n}\sum_{j=2}^{m} 9\epsilon\|\tilde{u}_{i,j} - \tilde{u}_{i-1,j}\|_2^2 + \|\tilde{u}_{i,j} - \tilde{u}_{i,j-1}\|_2^2 + \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{1}{64\epsilon}(1 - \tilde{u}_{i,j}^2)^2;$$

$$g_{\text{sqr}}(\tilde{u}; c_+, c_-, \tilde{u}^{(0)}) = \left[\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{1}{2}(1 + \tilde{u}_{i,j})^2(\tilde{u}_{i,j}^{(0)} - c_+)^2 + \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{1}{2}(1 - \tilde{u}_{i,j})^2(\tilde{u}_{i,j}^{(0)} - c_-)^2\right];$$

Then, c_+ and c_- can be computed as

$$c_+ = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(1 + \tilde{u}_{i,j})^2\tilde{u}_{i,j}^{(0)}}{\sum_{i=1}^{n}\sum_{j=1}^{n}(1 + \tilde{u}_{i,j})^2};$$
$$c_- = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(1 - \tilde{u}_{i,j})^2\tilde{u}_{i,j}^{(0)}}{\sum_{i=1}^{n}\sum_{j=1}^{n}(1 - \tilde{u}_{i,j})^2}.$$

Again, for g_{abs} these equations can easily be adapted. Finally, the segmentation is obtained by thresholding $\tilde{u}$, i.e. the foreground and background segments S_f, $S_b \subset \tilde{\Omega}$ are obtained as

$$S_f := \{(i,j) \in \tilde{\Omega} | \tilde{u}_{i,j} > \tau\}, \quad S_b := \tilde{\Omega} \setminus S_f.$$

To practically implement this segmentation model using g_{abs} we first rewrite the above discretization:

$$g_{\text{abs}}(\tilde{u}; c_+, c_-, \tilde{u}^{(0)}) = \left[\sum_{i=1}^{n}\sum_{j=1}^{m}|1 + \tilde{u}_{i,j}|(\tilde{u}_{i,j}^{(0)} - c_+)^2 + |1 - \tilde{u}_{i,j}|(\tilde{u}_{i,j}^{(0)} - c_-)^2\right];$$

then, we need to derive the proximal mapping for

$$\rho(x) := |1 - x|C_+ + |1 + x|C_-.$$

LEMMA 16. *For ρ as defined above, the proximal mapping $prox_{\alpha\rho}$ is given as*

$$prox_{\alpha\rho}(x) = \begin{cases} -\alpha(C_+ + C_-) + x) & x > 1 + \alpha(C_+ + C_-) \\ 1 & x \in [1 + \alpha(C_+ - C_-), 1 + \alpha(C_p + C_-) \\ \alpha(-C_+ + C_-) + x & x \in (-1 + \alpha(C_+ - C_-), 1 + \alpha(C_+ - C_-)) \\ -1 & x \in [-1 - \alpha(C_+ + C_-), -1 + \alpha(C_+ - C_-)] \\ \alpha(C_+ + C_-) + x & x < -1 - \alpha(C_+ + C_-) \end{cases}. \tag{33}$$

Proof. We first rewrite ρ as piecewise linear function:

$$\rho(x) = \begin{cases} 2C_+ + (C_+ + C_-)(x - 1) & x > 1 \\ 2C_- + (C_+ - C_-)(1 + x) & x \in [-1, 1] \\ 2C_- + (C_+ + C_-)(1 + x) & x < -1 \end{cases}.$$

Fig. 4: Segmentation results for five images from the Berkeley Segmentation Dataset [2] for thresholds $\tau = -0.1, -0.2, 0.0, 0.1, 0.2$ and using g_{sqr}; the foreground segment S_f is depicted in white.

The proximal mapping is then defined as

$$\text{prox}_{\alpha\rho}(x) = \alpha\rho(\hat{x}) + \frac{1}{2}(\hat{x} - x)^2.$$

Note that without loss of generality we can assume $\alpha = 1$ as α merely scales C_+ and C_-. Then, using the subdifferential

$$\partial f(x) = \begin{cases} C_+ + C_- & x > 1 \\ [C_+ - C_-, C_+ + C_-] & x = 1 \\ C_+ - C_- & -1 < x < 1 \\ [-(X_+ + C_-), C_+ - C_-] & x = -1 \\ -(C_+ + C_-) & x < -1 \end{cases},$$

we derive the first-order condition

$$\partial f(\hat{x}) + (\hat{x} - x) = 0.$$

We distinguish the following cases:

1) $x > 1$ gives

$$\begin{aligned} C_+ + C_- + \hat{x} - x = 0 \quad &\Leftrightarrow \quad \hat{x} = -(C_+ + C_-) + x > 1 \\ &\Leftrightarrow \quad x > 1 + C_+ + C_- \end{aligned}$$

2) $-1 < x < 1$ gives

$$\begin{aligned} C_+ - C_- + \hat{x} - x = 0 \quad &\Leftrightarrow \quad -1 < \hat{x} = -C_+ + C_- + x < 1 \\ &\Leftrightarrow \quad -1 + C_- - C_- < x < 1 + C_- - C_-. \end{aligned}$$

3) $x < -1$ gives

$$\begin{aligned} -(C_+ + C_-) + \hat{x} - x = 0 \quad &\Leftrightarrow \quad \hat{x} = (C_+ + C_-) + x < -1 \\ &\Leftrightarrow \quad x < -1 - (C_+ + C_-). \end{aligned}$$

Fig. 5: Segmentation results for an image from the Berkeley Segmentation Dataset [2] for thresholds $\tau = -0.2, -0.1, 0.0, 0.1, 0.2$ and using g_{abs}; the foreground segment S_f is depicted in white.

It remains to discuss the following cases:

4) $x = 1 + C_+ + C_-$ and $x = 1 + C_- - C_-$ give $\hat{x} = 1$.

5) $x = -1 + C_- - C_-$ and $x = -1 - (C_+ + C_-)$ give $\hat{x} = -1$.

Overall, Equation (33) follows. ∎

4.2.2. Experiments. We conducted experiments on several color images from the Berkeley Segmentation Dataset [2]; results for several thresholds τ and $\lambda = 100$ using g_{sqr} are shown in Figure 4. Figure 5, in contrast, shows results using g_{abs}. As can be seen, using g_{abs} resolves parts of the artifacts on the lower left border of the "airplane' image. However, this also removes clear contours around the airplane.

5. Conclusion. In this paper, we introduced the iPiano algorithm for minimization of non-convex and non-smooth composite functions. Following Ochs et al. [15], we proved convergence for objectives satisfying the Kurdyka-Lojasiewicz property [12, 11] and discussed the practical implementation of the proposed algorithm. Finally, we discussed the application of iPiano to image denoising and image segmentation. Overall, we showed that iPiano is able to reliably minimize non-smooth and non-convex composite functions frequently used in image processing and computer vision applications.

5.1. Connection to [5]. In [5, Bem. 4.12], Berkels introduces the reader to the forward-backward splitting algorithm where iterates are computed according to

$$x^{(n+1)} = \text{prox}_{\alpha g}(x^{(n)} - \nabla f(x^{(n)})), \tag{34}$$

assuimg that both g and f are proper closed convex and f is continuously differentiable. As mentioned in Section 2, Ochs et al. extend this scheme by adding an inertial force (i.e. a momentum term) and disregarding the requirement of f being proper closed convex. In particular, Equation (5) extends Equation (34) by the inertial force term $\beta_n(x^{(n)} - x^{(n-1)})$.

REFERENCES

[1] L. Ambrosio and V. M. Tortorelli, *Approximation of functional depending on jumps by elliptic functional via t-convergence*, Communications on Pure and Applied Mathematics, 43 (1990), pp. 999–1036.

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, *Contour detection and hierarchical image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 33 (2011), pp. 898–916.

[3] H. Attouch, J. Bolte, P. Redont, and A. Soubeyran, *Proximal alternating minimization and projection methods for nonconvex problems: An approach based on the kurdyka-lojasiewicz inequality*, Math. Oper. Res., 35 (2010), pp. 438–457.

[4] H. Attouch, J. Bolte, and B. F. Svaiter, *Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward-backward splitting, and regularized gauss-seidel methods*, Math. Program., 137 (2013), pp. 91–129.

[5] B. Berkels, *Lecture notes "variationsmethoden in der bildverarbeitung"*, 2015.

[6] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, New York, New York, 2004.

[7] Y. Chen, T. Pock, R. Ranftl, and H. Bischof, *Revisiting loss-specific training of filter-based mrfs for image restoration*, in Pattern Recognition, German Conference on, Saarbrücken, Germany, September 2013, pp. 271–281.

[8] P. L. Combettes and J.-C. Pesquet, *Proximal splitting methods in signal processing*, in Fixed-Point Algorithms for Inverse Problems in Science and Engineering, H. Bauschke, R. Burachik, P. Combettes, V. Elser, D. Luke, and H. E. Wolkowicz, eds., Springer, 2011, pp. 185–212.

[9] M. Coste, *An introduction to semialgebraic geometry*, October 2002.

[10] P. Frankel, G. Garrigos, and J. Peypouquet, *Splitting methods with variable metric for kurdyka-łojasiewicz functions and general convergence rates*, J. Optimization Theory and Applications, 165 (2015), pp. 874–900.

[11] K. Kurdyka, *On gradients of functions definable in o-minimal structures*, Annales de l'institut Fourier, 48 (1998), pp. 769–783.

[12] S. Lojasiewicz, *Sur la gomtrie semi- et sous- analytique*, Annales de l'institut Fourier, 43 (1993), pp. 1575–1595.

[13] D. Mumford and J. Shah, *Optimal approximations by piecewise smooth functions and associated variational problems*, Comm. on Pure and Applied Mathematics, 42 (1989), pp. 577–685.
[14] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, New York, 2 ed., 2006.
[15] P. Ochs, Y. Chen, T. Brox, and T. Pock, *ipiano: Inertial proximal algorithm for nonconvex optimization*, SIAM J. Imaging Sciences, 7 (2014), pp. 1388–1419.
[16] N. Parikh and S. P. Boyd, *Proximal algorithms*, Foundations and Trends in Optimization, 1 (2014), pp. 127–239.
[17] B. T. Polyak, *Some methods of speeding up the convergence of iteration methods*, USSR Computational Mathematics and Mathematical Physics, 4 (1964), pp. 1–17.
[18] R. T. Rockafellar, *Convex analysis*, Princeton Mathematical Series, Princeton University Press, Princeton, New Jersey, 1970.
[19] S. Roth and M. J. Black, *Fields of experts*, International Journal of Computer Vision, 82 (2009), pp. 205–229.
[20] J. Shen, Gamma-*convergence approximation to piecewise constant mumford-shah segmentation*, in Advanced Concepts for Intelligent Vision Systems, International Conference on, vol. 3708 of Lecture Notes in Computer Science, Antwerpen, Belgium, September 2005, Springer, pp. 499–506.
[21] L. A. Vese and T. F. Chan, *A multiphase level set framework for image segmentation using the mumford and shah model*, International Journal of Computer Vision, 50 (2002), pp. 271–293.
[22] Y. Xu and W. Yin, *A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion*, SIAM J. Imaging Sciences, 6 (2013), pp. 1758–1789.

Appendix A. C++ Implementation. The following sections present a C++ implementation of Algorithms 2 and 4. The implementation requires C++11, OpenCV, Eigen as well as Google GLOG and will be made publicly available at https://github.com/davidstutz. For brevity, some details are skipped in the following listings.

A.1. nmiPiano. The following listing presents the implementation of Algorithm 2, i.e. nmiPiano.

```
#ifndef NMIPIANO_H
#define NMIPIANO_H

#include <random>
#include <fstream>
#include <functional>
#include <Eigen/Dense>
#include <glog/logging.h>

/** \brief Implementation of nmiPiano (algorithm 4) as proposed in [1]:
 *  [1] P. Ochs, Y. Chen, T. Brox, T. Pock.
 *      iPiano: Inertial Proximal Algorithm for Nonconvex Optimization.
 *      SIAM J. Imaging Sciences, vol. 7, no. 2, 2014.
 * \author David Stutz
 */
class nmiPiano
{
public:

    /** \brief Options of algorithm. */
    struct Options
    {
        /** \brief Initial iterate. */
        Eigen::MatrixXf x_0;
        /** \brief Maximum number of iterations. */
        unsigned int max_iter;
        /** \brief Fixed beta in [0, 1). */
        float beta = 0.5f;
        /** \brief Fixed eta for backtracking the local lipschitz constant. */
        float eta = 1.05f;
        /** \brief Initialization of loca Lipschitz. */
        float L_0m1 = 1.f;
        /** \brief Whether to bound estimated Lipschitz constant below by the given L_n. */
        bool BOUND_L_N = false;
        /** \brief Termination criterion; stop if Delta_n smaller than epsilon. */
        float epsilon = 0;
    };

    /** \brief Structure representing an iteration, passed as is to a callback function
     * to be able to monitor process. */
    struct Iteration
    {
        /** \brief Current iterate. */
        Eigen::MatrixXf x_n;
        /** \brief Last iterate. */
        Eigen::MatrixXf x_nm1;
        /** \brief Update: x_np1 = x_n + Delta_x_n. */
        Eigen::MatrixXf Delta_x_n;
        /** \brief Difference between two iterates. */
        float Delta_n;
        /** \brief Smooth objective function at current iterate. */
        float f_x_n;
```

```
53          /** \brief Derivative of smooth objective at current iterate. */
54          Eigen::MatrixXf df_x_n;
55          /** Nonsmooth objective function at current iterate. */
56          float g_x_n;
57          /** \brief L_n of current iterate (within iterations, this is also used to estimate L_np1 via
        backtracking). */
58          float L_n;
59          /** \brief alpha_n of current iterate (within iterations, this is also used to estimate
        alpha_np1). */
60          float alpha_n;
61          /** \brief Current iteration. */
62          int n;
63      };
64
65      // Definition of different callbacks skipped ...
66
67      /** \brief Constructor.
68       * \param[in] f
69       * \param[in] df
70       * \param[in] prox_g
71       * \param[in] callback
72       */
73      nmiPiano(const std::function<float(const Eigen::MatrixXf&)> f,
74               const std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&)> df,
75               std::function<float(const Eigen::MatrixXf&)> g,
76               const std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&, float alpha)> prox_g,
77               Options options,
78               const std::function<void(const Iteration&)> callback = [](const Iteration &iteration) ->
        void { /* Nothing ... */ });
79
80      /** \brief Destructor. */
81      ~nmiPiano();
82
83      /** \brief Optimize the given objective using the nmiPiano algorithm.
84       * \param[out] x_star
85       * \param[out] f_x_star
86       */
87      void optimize(Eigen::MatrixXf &x_star, float &f_x_star);
88
89  protected:
90
91      /** \brief Initialize the iteration structure (i.e. set iteration 0):
92       * \param[in] iteration
93       */
94      void initialize(Iteration &iteration);
95
96      float estimateL(const Iteration &iteration);
97
98      /** \brief Do an iteration, i.e. compute x_np1.
99       * \param[in] iteration
100      * \param[in] beta
101      * \param[out] x_np1
102      */
103     void iterate(Iteration &iteration, float beta, Eigen::MatrixXf &x_np1);
104
105     /** \brief Check the lipschitz condition for backtracking.
106      * \param[in] iteration
107      * \param[in] x_np1
108      * \return
109      */
110     bool checkLipschitz(const Iteration& iteration, const Eigen::MatrixXf &x_np1);
111
112     /** \brief The smooth part of the objective function. */
113     std::function<float(const Eigen::MatrixXf&)> f_;
114     /** \brief Gradient of smooth part of the objective function (i.e. derivative). */
115     std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&)> df_;
116     /** \brief Convex, nonsmooth part of objective. */
117     std::function<float(const Eigen::MatrixXf&)> g_;
118     /** \brief Proximal map for g, i.e. (I + alpha_n*g)^(-1). */
119     std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&, float)> prox_g_;
120     /** \brief Callback, can e.g. be sued to monitor process using the provided information in the
        Iteration structure. */
121     std::function<void(const Iteration&)> callback_;
122     /** \brief Options.*/
123     Options options_;
124 };
125
126 // Implementation of different callbacks skipped ...
127
128 ////////////////////////////////////////////////////////////////////////////////
129 // nmiPiano::nmiPiano
130 ////////////////////////////////////////////////////////////////////////////////
131
132 nmiPiano::nmiPiano(const std::function<float(const Eigen::MatrixXf&)> f,
```

```
133         const std::function<void(const Eigen::MatrixXf&,Eigen::MatrixXf&)> df,
134         std::function<float(const Eigen::MatrixXf &)> g,
135         const std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&, float alpha)> prox_g,
136         nmiPiano::Options options,
137         const std::function<void(const nmiPiano::Iteration&)> callback)
138 {
139     f_ = f;
140     df_ = df;
141     g_ = g;
142     prox_g_ = prox_g;
143     callback_ = callback;
144     options_ = options;
145 }
146 
147 ////////////////////////////////////////////////////////////////////////////////
148 // nmiPiano::~nmiPiano
149 ////////////////////////////////////////////////////////////////////////////////
150 
151 nmiPiano::~nmiPiano()
152 {
153     // ...
154 }
155 
156 ////////////////////////////////////////////////////////////////////////////////
157 // nmiPiano::optimize
158 ////////////////////////////////////////////////////////////////////////////////
159 
160 void nmiPiano::optimize(Eigen::MatrixXf &x_star, float &f_x_star)
161 {
162     nmiPiano::Iteration iteration;
163     initialize(iteration);
164 
165     // Used for backtracking; referring to the next iterate x_np1:
166     Eigen::MatrixXf x_np1;
167 
168     for (unsigned int t = 0; t <= options_.max_iter; t++)
169     {
170         callback_(iteration);
171 
172         // Backtrack for the lipschitz constant L_n:
173         float L_nm1 = estimateL(iteration);
174 
175         // Fall back to last L_n in worst case.
176         if (std::isinf(L_nm1) || std::isnan(L_nm1) || options_.BOUND_L_N)
177         {
178             LOG_IF(INFO, !options_.BOUND_L_N) << "Could not get starting value for local Lipschitz
    constant, using L_nm1 instead (L_n = " << L_nm1 << ")";
179             L_nm1 = iteration.L_n;
180         }
181 
182         bool condition = false;
183 
184         int l = 0; // Backtracking steps.
185         do
186         {
187             iteration.L_n = std::pow(options_.eta,  l)*L_nm1;
188 
189             LOG_IF(FATAL, std::isinf(iteration.L_n) || std::isnan(iteration.L_n))
190                     << "Could not find the local Lipschitz constant - L_n = "
191                     << iteration.L_n << std::endl;
192             LOG_IF(INFO, l > 0 && l%1000 == 0)
193                     << "Having a hard time finding the local Lipschitz constant (L_n = " << iteration.
    L_n
194                     << "; L_nm1 = " << L_nm1 << "; eta = " << options_.eta << "; l = " << l << ")" <<
    std::endl;
195 
196             iteration.alpha_n = 2*(1 - options_.beta)/iteration.L_n;
197             LOG_IF(FATAL, iteration.alpha_n < 0)
198                     << "Cannot choose alpha_n - it does not exist: alpha_n = "
199                     << iteration.alpha_n << " (L_n = " << iteration.L_n << ")!";
200 
201 
202             // We fixes L_n and alpha_n, so we can try an iteration to check the
203             // Lipschitz condition.
204             iterate(iteration, options_.beta, x_np1);
205             condition = checkLipschitz(iteration, x_np1);
206 
207             l++;
208         }
209         while (!condition); // Now alpha_n and L_n are set correctly.
210 
211         iteration.x_nm1 = iteration.x_n;
212         iteration.x_n = x_np1;
213 
```

```
214         // For iteration 0, this is done in initialize():
215         iteration.f_x_n = f_(iteration.x_n);
216         iteration.g_x_n = g_(iteration.x_n);
217         df_(iteration.x_n, iteration.df_x_n);
218 
219         LOG_IF(FATAL, iteration.x_n.rows() != iteration.df_x_n.rows()
220                 || iteration.x_n.cols() != iteration.df_x_n.cols()) << "Output dimensions of df
    invalid.";
221 
222         iteration.n++;
223 
224         // Termination criterion
225         if (iteration.Delta_n < options_.epsilon && options_.epsilon > 0)
226         {
227             break;
228         }
229     }
230 
231     x_star = iteration.x_n;
232     f_x_star = iteration.f_x_n;
233 }
234 
235 ////////////////////////////////////////////////////////////////////////////////
236 // nmiPiano::initialize
237 ////////////////////////////////////////////////////////////////////////////////
238 
239 void nmiPiano::initialize(nmiPiano::Iteration &iteration)
240 {
241     iteration.n = 0;
242     iteration.x_n = options_.x_0;
243     iteration.x_nm1 = options_.x_0;
244 
245     iteration.g_x_n = g_(iteration.x_n);
246     iteration.f_x_n = f_(iteration.x_n);
247     df_(iteration.x_n, iteration.df_x_n);
248 
249     LOG_IF(FATAL, iteration.x_n.rows() != iteration.df_x_n.rows()
250             || iteration.x_n.cols() != iteration.df_x_n.cols()) << "Output dimensions of df invalid.";
251 
252     iteration.alpha_n = 0.1f; // Required for estimateL!
253 
254     // Estimate L_n.
255     if (options_.L_0m1 > 0)
256     {
257         iteration.L_n = options_.L_0m1;
258     }
259 
260     // Estimate L and take max.
261     if (iteration.df_x_n.squaredNorm() > 0.001)
262     {
263         iteration.L_n = std::max(iteration.L_n, estimateL(iteration));
264     }
265 
266     iteration.alpha_n = 2*(1 - options_.beta)/iteration.L_n;
267 }
268 
269 float nmiPiano::estimateL(const nmiPiano::Iteration &iteration)
270 {
271     Eigen::MatrixXf x_tilde = iteration.x_n - iteration.alpha_n*iteration.df_x_n;
272 
273     Eigen::MatrixXf df_x_tilde;
274     df_(x_tilde, df_x_tilde);
275 
276     Eigen::MatrixXf delta_df_x = iteration.df_x_n - df_x_tilde;
277     Eigen::MatrixXf delta_x = iteration.x_n - x_tilde;
278 
279     float L_n = std::sqrt(delta_df_x.squaredNorm())/std::sqrt(delta_x.squaredNorm ());
280     if (options_.BOUND_L_N) {
281         L_n = std::max(L_n, options_.L_0m1);
282     }
283 
284     return L_n;
285 }
286 
287 ////////////////////////////////////////////////////////////////////////////////
288 // nmiPiano::iterate
289 ////////////////////////////////////////////////////////////////////////////////
290 
291 void nmiPiano::iterate(nmiPiano::Iteration& iteration, float beta, Eigen::MatrixXf& x_np1)
292 {
293     // TODO: precompute x_n - x_nm1
294     iteration.Delta_x_n = - iteration.alpha_n*iteration.df_x_n + beta*(iteration.x_n - iteration.x_nm1
    );
295 
```

```
296     prox_g_(iteration.x_n + iteration.Delta_x_n, x_np1, iteration.alpha_n);
297 
298     LOG_IF(FATAL, iteration.x_n.rows() != x_np1.rows()
299             || iteration.x_n.cols() != x_np1.cols()) << "Output dimensions of prox_g invalid.";
300 
301     Eigen::MatrixXf Delta_n = iteration.x_n - x_np1;
302     iteration.Delta_n = std::sqrt(Delta_n.squaredNorm());
303 }
304 
305 ////////////////////////////////////////////////////////////////////////////////
306 // nmiPiano::checkLipschitz
307 ////////////////////////////////////////////////////////////////////////////////
308 
309 bool nmiPiano::checkLipschitz(const nmiPiano::Iteration& iteration, const Eigen::MatrixXf& x_np1)
310 {
311     float f_x_np1 = f_(x_np1);
312 
313     Eigen::MatrixXf residual = x_np1 - iteration.x_n;
314     Eigen::MatrixXf product = iteration.df_x_n.array()*residual.array();
315     float threshold = iteration.f_x_n + product.squaredNorm() + iteration.L_n/2.f * residual.
      squaredNorm();
316 
317     return (f_x_np1 <= threshold);
318 }
319 
320 #endif  /* NMIPIANO_H */
321 
```

A.2. iPiano. The following listing presents the implementation of Algorithm 4, i.e. iPiano.

```
1  #ifndef IPIANO_H
2  #define IPIANO_H
3  
4  #include <random>
5  #include <fstream>
6  #include <functional>
7  #include <Eigen/Dense>
8  #include <glog/logging.h>
9  
10 #include "nmipiano.h"
11 
12 /** \brief Implementation of iPiano (algorithm 5) as proposed in [1]:
13  *  [1] P. Ochs, Y. Chen, T. Brox, T. Pock.
14  *      iPiano: Inertial Proximal Algorithm for Nonconvex Optimization.
15  *      SIAM J. Imaging Sciences, vol. 7, no. 2, 2014.
16  * \author David Stutz
17  */
18 class iPiano : public nmiPiano
19 {
20 public:
21 
22     /** \brief Options of algorithm. */
23     struct Options : public nmiPiano::Options
24     {
25         /** \brief beta_0m1 to initialize alpha_0m1, delta_0m1 and gamma_0m1 according to Equations
       (21) and (22). */
26         float beta_0m1 = 0.5f;
27         /** \brief Fixed c_1. */
28         float c_1 = 1e-8;
29         /** \brief Fixed c_2. */
30         float c_2 = 1e-8;
31         /** \brief Number of dicsrete steps for alpha_n and beta_n to try. */
32         int steps = 10000;
33     };
34 
35     /** \brief Structure representing an iteration, passed as is to a callback function
36      * to be able to monitor process. */
37     struct Iteration : public nmiPiano::Iteration
38     {
39         /** \brief beta_n of current iterate (within iterations, this is also used to estimate
       beta_np1). */
40         float beta_n;
41         /** \brief delta_n of current iterate (within iterations, this is also used to estimate
       delta_np1). */
42         float delta_n;
43         /** \brief gamma_n of current iterate (within iterations, this is also used to estimate
       gamma_np1). */
44         float gamma_n;
45     };
46 
47     // Definition of different callbacks skipped ...
48 
```

```
49      /** \brief Constructor.
50       * \param[in] f
51       * \param[in] df
52       * \param[in] prox_g
53       * \param[in] callback
54       */
55      iPiano(const std::function<float(const Eigen::MatrixXf&)> f,
56              const std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&)> df,
57              std::function<float(const Eigen::MatrixXf&)> g,
58              const std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&, float alpha)> prox_g,
59              Options options,
60              const std::function<void(const Iteration&)> callback = [](const Iteration &iteration) ->
       void { /* Nothing ... */ });
61
62      /** \brief Destructor. */
63      ~iPiano();
64
65      /** \brief Optimize the given objective using the iPiano algorithm.
66       * \param[out] x_star
67       * \param[out] f_x_star
68       */
69      void optimize(Eigen::MatrixXf &x_star, float &f_x_star);
70
71  protected:
72
73      /** \brief Initialize the iteration structure (i.e. set iteration 0):
74       * \param[in] iteration
75       */
76      void initialize(Iteration &iteration);
77
78      /** \brief Callback, note: overwrites nmiPiano::callback_! */
79      std::function<void(const Iteration&)> callback_;
80      /** \brief Options (note: overwrites nmiPiano::options_!). */
81      Options options_;
82  };
83
84  // Implementation of different callbacks skipped ...
85
86  ////////////////////////////////////////////////////////////////////////////////
87  // iPiano::iPiano
88  ////////////////////////////////////////////////////////////////////////////////
89
90  iPiano::iPiano(const std::function<float(const Eigen::MatrixXf&)> f,
91          const std::function<void(const Eigen::MatrixXf&,Eigen::MatrixXf&)> df,
92          std::function<float(const Eigen::MatrixXf &)> g,
93          const std::function<void(const Eigen::MatrixXf&, Eigen::MatrixXf&, float alpha)> prox_g,
94          iPiano::Options options,
95          const std::function<void(const iPiano::Iteration&)> callback)
96          : nmiPiano(f, df, g, prox_g, options)
97  {
98      options_ = options;
99      callback_ = callback;
100 }
101
102 ////////////////////////////////////////////////////////////////////////////////
103 // iPiano::~iPiano
104 ////////////////////////////////////////////////////////////////////////////////
105
106 iPiano::~iPiano()
107 {
108     // ...
109 }
110
111 ////////////////////////////////////////////////////////////////////////////////
112 // iPiano::optimize
113 ////////////////////////////////////////////////////////////////////////////////
114
115 void iPiano::optimize(Eigen::MatrixXf &x_star, float &f_x_star)
116 {
117     iPiano::Iteration iteration;
118     initialize(iteration);
119
120     // Used for backtracking; referring to the next iterate x_np1:
121     Eigen::MatrixXf x_np1;
122
123     for (unsigned int t = 0; t <= options_.max_iter; t++)
124     {
125         callback_(iteration);
126
127         // Backtrack for the Lipschitz constant L_n and the updates
128         // alpha_n and beta_n:
129         float L_nm1 = estimateL(iteration);
130
131         // Fall back to last L_n in worst case.
```

```
132         if (std::isinf(L_nm1) || std::isnan(L_nm1) || options_.BOUND_L_N)
133         {
134             LOG_IF(INFO, !options_.BOUND_L_N) << "Could not get starting value for local Lipschitz
    constant, using L_nm1 (L_n = " << L_nm1 << ")";
135             L_nm1 = iteration.L_n;
136         }
137 
138         float delta_nm1 = iteration.delta_n;
139         bool condition = false;
140 
141         int l = 0;
142         do
143         {
144             iteration.L_n = std::pow(options_.eta, l)*L_nm1;
145 
146             LOG_IF(FATAL, std::isinf(iteration.L_n) || std::isnan(iteration.L_n))
147                     << "Could not find the local Lipschitz constant - L_n = "
148                     << iteration.L_n << std::endl;
149             LOG_IF(INFO, l > 0 && l%1000 == 0)
150                     << "Having a hard time finding the local Lipschitz constant (L_n = " << iteration.
    L_n
151                     << "; L_nm1 = " << L_nm1 << "; eta = " << options_.eta << "; l = " << l << ")" <<
    std::endl;
152 
153             float b = (delta_nm1 + iteration.L_n/2)/(options_.c_2 + iteration.L_n/2);
154             float beta = (b - 1)/(b - 1.f/2.f);
155 //            float beta = 0.999;
156 
157             for (iteration.beta_n = beta; iteration.beta_n >= 0;
158                     iteration.beta_n -= beta/options_.steps)
159             {
160                 bool take = false; // Whether to take current beta_n.
161 //                float alpha = 2*(1 - iteration.beta_n)/(iteration.L_n/2.f + options_.c_2);
162                 float alpha = 2*(1 - iteration.beta_n)/(iteration.L_n);
163 
164                 LOG_IF(FATAL, alpha < options_.c_1)
165                         << "Cannot choose alpha_n - it does not exist: c_1 = "
166                         << options_.c_1  << "; alpha_n = " << alpha
167                         << " (L_n = " << iteration.L_n << ")!";
168 
169                 for (iteration.alpha_n = alpha; iteration.alpha_n > options_.c_1;
170                         iteration.alpha_n -= (alpha - options_.c_1)/options_.steps)
171                 {
172                     // TODO save some computation here!
173                     iteration.delta_n = 1/iteration.alpha_n - iteration.L_n/2
174                             - iteration.beta_n/(2*iteration.alpha_n);
175                     iteration.gamma_n = 1/iteration.alpha_n - iteration.L_n/2
176                             - iteration.beta_n/iteration.alpha_n;
177 
178                     if (iteration.delta_n < delta_nm1
179                             && iteration.delta_n >= iteration.gamma_n
180                             && iteration.gamma_n >= options_.c_2)
181                     {
182                         // Good parameters, so take these!
183                         take = true;
184                         break;
185                     }
186                 }
187 
188                 if (take)
189                 {
190                     break;
191                 }
192             }
193 
194             // We fixes L_n and alpha_n, so we can try an iteration to check the
195             // Lipschitz condition.
196             iterate(iteration, iteration.beta_n, x_np1);
197             condition = checkLipschitz(iteration, x_np1);
198 
199             l++;
200         }
201         while (!condition); // Now alpha_n and L_n are set correctly.
202 
203         iteration.x_nm1 = iteration.x_n;
204         iteration.x_n = x_np1;
205 
206         // For iteration 0, this is done in initialize():
207         iteration.f_x_n = f_(iteration.x_n);
208         iteration.g_x_n = g_(iteration.x_n);
209         df_(iteration.x_n, iteration.df_x_n);
210 
211         LOG_IF(FATAL, iteration.x_n.rows() != iteration.df_x_n.rows()
212                 || iteration.x_n.cols() != iteration.df_x_n.cols()) << "Output dimensions of df
```

```
    invalid.";

        iteration.n++;

        // Termination criterion
        if (iteration.Delta_n < options_.epsilon && options_.epsilon > 0)
        {
            break;
        }
    }

    x_star = iteration.x_n;
    f_x_star = iteration.f_x_n;
}

////////////////////////////////////////////////////////////////////////////////
// iPiano::initialize
////////////////////////////////////////////////////////////////////////////////

void iPiano::initialize(iPiano::Iteration &iteration)
{
    iteration.n = 0;
    iteration.x_n = options_.x_0;
    iteration.x_nm1 = options_.x_0;

    iteration.g_x_n = g_(iteration.x_n);
    iteration.f_x_n = f_(iteration.x_n);

    df_(iteration.x_n, iteration.df_x_n);

    LOG_IF(FATAL, iteration.x_n.rows() != iteration.df_x_n.rows()
            || iteration.x_n.cols() != iteration.df_x_n.cols()) << "Output dimensions of df invalid.";

    iteration.alpha_n = 0.1f; // Required for estimateL!

    // Estimate L_n.
    if (options_.L_0m1 > 0)
    {
        iteration.L_n = options_.L_0m1;
    }

    // Estimate L and take max.
    if (iteration.df_x_n.squaredNorm() > 0.001)
    {
        iteration.L_n = std::max(iteration.L_n, estimateL(iteration));
    }

    // beta_0m1 is given by the options!
    iteration.beta_n = options_.beta_0m1;

    // Initialize alpha_0m1 to suffice alpha_0m1 <= 2*(1 - beta_0m)/L_0m1
    // and delta_0m1 >= gamma_0m1 > c_2 where delta_0m1 and gamma_0m1 depend upon alpha_0m1.

//    float alpha = 2*(1 - iteration.beta_n)/(iteration.L_n/2.f + options_.c_2);
    float alpha = 2*(1 - iteration.beta_n)/(iteration.L_n);

    LOG_IF(FATAL, alpha < options_.c_1)
            << "Cannot choose alpha_n - it does not exist: c_1 = "
            << options_.c_1  << "; alpha_n = " << alpha
            << " (L_n = " << iteration.L_n << ")!";

    for (iteration.alpha_n = alpha; iteration.alpha_n > options_.c_1; iteration.alpha_n -= (alpha -
    options_.c_1)/options_.steps)
    {
        // TODO save some computation here!
        iteration.delta_n = 1/iteration.alpha_n - iteration.L_n/2
                - iteration.beta_n/(2*iteration.alpha_n);
        iteration.gamma_n = 1/iteration.alpha_n - iteration.L_n/2
                - iteration.beta_n/iteration.alpha_n;

        if (iteration.delta_n >= iteration.gamma_n && iteration.gamma_n >= options_.c_2)
        {
            // Good parameters, so take these!
            break;
        }
    }
}

#endif  /* IPIANO_H */
```