Archive of Numerical Software
vol. 5, no. 1, 2017, pages 3–19
DOI: 10.11588/ans.2017.1.27475

Using DUNE-ACFEM for Non-smooth Minimization of Bounded Variation Functions

Martin Alkämper[1] and Andreas Langer[1]

[1]Institute for Applied Analysis and Numerical Simulation, University of Stuttgart

Received: January 29th, 2016; **final revision:** October 6th, 2016; **published:** March 6th, 2017.

Abstract: The utility of DUNE-ACFEM is demonstrated to work well for solving a non-smooth minimization problem over bounded variation functions by implementing a primal-dual algorithm. The implementation is based on the simplification provided by DUNE-ACFEM. Moreover, the convergence of the discrete minimizer to the continuous one is shown theoretically.

1 Introduction

The DUNE-framework [7] and in particular the discretization module DUNE-FEM [11] provides the means to handle discrete functions, operators and solvers on different grids. Still, implementing complicated partial differential equations (PDEs) and their solvers is cumbersome and tedious. The DUNE-module DUNE-ACFEM aims to simplify the usage of DUNE-FEM by defining expression templates for discrete functions and PDE models. This allows to linearly combine discrete functions and PDE models. Additionally it supports parallel and adaptive finite-element schemes on continuous discrete functions for the predefined and combined models [14]. The flexibility of DUNE (and DUNE-FEM, DUNE-ACFEM) allows e.g. to exchange discrete spaces by a single line of code or to change the gridtype and linear solvers.

We will demonstrate the ease of implementation with DUNE-ACFEM by minimizing a non-smooth functional consisting of a combined L^1/L^2-data fidelity term and a total variation term. Such an optimization problem has been shown to effectively remove Gaussian and salt-and-pepper noise, see [16, 19]. In order to compute an approximate solution we use the primal-dual algorithm proposed in [10], which requires a saddle point formulation of the problem. For the numerical implementation we discretize using finite-element spaces defined over locally refined conforming grids. Motivated by the works [3, 4, 5, 17], where the considered functional is composed solely of an L^2-data term and a total variation term, we refine the grid adaptively using an a priori criterion. Similar as in [3] we show for the considered minimization problem, that a minimizer over a finite element space converges to a minimizer in the space of functions of bounded variation as the mesh-size goes to 0.

In contrast to previous works [3, 4, 5, 17], we consider an additional non-smooth L^1-data term in the objective, which has to be treated carefully. Moreover, due to the use of DUNE-ALUGRID [1] and the capabilities of DUNE-ACFEM the resulting algorithm is intrinsically parallelized by domain decomposition.

The rest of the paper is structured as follows. In Section 2 we formulate the continuous and the discrete problem with the respective discrete spaces. In particular for a certain discretization we prove that the discrete problem converges to the continuous one as the mesh-size goes to zero. In Section 3 we give a short overview of Dune-ACFem and discuss how it is used to implement the primal-dual algorithm. Numerical examples showing applicability of our proposed implementation and experiments testing different discretizations are presented in Section 4. Finally in Section 5 we conclude with a short summary and possible future research.

2 Problem Formulation

We consider the following problem

$$\min_{v\in BV(\Omega)\cap L^2(\Omega)} J_{\alpha_1,\alpha_2}(v) := \alpha_1\|v-g\|_{L^1(\Omega)} + \alpha_2\|v-g\|^2_{L^2(\Omega)} + |Dv|(\Omega), \tag{1}$$

where $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, is an open bounded set with Lipschitz boundary, $g \in L^2(\Omega)$ is a given datum, $\alpha_i \geq 0$ for $i = 1, 2$ with $\alpha_1 + \alpha_2 > 0$ and $BV(\Omega) \subset L^1(\Omega)$ denotes the space of functions with bounded variation. That is, $v \in BV(\Omega)$ if and only if

$$|Dv|(\Omega) := \sup\left\{\int_\Omega v \operatorname{div}\vec{\phi}\,dx,\ \vec{\phi}\in C_0^\infty(\Omega,\mathbb{R}^d),\ \|\,|\vec{\phi}|_2\|_{C^0(\Omega)} \leq 1\right\} \tag{2}$$

is finite, see [2, 13]. The space $BV(\Omega)$ endowed with the norm $\|v\|_{BV(\Omega)} = \|v\|_{L^1(\Omega)} + |Dv|(\Omega)$ is a Banach space [13]. For $\alpha_2 > 0$ the minimization problem (1) admits a unique solution owing to the strict convexity of the quadratic term [19]. Note, that if $\alpha_1 = 0$ in (1), then we obtain the functional used in [3, 4, 5, 17].

2.1 Discretization

Let $(\mathcal{T}_h)_{h>0}$ be a sequence of shape-regular triangulations of Ω with diameter $h = \max_{T\in\mathcal{T}_h} \operatorname{diam}(T)$ and $\mathcal{S}_h$ be the set of its mesh entities of codimension 1 (i.e. edges for $d = 2$). We define the following finite element spaces

$$\mathcal{L}^0(\mathcal{T}_h) = \{q_h \in L^1(\Omega) : q_h|_T \text{ is constant for each } T \in \mathcal{T}_h\}$$
$$\mathcal{S}^1(\mathcal{T}_h) = \{v_h \in C(\overline{\Omega}) : v_h|_T \text{ is affine for each } T \in \mathcal{T}_h\}.$$

For the vector-valued versions, we write $\mathcal{L}^0(\mathcal{T}_h)^d$ and $\mathcal{S}^1(\mathcal{T}_h)^d$, respectively, and boundary conditions are denoted via a subindex. In particular we use the subindex $_0$ for zero boundary values and the subindex $_N$ for zero boundary values in normal direction, e.g., $\mathcal{S}^1_0(\mathcal{T}_h) := \{v_h \in \mathcal{S}^1(\mathcal{T}_h) : v_h = 0 \text{ on } \partial\Omega\}$ and $\mathcal{L}^0_N(\mathcal{T}_h)^d = \{q_h \in \mathcal{L}^0(\mathcal{T}_h)^d : q_h \cdot \vec{n} = 0 \text{ on } \partial\Omega\}$ where $\vec{n}$ is the unit vector in normal direction. The nodal interpolant $\mathcal{I}_h v \in \mathcal{S}^1(\mathcal{T}_h)$ of a function $v \in W^{2,p}$, with $\frac{d}{2} < p \leq \infty$ or $p = 1$ if $d = 2$, satisfies

$$\|v - \mathcal{I}_h v\|_{L^p(\Omega)} + h\|\nabla(v - \mathcal{I}_h v)\|_{L^p(\Omega)} \leq c_{\mathcal{I}} h^2 \|D^2 v\|_{L^p(\Omega)},$$

where $c_{\mathcal{I}} > 0$ is a constant independent of h; cf. [8].

We recall, that the space $BV(\Omega)$ is continuously embedded in $L^p(\Omega)$ for $1 \leq p \leq \frac{d}{d-1}$, i.e., there is a constant $c_{BV} > 0$ such that $\|v\|_{L^p(\Omega)} \leq c_{BV}\|v\|_{BV} = c_{BV}\left(\|v\|_{L^1(\Omega)} + |Dv|(\Omega)\right)$ for any $v \in BV(\Omega)$. For $1 \leq p < \frac{d}{d-1}$ this embedding is compact; cf. [2]. Smooth functions are dense in $BV(\Omega) \cap L^p(\Omega)$, $1 \leq p < \infty$. In particular, for $v \in BV(\Omega) \cap L^2(\Omega)$ and $\delta > 0$ there exists $\varepsilon := \varepsilon(\delta) > 0$ and functions $(v_\varepsilon)_{\varepsilon>0} \subset C^\infty \cap BV(\Omega) \cap L^2(\Omega)$ such that

$$\|\nabla v_\varepsilon\|_{L^1(\Omega)} \leq |Dv|(\Omega) + c_0\delta, \tag{3}$$
$$\|v - v_\varepsilon\|_{L^2(\Omega)} \leq c_1\delta, \qquad \|v - v_\varepsilon\|_{L^1(\Omega)} \leq c_2\delta, \tag{4}$$
$$\|D^2 v_\varepsilon\|_{L^2(\Omega)} \leq \varepsilon^{-2}\|v\|_{L^2(\Omega)}, \qquad \|D^2 v_\varepsilon\|_{L^1(\Omega)} \leq \varepsilon^{-2}\|v\|_{L^1(\Omega)}, \tag{5}$$

cf. [3]. These inequalities follow from standard mollifier techniques, see e.g. [13]. Now we are able to show the following convergence result, which follows similar ideas as the proof of [3, Theorem 3.1].

Theorem 2.1 *Let $u_h \in \arg\min_{v\in\mathcal{S}^1(\mathcal{T}_h)} J_{\alpha_1,\alpha_2}(v)$ and $u \in BV(\Omega)\cap L^2(\Omega)$ be a minimizer of the function J_{α_1,α_2}. Then we have that $J_{\alpha_1,\alpha_2}(u_h) \to J_{\alpha_1,\alpha_2}(u)$ as $h \to 0$. If additionally $\alpha_2 > 0$, then $u_h \to u$ in $L^2(\Omega)$ as $h \to 0$.*

Proof *By the optimality of u we have $J_{\alpha_1,\alpha_2}(u_h) - J_{\alpha_1,\alpha_2}(u) \geq 0$. For $\delta > 0$ let $u_\varepsilon \in C^\infty(\Omega)\cap L^2(\Omega)$ as above and $\mathcal{I}_h u_\varepsilon$ its nodal interpolant, i.e., $\mathcal{I}_h u_\varepsilon \in \mathcal{S}^1(\mathcal{T}_h)$. Then we deduce*

$$\begin{aligned} J_{\alpha_1,\alpha_2}(u_h) - J_{\alpha_1,\alpha_2}(u) &\leq J_{\alpha_1,\alpha_2}(\mathcal{I}_h u_\varepsilon) - J_{\alpha_1,\alpha_2}(u) \\ &= \|\nabla \mathcal{I}_h u_\varepsilon\|_{L^1(\Omega)} + \alpha_1\|\mathcal{I}_h u_\varepsilon - g\|_{L^1(\Omega)} + \alpha_2\|\mathcal{I}_h u_\varepsilon - g\|^2_{L^2(\Omega)} \\ &\quad - |Du|(\Omega) - \alpha_1\|u-g\|_{L^1(\Omega)} - \alpha_2\|u-g\|^2_{L^2(\Omega)}. \end{aligned}$$

Using (3) and $\|\mathcal{I}_h u_\varepsilon - g\|^2_{L^2(\Omega)} - \|u-g\|^2_{L^2(\Omega)} = \int_\Omega (\mathcal{I}_h u_\varepsilon - u)(\mathcal{I}_h u_\varepsilon + u - 2g)$ we obtain

$$\begin{aligned} J_{\alpha_1,\alpha_2}(u_h) - J_{\alpha_1,\alpha_2}(u) &\leq \|\nabla \mathcal{I}_h u_\varepsilon\|_{L^1(\Omega)} - \|\nabla u_\varepsilon\|_{L^1(\Omega)} + c_0\delta + \alpha_1\|\mathcal{I}_h u_\varepsilon - g\|_{L^1(\Omega)} \\ &\quad - \alpha_1\|u-g\|_{L^1(\Omega)} + \alpha_2\int_\Omega (\mathcal{I}_h u_\varepsilon - u)(\mathcal{I}_h u_\varepsilon + u - 2g). \end{aligned}$$

By the triangle-inequality and the Cauchy-Schwarz inequality we get

$$\begin{aligned} J_{\alpha_1,\alpha_2}(u_h) - J_{\alpha_1,\alpha_2}(u) &\leq \|\nabla(\mathcal{I}_h u_\varepsilon - u_\varepsilon)\|_{L^1(\Omega)} + c_0\delta + \alpha_1\|\mathcal{I}_h u_\varepsilon - g - (u-g)\|_{L^1(\Omega)} \\ &\quad + \alpha_2\|\mathcal{I}_h u_\varepsilon - u\|_{L^2(\Omega)}(\|\mathcal{I}_h u_\varepsilon\|_{L^2(\Omega)} + \|u\|_{L^2(\Omega)} + 2\|g\|_{L^2(\Omega)}) \\ &= \|\nabla(\mathcal{I}_h u_\varepsilon - u_\varepsilon)\|_{L^1(\Omega)} + c_0\delta + \alpha_1\|\mathcal{I}_h u_\varepsilon - u_\varepsilon + u_\varepsilon - u\|_{L^1(\Omega)} \\ &\quad + \alpha_2\|\mathcal{I}_h u_\varepsilon - u_\varepsilon + u_\varepsilon - u\|_{L^2(\Omega)}(\|\mathcal{I}_h u_\varepsilon\|_{L^2(\Omega)} + \|u\|_{L^2(\Omega)} + 2\|g\|_{L^2(\Omega)}) \\ &\leq \|\nabla(\mathcal{I}_h u_\varepsilon - u_\varepsilon)\|_{L^1(\Omega)} + c_0\delta + \alpha_1\|\mathcal{I}_h u_\varepsilon - u_\varepsilon\|_{L^1(\Omega)} + \alpha_1\|u_\varepsilon - u\|_{L^1(\Omega)} \\ &\quad + \alpha_2(\|\mathcal{I}_h u_\varepsilon - u_\varepsilon\|_{L^2(\Omega)} + \|u_\varepsilon - u\|_{L^2(\Omega)})(\|\mathcal{I}_h u_\varepsilon\|_{L^2(\Omega)} + \|u\|_{L^2(\Omega)} + 2\|g\|_{L^2(\Omega)}). \end{aligned}$$

The bound $\|\mathcal{I}_h u_\varepsilon\|_{L^2(\Omega)} + \|u\|_{L^2(\Omega)} + 2\|g\|_{L^2(\Omega)} \leq \tilde{c}$, which holds provided that $h \leq \varepsilon$, and the nodal interpolant estimate yield

$$\begin{aligned} J_{\alpha_1,\alpha_2}(u_h) - J_{\alpha_1,\alpha_2}(u) &\leq c_I h\|D^2 u_\varepsilon\|_{L^1(\Omega)} + c_0\delta + c_I\alpha_1 h^2\|D^2 u_\varepsilon\|_{L^1(\Omega)} + \alpha_1\|u_\varepsilon - u\|_{L^1(\Omega)} \\ &\quad + \tilde{c}\alpha_2(c_I h^2\|D^2 u_\varepsilon\|_{L^2(\Omega)} + \|u_\varepsilon - u\|_{L^2(\Omega)}). \end{aligned}$$

Using (4), (5), and the bound $\|u\|_{L^2(\Omega)} \leq \tilde{c}$ we get

$$J_{\alpha_1,\alpha_2}(u_h) - J_{\alpha_1,\alpha_2}(u) \leq C_1\frac{h}{\varepsilon^2} + C_2\delta + C_3\frac{h^2}{\varepsilon^2} + C_4\delta + C_5\frac{h^2}{\varepsilon^2} + C_6\delta.$$

Let h be sufficiently small, i.e., $h \leq \min\{\delta, \delta\varepsilon^2\}$. Then for $\delta \to 0$ we deduce that $h \to 0$ and hence $J_{\alpha_1,\alpha_2}(u_h) \to J_{\alpha_1,\alpha_2}(u)$.

If $\alpha_2 > 0$, then J_{α_1,α_2} is strictly convex and it follows that

$$J_{\alpha_1,\alpha_2}(u_h) - J_{\alpha_1,\alpha_2}(u) \geq \alpha_2\|u_h - u\|^2_{L^2(\Omega)},$$

cf. [19, Lemma 3.8]. Hence $u_h \to u$ for $h \to 0$. □

By the definition of the total variation (2) an equivalent formulation of (1) reads

$$\min_{v\in BV(\Omega)\cap L^2(\Omega)} \max_{\vec{p}\in C_0^\infty(\Omega,\mathbb{R}^d),|\vec{p}|_2\leq 1} \alpha_1\|v-g\|_{L^1(\Omega)} + \alpha_2\|v-g\|^2_{L^2(\Omega)} + \int_\Omega v\,\mathrm{div}\,\vec{p}. \tag{6}$$

An analog equivalence exists for finite element spaces [3]. In particular, for $v \in \mathcal{S}^1(\mathcal{T}_h)$ we have that

$$|Dv|(\Omega) = \sup_{\vec{p} \in \mathcal{L}^0_N(\mathcal{T}_h)^d, |\vec{p}|_2 \leq 1} \int_\Omega \nabla v \cdot \vec{p} \, \mathrm{d}x \tag{7}$$

leading to

$$\inf_{v \in \mathcal{S}^1(\mathcal{T}_h)} J_{\alpha_1,\alpha_2}(v) = \inf_{v \in \mathcal{S}^1(\mathcal{T}_h)} \sup_{\vec{p} \in \mathcal{L}^0_N(\mathcal{T}_h)^d} \int_\Omega \nabla v \cdot \vec{p} \, \mathrm{d}x + \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|^2_{L^2(\Omega)} - I_{\mathcal{K}}(\vec{p}),$$

where

$$I_{\mathcal{K}}(\vec{p}) := \begin{cases} 0 & \text{if } \vec{p} \in \mathcal{K} \\ \infty & \text{else} \end{cases}$$

is the indicator function of $\mathcal{K} := \{\vec{p} \in L^1(\Omega)^d \ : \ |\vec{p}|_2 \leq 1\}$. For $v \in \mathcal{L}^0(\mathcal{T}_h)$ we obtain

$$|Dv|(\Omega) = \sup_{(\vec{\beta}_S)_{S \in \mathcal{S}_h}, |\vec{\beta}_S|_2 \leq 1} \sum_{S \in \mathcal{S}_h} |S| \vec{\beta}_S [v]_S, \tag{8}$$

where $[v]_S \in \mathbb{R}$ defines the jump across $S \in \mathcal{S}_h$ in normal direction. Hence in this case the discrete problem reads as

$$\inf_{v \in \mathcal{L}^0(\mathcal{T}_h)} J_{\alpha_1,\alpha_2}(v) = \inf_{v \in \mathcal{L}^0(\mathcal{T}_h)} \sup_{\vec{p} \in \mathcal{L}^0_N(\mathcal{S}_h)} \sum_{S \in \mathcal{S}_h} |S| \vec{p}_S [v]_S + \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|^2_{L^2(\Omega)} - I_{\mathcal{K}}(\vec{p}).$$

While this formulation is equivalent to the primal formulation, in general we cannot expect convergence to the continuous solution, unless all the jumps are correctly represented by a descendant triangulation. For more details we refer the reader to [3, Section 4].

The primal-dual problem does not have a unique solution in general, even if the primal problem is strictly convex ($\alpha_2 > 0$). This is due to the fact, that the dual problem of (1), even for $\alpha_1 = 0$ and $\alpha_2 > 0$, is only convex but not strictly convex, see for example [15, 18].

2.2 Primal-Dual Algorithm

In the following we discretize using finite element spaces $\mathcal{U}$ (e.g. $\mathcal{S}^1(\mathcal{T}_h)$ or $\mathcal{L}^0(\mathcal{T}_h)$) and $\mathcal{P}$ (e.g. $\mathcal{S}^1_0(\mathcal{T}_h)^d$ or $\mathcal{L}^0_N(\mathcal{T}_h)^d$) for the primal variable u and the dual variable $\vec{p}$, respectively. We denote by $\langle \cdot, \cdot \rangle$ the L^2-inner product or the application of an L^2-functional. Following the ideas of Chambolle and Pock [10, Algorithm 1] we formulate our primal-dual algorithm by identifying $F^* : \mathcal{P} \to \mathbb{R} \cup \{+\infty\}$, $G : \mathcal{U} \to \mathbb{R} \cup \{+\infty\}$, and $K : \mathcal{U} \to \mathcal{P}^*$ with

$$F^*(\vec{p}) = I_{\mathcal{K}}(\vec{p}), \ G(v) = \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|^2_{L^2(\Omega)}, \text{ and } \langle Kv, \vec{p} \rangle = \int_\Omega v \operatorname{div} \vec{p},$$

where we assume that each element in $\mathcal{P}$ has a weak derivative. If this is not the case, $\langle Kv, \vec{p} \rangle$ is to be understood using the identifications suggested by the equations (7) and (8). That is $\langle Kv, \vec{p} \rangle = \int_\Omega \nabla v \cdot \vec{p} \, \mathrm{d}x$ and

$$\langle Kv, \vec{p} \rangle = \int_\Omega v \operatorname{div} \vec{p} = \sum_{T \in \mathcal{T}_h} \int_{\partial T} v \vec{p} \cdot \vec{n} = \sum_{S \in \mathcal{S}_h} [v]_S \vec{n} \cdot \int_S \vec{p} = \sum_{S \in \mathcal{S}_h} |S| \vec{p}_S [v]_S,$$

cf. [3, Lemma 4.1], respectively.

In the case that $\nabla v \notin L^2(\Omega)$ for $v \in \mathcal{U}$ or $\operatorname{div} \vec{q} \notin L^2(\Omega)$ for $\vec{q} \in \mathcal{P}$ we use the following identities

$$\begin{aligned}
&\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h), \mathcal{P} = \mathcal{L}^0_N(\mathcal{S}_h): && \langle \nabla v, \vec{q} \rangle := -\sum_{S \in \mathcal{S}_h} |S| \vec{p}_{|S}[v]_S =: -\langle v, \operatorname{div} \vec{q} \rangle, \\
&\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h), \mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h): && \langle \nabla v, \vec{q} \rangle := -\langle v, \operatorname{div} \vec{q} \rangle, \\
&\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h), \mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h): && \langle \nabla v, \vec{q} \rangle =: -\langle v, \operatorname{div} \vec{q} \rangle.
\end{aligned} \tag{9}$$

We assume, that the datum $g \in L^2(\Omega)$ and the cost parameters α_i are given. Then in our algorithm we initialize $u_0 = \bar{u}_0 \in \mathcal{U}$, e.g., $u_0 \equiv 0$ or $u_0 = \mathbb{P}g$, where $\mathbb{P} : L^2(\Omega) \to \mathcal{U}$ denotes the L^2-projection onto the discrete space $\mathcal{U}$, and $\vec{p}_0 \in \mathcal{P}$ to be the zero function. As parameters we need the step sizes $\sigma, \tau > 0$, the coefficient $\beta = \frac{\tau\alpha_1}{1+2\tau\alpha_2}$, and the overrelaxation parameter $\theta \in [0, 1]$.
Then our primal-dual algorithm iterates starting with $k = 0$ as follows:

1. Set $\bar{p} \in \mathcal{P}$ with step size σ as

$$\langle \bar{p}, \vec{q} \rangle = \langle \vec{p}_k + \sigma \nabla \bar{u}_k, \vec{q} \rangle \quad \forall \vec{q} \in \mathcal{P}. \tag{10}$$

   If $\nabla \bar{u}_k \in \mathcal{P}$, then we can use the strong formulation, otherwise we use the identities (9). As the algorithm is derived from formulation (6), we need to guarantee that $|\vec{p}_k|_2 \leq 1$ for all k. This is done by the following update

$$\vec{p}_{k+1} = (I + \sigma \partial F^*)^{-1}(\bar{p}) \quad \Leftrightarrow \quad \vec{p}_{k+1}(x) = \frac{\bar{p}(x)}{\max\left(|\bar{p}(x)|_2, 1\right)}, \tag{11}$$

   for almost any $x \in \Omega$. Note, that due to the structure of the operator $(I + \sigma \partial F^*)^{-1}$ (see [10] for more details) $\vec{p}_{k+1} \in \mathcal{P} \cap \mathcal{K}$.

2. Update u_k using

$$u_{k+1} = (I + \tau \partial G)^{-1}(u_k + \tau \operatorname{div} \vec{p}_{k+1}) \quad \Leftrightarrow \quad u_{k+1}(x) = \begin{cases} z(x) - \beta & \text{if } z(x) - \beta \geq \mathbb{P}g(x) \\ z(x) + \beta & \text{if } z(x) + \beta \leq \mathbb{P}g(x) \\ \mathbb{P}g(x) & \text{else}, \end{cases} \tag{12}$$

   for almost any $x \in \Omega$, where $z \in \mathcal{U}$ is defined via

$$\langle z, v \rangle = \frac{1}{1 + 2\tau\alpha_2} \left(\langle u_k + 2\tau\alpha_2 g, v \rangle + \tau \langle \operatorname{div} \vec{p}_{k+1}, v \rangle \right) \quad \forall v \in \mathcal{U}. \tag{13}$$

   If the $\operatorname{div} \vec{p}_{k+1} \notin L^2(\Omega)$, then we use again the identities of (9).

3. As proposed in [10] we overrelaxate to speed up the algorithm:

$$\bar{u}_{k+1} = u_{k+1} + \theta(u_{k+1} - u_k) \tag{14}$$

4. If the stopping criterion

$$\frac{\|u_{k+1} - u_k\|_{L^2}}{\tau} + \frac{\|\vec{p}_{k+1} - \vec{p}_k\|_{L^2}}{\sigma} < TOL \tag{15}$$

   holds, we terminate the algorithm, otherwise we set $k \to k + 1$ and repeat.

Note that in equation (11) with $\mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d$ it is a priori not clear that $\vec{p}_{k+1} \in \mathcal{S}^1_0(\mathcal{T}_h)^d$, as taking the pointwise maximum of two piecewise linear functions results in a piecewise linear function on a finer grid. So in this case we additionally apply the nodal interpolation operator $\mathcal{I}_h : C^0(\Omega)^d \to \mathcal{P}$, which then guarantees $\| |\vec{p}_{k+1}|_2 \|_{L^\infty} \leq 1$ and $\vec{p}_{k+1} \in \mathcal{P}$. A similar problem arises for $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$

and equation (12), which we treat analogously. Furthermore note, that testing with the complete test-space $\mathcal{U}$ or $\mathcal{P}$ (eqs. (10), (13)), respectively, is equivalent to an L^2-projection onto the respective space.

Theorem 1 of [10] guarantees convergence of this algorithm to a saddle point, if the according discretization of problem (6) has a saddle point and additionally $\theta = 1$ and

$$\sigma\tau B^2 < 1, \tag{16}$$

where $B := \|\nabla\|_{L^2}$ is the operator norm of the gradient operator on the discrete space $\mathcal{U}$. This norm is bounded for discrete functions and of order $O(1/h_{\min})$, where $h_{\min} := \min_{T\in\mathcal{T}_h} \operatorname{diam}(T)$, which implies that using finer meshes will always result in more steps of the primal-dual algorithm. Even using an adaptive mesh is not beneficial to the stepsize, as B depends on the minimum mesh-width.

3 Implementation Details

The algorithm is implemented in the Dune-project non–smooth–minization . This project is compatible with the 2.4-release and available on the website `http://www.ians.uni-stuttgart.de/nmh/downloads`. It requires the modules Dune-ACFem and all modules that are required by it, available at `gitlab.dune-project.org`. The installation works the standard Dune-way. (see Appendix A)

3.1 On Dune-ACFem

Dune-ACFem [14] is a simulation framework based on Dune-Fem [11], which is a discretization framework based on Dune. Dune-Fem provides most generally speaking finite-element spaces on generic grids and all the corresponding utilities to construct a finite-element scheme. Examples are given in the Dune-Fem-Howto. For more information consult [11].

The add-on module Dune-ACFem provides expression templates (and also analytical functions, that can be evaluated on the mesh) for the discrete functions of Dune-Fem and also for models similar but more elaborate to those in the Dune-Fem-Howto. This aims at simplifying the algorithmic formulation of elliptic and parabolic PDEs. The expression templates for discrete functions allow for addition, multiplication with scalars, multiplication with scalar functions, scalarproducts of the components of two functions, and unary expressions like componentwise sin, cos, sqrt, exp.

Dune-ACFem is designed to treat 2nd order elliptic PDEs of the form

$$\begin{aligned} -\nabla\cdot(A(x,u,\nabla u)\,\nabla u) + \nabla\cdot(b(x,u,\nabla u)\,u) + c(x,u,\nabla u)\,u &= f(x) \quad \text{in } \Omega,\\ u &= g_D \quad \text{on } \Gamma_D,\\ (A(x,u,\nabla u)\nabla u)\cdot\nu + \alpha(x,u)\,u &= g_N \quad \text{on } \Gamma_R,\\ (A(x,u,\nabla u)\nabla u)\cdot\nu &= g_N \quad \text{on } \Gamma_N, \end{aligned} \tag{17}$$

or, in weak formulation,

$$\begin{aligned} \int_\Omega (A\,\nabla u)\cdot\nabla\phi\,dx + \int_\Omega (\nabla\cdot(b\,u) + c\,u)\,\phi\,dx - \int_\Omega f(x)\,\phi\,dx \\ + \int_{\Gamma_R} \alpha\,u\,\phi\,do - \int_{\Gamma_N\cup\Gamma_R} g_N\,\phi\,do = 0, \\ \langle\Pi, u\rangle = \langle\Pi, g_D\rangle. \end{aligned} \tag{18}$$

Possible Dirichlet data is enforced by standard Lagrange test-functions Π. The multiplication with the test-functions ϕ and the integral (quadrature) is provided by Dune-Fem. The other part

of the integral has to be provided by the model, which is the central concept of DUNE-ACFEM. A model is basically a tuple of the form

$$\mathcal{M} := (\sigma_E,\ \mu_E,\ \rho_I,\ g_N,\ g_D,\ w_D,\ f, \chi_R,\ \chi_N,\ \chi_D, \Psi)$$

with the following constituents:

Flux	$\sigma_E : \mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}^{m\times d} \to \mathbb{R}^{m\times d}$,
Source	$\mu_E : \mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}^{m\times d} \to \mathbb{R}^m$,
Robin-flux	$\rho_I : \mathbb{R}^d \times R^m \to \mathbb{R}^m$,
Robin-indicator	$\chi_R : \partial\Omega \to \{0,1\}$,
Neumann-data	$g_N : \mathbb{R}^d \to \mathbb{R}^m$,
Neumann-indicator	$\chi_N : \partial\Omega \to \{0,1\}$,
Dirichlet-data	$g_D : \mathbb{R}^d \to \mathbb{R}^m$,
Dirichlet-indicator	$\chi_D : \partial\Omega \to \{0,1\}$,
Bulk-forces	$f : \mathbb{R}^d \to \mathbb{R}^m$,
Force-functional	$\Psi \in \mathcal{V}^*$.

Using such a model results in the following weak formulation

$$\begin{aligned}\sum_{E\in\mathcal{T}} \Big(\int_E \Big(\sigma_E(x,U,\nabla U) : \nabla V + \mu_E(x,U,\nabla U)\cdot V - f\cdot V\Big) \\ + \int_{I\in E\cap\partial\Omega} \Big(\big(\chi_R(x)\,\rho_I(x,U) - \chi_N(x)\,g_N(x)\big)\cdot V\Big)\Big) - \langle \Psi, V\rangle = 0, \\ \langle \Pi, \chi_D\ U\rangle = \langle \Pi, \chi_D\, g_D\rangle.\end{aligned} \tag{19}$$

This is directly reflected in code, as a model is derived from an interface class that requires exactly the implementation of the above constituents (and the linearization of Flux, Source and Robin-flux for non-linear models). Each model is derived from a default *zero-model*, where all the constituents are set to zero, so only non-zero contributions need to be implemented. DUNE-ACFEM allows forming algebraic expressions from existing models. The generated model-expressions fulfill the model-interface just as "hand-coded" ones. This allows forming of complicated models from a set of convenient pre-built skeleton-models. For a list of predefined models consult [14]. The algebraic expressions include linear combinations of models and multiplying with L^∞-functions.

Other important concepts are the FemScheme and the EllipticOperator. The EllipticOperator applies or assembles the discretization of equation (18). The FemScheme then solves the given problem and chooses the necessary linear or non-linear solvers, depending on structural information given by the model. DUNE-ACFEM mostly uses preconditioned iterative solvers like CG or GMRes.

3.2 Image Read-In and Noise Simulation

To read-in images we use the CIMG Library - C++ Template Image Processing Toolkit [21]. The single header file library CIMG is an open source project to provide easy handling and processing of images. It is capable of reading-in standard image formats (e.g. .png, .tif, .jpg) and of adding certain types of noise. Each pixel is accessible by its coordinates utilizing the data type image, whose constructor gets the filename containing the image location. Moreover, image provides a method to add noise to the data with a given noise level for different noise-types. In particular, Gaussian noise with noise level (variance) γ and mean 0 and salt-and-pepper noise S with noise level $s \in [0,1]$, which corrupts an original image $\hat{g}$ by

$$S\hat{g}(x) = \begin{cases} \min \hat{g} & \text{with probability } s/2 \\ \max \hat{g} & \text{with probability } s/2 \\ \hat{g}(x) & \text{with probability } 1-s \end{cases},$$

are available.

We define two adapter classes within the header file src/datafunction.hh to convert either the data type image or an algebraic expression into a Dune-Fem discrete function. They derive from the GridFunctionAdapter that creates discrete functions from an evaluation method on a mesh element. These functions are defined on the unit square $[0,1] \times [0,1]$, as we can scale any rectangular image accordingly. The image resolution and aspect ratio relate to the level of initial refinement and to the number of cells in each direction. So n uniform refinements with $l \times k$ initial cells lead to an image resolution of size $l2^n \times k2^n$ square pixels, where every square is composed of two triangles and l/k corresponds to the aspect ratio. This is done in the gridfile data/unitsquare2d.dgf which describes the unit square using the Dune DGF – Interval [6] nomenclature.

The artificial addition of noise (NSM_USE_NOISE=**true**) is only needed for demonstration purposes as in real-world application the image is already corrupted by noise due to certain physical processes. We currently add two independent types of noise, where every noise implemented by CImg can be applied. Noise type and noise level are controlled by a set of parameters defined in the parameter file. For instance, setting nsm.noiseType1 to 0 leads to Gaussian noise and setting it to 2 corresponds to salt-and-pepper noise. Setting any noise level parameter to 0 disables the respective noise. For other noise types and more information consult the documentation of CImg .

3.3 Implementation of the Primal-Dual Algorithm

The primal-dual algorithm from Section 2.2 is implemented in the file src/nsm.cc, where some of its subroutines, which depend on the continuity of the discrete spaces $\mathcal{U}$ and $\mathcal{P}$, are outsourced into the file src/phc.hh. The template class ProjectionHelperClass uses partial template specialization to correctly define the methods calculateZ() and entitywiseProjection().

The method phc.entitywiseProjection() realizes equations (10) and (11). Solving eq. (10) is relatively easy, as all steps are provided by Dune-ACFem, e.g. for continuous $\mathcal{U}$ and $\mathcal{P}$:

```
void entitywiseProjection ( const ForwardDiscreteFunctionType & uBar, AdjointDiscreteFunctionType
    & p )
  {
    // the gradient model defines the weak gradient using continuous test-functions
    // from the space of p
    auto gradU = gradientModel(uBar);
    auto Dbc0 = dirichletZeroModel(p);
    auto mass = massModel(p);
    // we solve p = sigma * nabla u + p
    // with dirichlet zero boundary
    auto model = mass - sigma_ * gradU - p + Dbc0 ;
    // Testspace = AdjointDiscreteFunctionSpaceType = space of p
    typedef EllipticFemScheme<AdjointDiscreteFunctionType , DecltType(model)> SchemeType;
    // p is the returned solution
    SchemeType scheme(p, model);
    scheme.solve();
```

In the above case, for given $\vec{p}$ and $\bar{u}$ the equation

$$0 = \langle \bar{p}, \phi \rangle - \sigma \langle \bar{u}, -\operatorname{div} \phi \rangle - \langle \vec{p}, \phi \rangle \quad \forall \phi \in \mathcal{P}$$

with dirichlet-zero boundary conditions is solved with respect to $\bar{p}$. The solution is written into the variable $\vec{p}$.

Projecting $\vec{p}$ to be feasible (eq. (11)) is not in the features of Dune-ACFem. So we have to use the features of Dune-Fem directly. We choose to do the projection entity-wise, i.e. we iterate over all entities of the grid. On each entity we iterate over the degrees of freedom and if $|\vec{p}|_2 > 1$ holds, we restrict the corresponding value to length 1 in the same direction. If the polynomial order of $\mathcal{P}$ is less or equal than 1, this implies the necessary condition $\| |\vec{p}|_2 \|_{L^\infty} \leq 1$. For the space $\mathcal{S}^1(\mathcal{T}_h)$ this implies the application of the nodal interpolant $\mathcal{I}_h$.

The calculation of z (eq. (13)) translates very nicely into code. We use the L2Projection of DUNE-ACFEM and its ability to linearly combine discrete functions. If $\mathcal{P}$ is continuous, the weak form of eq. (13) is

$$\langle z_k, \psi \rangle = \frac{1}{1 + 2\tau\alpha_2} \langle u_k + \tau \operatorname{div} \vec{p}_{k+1} + 2\tau\alpha_2 g, \psi \rangle \quad \forall \psi \in \mathcal{U}$$

and translates into code in the following way:

```
void calculateZ (const AdjointDiscreteFunctionType &p, const ForwardDiscreteFunctionType & uOld,
    ForwardDiscreteFunctionType& z)
  {
    auto divP  = divergence(p);
    L2Projection(1./(1.+tau_*lambda_2_) * ( tau_ * divP + uOld + tau_ * lambda_2_ * projG_), z);
  }
```

For $\mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h)^d$ we use the weak divergence to shift the derivative to the test-space $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$. As above we use the basic Models of DUNE-ACFEM, in particular the massModel and the weakDivergenceModel to implement eq. (13) as follows:

```
void calculateZ (const AdjointDiscreteFunctionType &p, const ForwardDiscreteFunctionType & uOld,
    ForwardDiscreteFunctionType& z)
  {
    //get the mass model of the Forward space
    auto U_Phi = massModel(projG_.space());
    //calculate z
    auto weakDiv_P = weakDivergenceModel(p);
    auto projModel = U_Phi -1./(1.+tau_*lambda_2_) * ( tau_ * weakDiv_P + uOld + tau_ * lambda_2_
        * projG_);
    typedef EllipticFemScheme<ForwardDiscreteFunctionType, decltype(projModel)> SchemeType;
    SchemeType scheme(z, projModel);
    scheme.solve();
  }
```

Note that in contrast to the implementation of the method entitywiseProjection() we construct the massModel from a discrete space instead of a discrete function. This is explicitly allowed by DUNE-ACFEM as the object simply has to provide the data type for the test-function space.

For the case of $\mathcal{P} = \mathcal{L}^0_N(\mathcal{S}_h)^d$ and $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$ the summand weakDiv_p is turned into a functional. This functional is called EdgeWeakDivergenceFunctional and is contained in the file edgeweakdivergencefunctional.hh . The main implementation is done in the coefficients() method, that implements the application of the functional to the basis functions in DUNE-FEM notation.
In the resulting code for the method calculateZ(), (see code example above) the main difference is that weakDivergenceModel(p) is replaced by edgeWeakDivergenceFunctional(p). Additionally weakDiv_p in the definition of projModel has been moved outside the brackets, as functions are not allowed to be added to functionals outside a model (see below).

```
    auto weakDiv_P = edgeWeakDivergenceFunctional(p);
    auto projModel = U_Phi -1./(1.+tau_*lambda_2_) * ( uOld + tau_ * lambda_2_ * projG_) -
        tau_/(1.+tau_*lambda_2_) * weakDiv_p;
```

3.4 Adaptive Refinement

Adaptive refinement is initiated by a positive parameter nsm.localRefine. This parameter denotes the number of additional local refinements to be done in the initialization phase in addition to the uniform refinements. If it is set to ≤ 0, no local refinement will be performed. The grid is locally refined at discontinuities of the piecewise constant datum $g \in \mathcal{L}^0(\mathcal{T}_h)$ in the following way: Given an entity E and its neighbour N, if

$$|g_{|N} - g_{|E}| > ADAPTTOL \tag{20}$$

holds, then E is marked for refinement, where the value of *ADAPTTOL* is given by the parameter nsm.adaptTolerance. We do not refine during the iterations of the primal-dual algorithm, but keep the mesh static. So the refinement increases the resolution of discontinuities and hence drastically improves the projection $\mathbb{P}g$. This is implemented in the method adaptGrid() in the file nsm.cc. We use the grid manager DUNE-ALUGRID [1], which is capable of handling the needed conforming, parallel, adaptive, triangular grids.

4 Numerical Experiments

To reproduce the data of the experiments of this section, consult Appendix A and follow the described steps.

4.1 Comparison of Discrete Spaces

We investigate for different discrete spaces $\mathcal{U}$ and $\mathcal{P}$ the behaviour of our algorithm with respect to the non-smooth minimization problem (1) considering the following setups

$$\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h), \mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d, \qquad \mathcal{U} = \mathcal{L}^0(\mathcal{T}_h), \mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d,$$
$$\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h), \mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h)^d, \qquad \mathcal{U} = \mathcal{L}^0(\mathcal{T}_h), \mathcal{P} = \mathcal{L}^0_N(\mathcal{S}_h)^d.$$

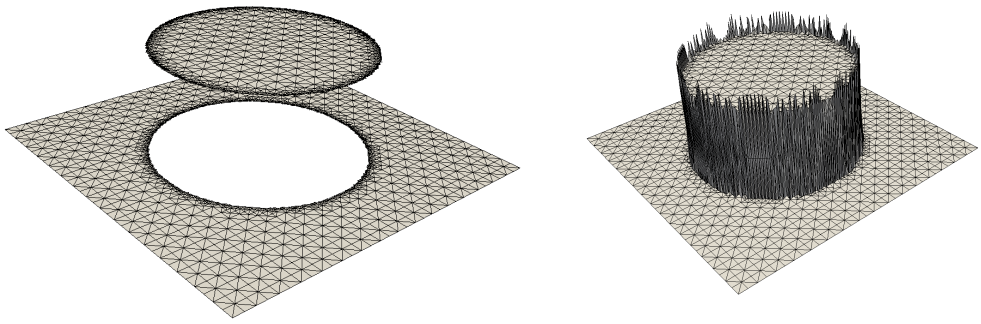
The setups using $\mathcal{P} = \mathcal{L}^0_N(\ldots)^d$ are motivated by the equivalence of the mixed formulation and the primal formulation for these discrete settings. For the case $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h), \mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h)^d$ we even have guaranteed convergence to the continuous formulation, see Theorem 2.1. The setting $\mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d$ is motivated by [5], where it is shown that for $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$, $\mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d$ the discrete pre-dual of the functional J_{0,α_2} Γ-converges to the continuous one. However, setting $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$ does not seem necessary for the proof there, but can be for example replaced by choosing $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$. Therefore we also investigate the case $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$.

For this set of experiments we choose a similar example as in [5], i.e., the observed data is 1 on a disk of radius 0.3 and 0 elsewhere. Hence, we choose the discrete datum g to be given as a piecewise constant approximation of the function

$$f(x) = \begin{cases} 1, & \text{if } |x - (0.5, 0.5)| \leq 0.3 \\ 0, & \text{else.} \end{cases}$$

Note that this datum changes under grid refinement. The cost parameters are set to $\alpha_1 = 10$ and $\alpha_2 = 20$, the stopping tolerance to 10^{-2}, and the adaptation tolerance to 0.1. For the $\mathcal{L}^0/\mathcal{L}^0$ case we choose the tolerance 10^{-4} as we use an extension of $\mathcal{P}$ into Ω to calculate the part of $\vec{p}$ in the stopping criterion $\frac{\|u_{k+1}-u_k\|_{L^2}}{\tau} + \frac{\|\vec{p}_{k+1}-\vec{p}_k\|_{L^2}}{\sigma} < TOL$. We do $a = 3$ uniform refinements and $b = 5$ additional local refinements. As convergence is guaranteed, if condition (16) holds, and since we know that in general $\|\nabla u\|_{L^2} < Ch\|u\|_{L^2}$ for $u \in \mathcal{U}$, the step sizes are automatically set to $\tau = \sigma = L * 2^{-(a+b)}$ with a constant L. For the $\mathcal{L}^0/\mathcal{L}^0$ case, numerics indicate, that it is possible to even set $\tau = \sigma = L * 2^{-(a+b)/2}$. Note that the choice of L does not only depend on the operator norm of the gradient on the domain, but also on the number of elements in the initial grid. Here we choose $L = 0.19$. The overrelaxation parameter θ is chosen to be 1.

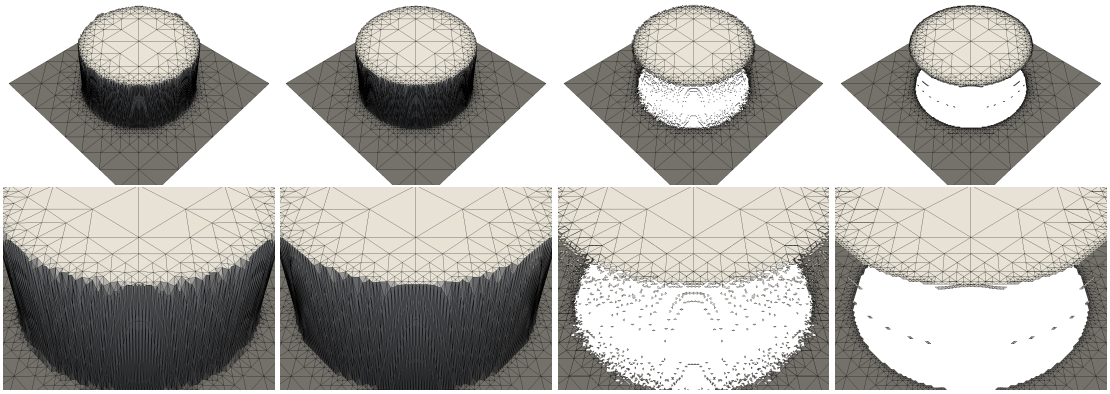
By imposing all equations weakly the primal-dual algorithm de facto uses the L^2-projection $\mathbb{P}g \in \mathcal{U}$ instead of the datum $g \in \mathcal{L}^0(\mathcal{T}_h)$. Figure 1 shows that in the case of $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$ by projecting a discontinuous function onto a continuous space we introduce an additional error in contrast to the case $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$. The over- and undershoots of the continuous projected data do not pose a problem, because they are regularized over the course of the algorithm, as we can see in Figure 2b. Comparing Figure 2a with 2b and Figure 2c with 2d we see that choosing $\mathcal{P} = \mathcal{S}^1(\mathcal{T}_h)^d$ smears out jumps, i.e., the discontinuities are less accurately approximated. This is due to the fact, that $\mathcal{P}$ is continuous and its basis functions have a larger support than if it were



(a) $g = \mathbb{P}g$, as $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$

(b) $\mathbb{P}g$ for $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$

Figure 1: The Projection of the piecewise constant datum g onto the space $\mathcal{U}$.



(a) $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$, $\mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d$

(b) $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$, $\mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h)^d$

(c) $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$, $\mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d$

(d) $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$, $\mathcal{P} = \mathcal{L}^0_N(\mathcal{S}_h)^d$

Figure 2: The discrete minima u^* of the functional $J_{10,20}$

discontinuous. Additionally $\mathcal{P} = \mathcal{S}^1(\mathcal{T}_h)^d$ is not the space for which the primal and the mixed formulation are equivalent leading to a worse approximation of $|Dv|(\Omega)$. This explains why the value of the functional in Figures 3 and 4 for these variants is higher than choosing $\mathcal{P} = \mathcal{L}^0_N(\dots)^d$. In particular this worsens the quality of the solutions obtained with $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h), \mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d$, as now multiple elements at the jump are hanging mid-air (Figure 2c), which drastically increases the total variation.

Figure 3 depicts the evolution of the energy J_{α_1,α_2} during the iterations for the considered space pairings. We observe that J_{α_1,α_2} is not monotonically decreasing but stagnates at a minimal value after a certain number of iterations (note that the scale of the number of iterations in Figure 3 is logarithmic). This demonstrates, that in all these settings the algorithm converges to a stationary point of the respective discrete problem. Due to the different combinations of spaces it is clear that the stationary points in general do not coincide and hence the minimal energy is different. The combination $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h), \mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h)^d$ yields the best result, as its final energy is the smallest among the considered cases.

We observe that choosing $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$ results in a decreasing energy for $h \to 0$, while for

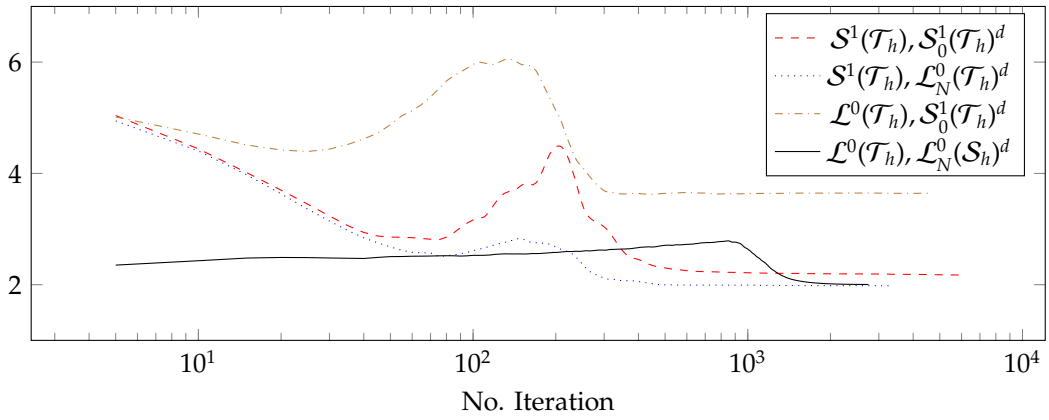


Figure 3: The value of the functional J_{α_1,α_2} over the course of the algorithm.

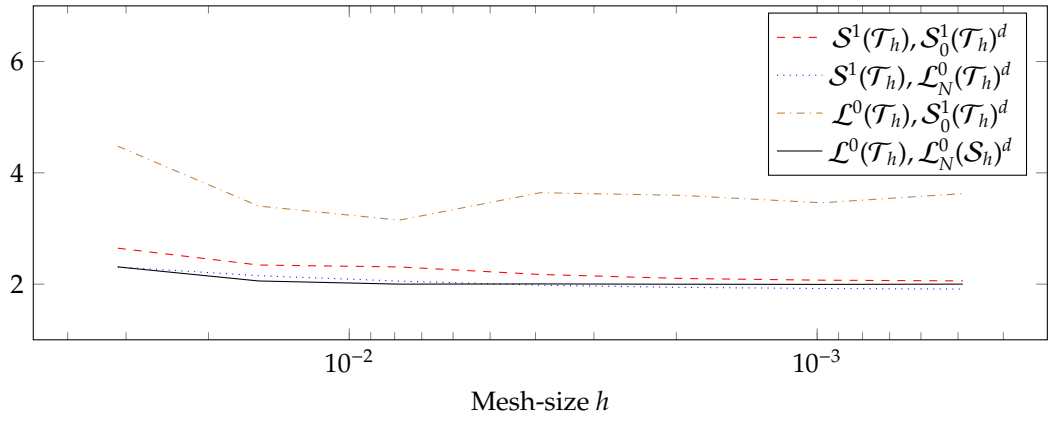


Figure 4: Final energy for $h = 2^{-5}, \ldots, 2^{-11}$

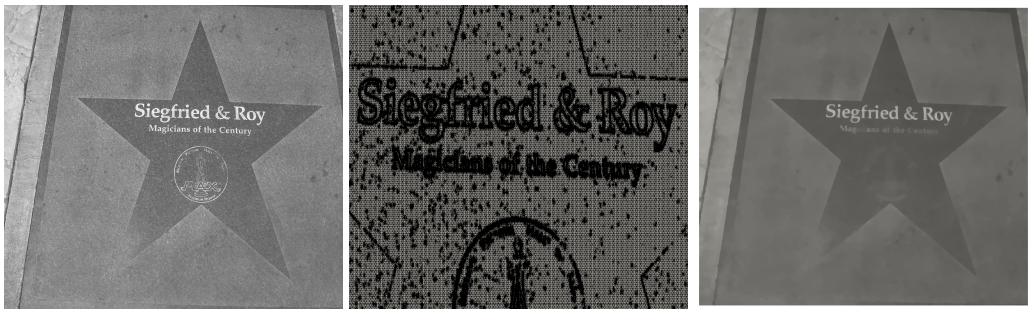
$\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$ this is not the case (cf. Figure 4). More precisely for the case $\mathcal{L}^0(\mathcal{T}_h), \mathcal{S}^1_0(\mathcal{T}_h)^d$ the energy seems to oscillate in a certain sense and for $\mathcal{L}^0(\mathcal{T}_h), \mathcal{L}^0_N(\mathcal{S}_h)^d$ the energy stays almost constant. This is due to the fact that, while the approximation of the circle gets better, the total variation does not diminish for $\mathcal{L}^0$. In general approximating functions of bounded variation is not possible with piecewise constant functions on a triangulation, since the length of discontinuities may not diminish over refinement [3]. This is different for the case $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$, where discontinuities of functions, that are not representable on the triangulation, can be better approximated.

4.2 Image Denoising

Here we demonstrate the denoising capability of the algorithm. Motivated by the above experiments we choose the spaces to be $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$, $\mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h)^d$. The considered image with values in $[0, 1]$ is first corrupted by Gaussian white noise with variance 0.1 and then salt-and-pepper noise with $s = 0.1$ is added. The obtained image is shown in Figure 5a. In order to reconstruct the image we choose $\alpha_1 = 250$, $\alpha_2 = 150$ in (1), and perform our primal-dual algorithm with the tolerance set to 10^{-2}, $\theta = 1$ and $\sigma = \tau = 2^{-8}$. To resolve the 256×256 pixel-sized picture we apply 8 uniform refinements and no additional local refinement. In Figure 5b we depict the output of our algorithm. The result is reasonably smoothed due to the total variation regularization, while discontinuities are still preserved.

(a) Noisy image (b) Result of the algorithm

Figure 5: A corrupted image before and after applying the algorithm



(a) Real image (b) Part of the mesh (c) Result of the algorithm

Figure 6: A real image before and after applying the algorithm

As a second example we run the algorithm on a much finer image, that has a resolution of 2576×1920 pixels (Figure 6a). The spaces are chosen as above, the cost parameters are $\alpha_1 = \alpha_2 = 500$, we run the image on an initial grid of 161×120 squares, each subdivided in two triangles and initiate 2 initial global refinements and 2 local Refinements. So locations with full refinement a triangle is half the size of a pixel. A part of the mesh is depicted in Figure 6b, where one may observe that the discontinuities are captured by the refinement, as intended. The resulting image (Figure 6c) behaves as expected, the noisy structure of the stone is reduced to a minimum, but we also lose some details inside the stars, that we may have wanted to keep. This may be attributed to the choice of the parameters α_1 and α_2. We note that in this example as well as in the previous one, these parameters are chosen at will, but not optimal. For an optimal choice of parameters, we refer the reader to [19]. Moreover it may be of interest to choose local cost parameters, as in [9, 12, 20].

4.3 Strong Scaling

We do a strong scaling experiment on the same datum as in Section 4.1 using the spaces $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$ and $\mathcal{P} = \mathcal{L}^0(\mathcal{T}_h)^d$. The computation is executed on a 32 core shared memory system. The grid manager DUNE-ALUGRID balances the computational load on the initial grid by partitioning it onto the different processors. So we cannot expect the algorithm to scale if the initial mesh is to coarse. E.g. if there are only two initial elements, only two processors can get partitions that are non-empty. Consequently we discretize the unit square by 8×8 elements, see `data/finecube_2d.dgf`. As the grid is already fine, we do not need as many uniform refinements to reach a good resolution, so

we do $a = 3$ uniform and another $b = 3$ local refinements. This results in a different operator norm of the gradient on the initial grid, so we have to set the corresponding parameter nsm.constantL to 0.02 for the stepsizes $\tau = \sigma = L * 2^{-(a+b)}$ to be computed correctly. The tolerance is set to 10^{-2} and $\alpha_1 = \alpha_2 = 150$ to reach a short runtime.

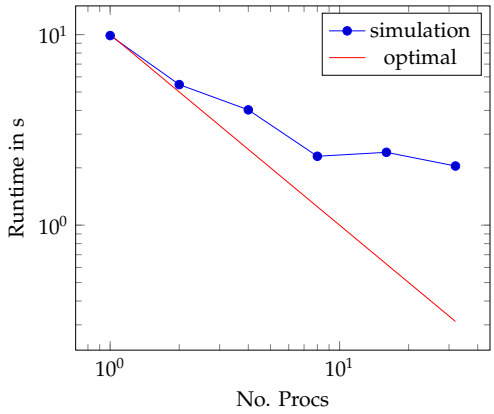


Figure 7: Strong scaling of the algorithm.

Figure 7 indicates that the strong scaling at least up to 8 cores is quite good and afterwards it stagnates, which is probably due to the small grid and not enough workload. We would have to increase the number of initial cells to improve the scaling further. The important part about this result is, that the parallelism is done inside Dune-ACFem and we spent almost no effort to parallelize the code. The only line of code needed initializes MPI.

5 Summary and Outlook

We have shown for a certain discretization that the associated minimizer converges to the continuous one. The primal-dual algorithm used here is implemented very conveniently within Dune-ACFem. The flexibility of Dune-Fem allows us to easily exchange discrete spaces and easily set parameters. Also the algorithm is now intrinsically parallel by domain decomposition with a decent strong scaling. Future research includes implementing a semi-smooth Newton method in Dune-ACFem to increase convergence speed even for highly adaptive grids.

Acknowledgements

The authors would like to thank Claus-Justus Heine for discussions on implementation issues and Dune-ACFem support. Moreover, Martin Alkämper would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/2) at the University of Stuttgart.

Appendix A - Installation Instructions

The program is designed to work in a Linux-environment. It may as well work on a unix machine. To install the program, first download the Dune modules, Dune-common, Dune-geometry, Dune-grid, Dune-istl, Dune-localfunctions, Dune-Fem, Dune-ALUGrid and Dune-ACFem from gitlab.dune-project.org and non–smooth–minization from http://www.ians.uni-stuttgart.de/nmh/downloads. For the Dune- modules checkout the branch releases/2.4. Put all the projects in a directory as direct subfolders. Use your config file config.opts to run the command

./dune=common/bin/dunecontrol ––opts=config.opts ––module=non–smooth–minization all

An example config file example.opts is provided in the main directory of the DUNE project non−smooth−minization .
Now DUNE is installed and the executable nsm from the subdirectory src should have been built within the cmake build directory. The scripts example1.sh to example5.sh produce the results presented in this paper. They require a python interpreter and example3.sh and example5.sh require MPI. The cmake build directory is assumed to be named build−cmake. This can be adjusted by modifying the BUILDDIR variable inside the scripts. The scripts are simple, so changes should be easy. They basically consist of changing directories, compiling the executable nsm, executing it with the right set of parameters and parsing the output into suitable directories. There are two types of output. Console output from std::out is parsed into a file output.graph in a subdirectory of src, which describes the value of the functional and the value of all its parts with respect to both g and $\mathbb{P}g$ and a file output.parameters, where the applied parameters and other output can be looked up. Console output from std::err is displayed on the console. The other type of output is in the cmake build directory located within the directory output in suitable subdirectories. These contain a set of .vtu/.vtk files (readable with Paraview), that contain the datum g (called "image"), the projection $\mathbb{P}g$ (called "projection of g"), the solution (called "u0") and the adjoint variable (componentwise "p0", "p1"). It may take some time to run the scripts. To get faster results, simply edit the LOCREF or INITREF variable of the bash scripts or lower the tolerance parameter nsm.tolerance.

Appendix B - Parameters

There are two types of parameters: run time parameters and compile time parameters.

- Run time parameters can be overloaded on the command line by
  ./nsm nsm.parameter:value ... .
  The default values are set inside the parameter file located at data/parameter. This file is copied into the cmake build directory at compile time. So parameters changed in the build directory will be overwritten when recompiling. Parameters with prefix fem, or istl belong to DUNE-FEM and DUNE-ISTL respectively, and are explained in their documentation. The specific parameters of this algorithm are prefixed nsm. The extensive list of run time parameters reads:

nsm.theta	Overrelaxation parameter θ
nsm.maxIt	Maximum number of iterations of the algorithm
nsm.tolerance	The algorithm breaks if the tolerance is reached
nsm.outputStep	Output every nth step
nsm.constantL	If ! NSM_SET_STEPSIZE, $\tau = \sigma = L * 2^{-(a+b)}$
nsm.tau	Stepsize τ
nsm.sigma	Stepsize σ
nsm.lambda_1	L^1-data term cost coefficient α_1
nsm.lambda_2	L^2-data term cost coefficient α_2
nsm.image	Filename of the image to be read in
nsm.initialRefinements	Number of initial uniform refinements a
nsm.localRefine	Number of initial adaptive refinements b
nsm.adaptTolerance	Adaptive tolerance
nsm.noiseType1	The CIMG type of noise to be applied first
nsm.noiseLevel1	The CIMG noise level of the first noise
nsm.noiseType2	The CIMG type of noise to be applied second
nsm.noiseLevel2	The CIMG noise level of the second noise

- Compile time parameters have to be declared when compiling. The location is in the file CMakelists.txt. As they are cmake -cache variables, they can be redefined in the usual way (see e.g. example1.sh). They determine what kind of problem to treat and which discrete

spaces are to be used.

NSM_SET_STEPSIZE	If true, τ and σ are set manually
NSM_USE_IMAGE	If true, image is used instead of geometric expression
NSM_USE_NOISE	If true, noised imaged is used, requires NSM_USE_IMAGE
NSM_U_DISCONT	If true, $\mathcal{U} = \mathcal{L}^0(\mathcal{T}_h)$, else $\mathcal{U} = \mathcal{S}^1(\mathcal{T}_h)$.
NSM_P_DISCONT	If false, $\mathcal{P} = \mathcal{S}^1_0(\mathcal{T}_h)^d$, else $\mathcal{P} = \mathcal{L}^0_N(\mathcal{T}_h)^d$ or $\mathcal{L}^0_N(\mathcal{S}_h)^d$ (depending on $\mathcal{U}$) .

References

[1] Martin Alkämper, Andreas Dedner, Robert Klöfkorn, and Martin Nolte. The dune-alugrid module. *Archive of Numerical Software*, 4(1):1–28, 2016.

[2] Luigi Ambrosio, Nicola Fusco, and Diego Pallara. *Functions of Bounded Variation and Free Discontinuity Problems*. Oxford Mathematical Monographs. The Clarendon Press, Oxford University Press, New York, 2000.

[3] Sören Bartels. Total variation minimization with finite elements: convergence and iterative solution. *SIAM Journal on Numerical Analysis*, 50(3):1162–1180, 2012.

[4] Sören Bartels. Broken sobolev space iteration for total variation regularized minimization problems. *IMA Journal of Numerical Analysis*, page drv023, 2015.

[5] Sören Bartels. Error control and adaptivity for a variational model problem defined on functions of bounded variation. *Mathematics of Computation*, 84(293):1217–1240, 2015.

[6] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger, and O. Sander. A generic grid interface for parallel and adaptive scientific computing. II. Implementation and tests in DUNE. *Computing*, 82(2-3):121–138, 2008.

[7] P. Bastian, M. Blatt, A. Dedner, Ch. Engwer, R. Klöfkorn, S.P. Kuttanikkad, M. Ohlberger, and O. Sander. The Distributed and Unified Numerics Environment (DUNE). In *Proc. of the 19th Symposium on Simulation Technique in Hannover, Sep. 12 - 14*, 2006.

[8] Susanne C. Brenner and Ridgway Scott. *The Mathematical Theory of Finite Element Methods*, volume 15. Springer Science & Business Media, 2008.

[9] Chung Van Cao, Juan Carlos De los Reyes, and Carola-Bibiane Schönlieb. Learning optimal spatially-dependent regularization parameters in total variation image restoration. *arXiv preprint arXiv:1603.09155*, 2016.

[10] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[11] A. Dedner, R. Klöfkorn, M. Nolte, and M. Ohlberger. A Generic Interface for Parallel and Adaptive Scientific Computing: Abstraction Principles and the DUNE-FEM Module. *Computing*, 90(3–4):165–196, 2010.

[12] Yiqiu Dong, Michael Hintermüller, and M. Monserrat Rincon-Camacho. Automated regularization parameter selection in multi-scale total variation models for image restoration. *J. Math. Imaging Vision*, 40(1):82–104, 2011.

[13] Enrico Giusti. *Minimal Surfaces and Functions of Bounded Variation*, volume 80 of *Monographs in Mathematics*. Birkhäuser Verlag, Basel, 1984.

[14] Claus-Justus Heine. Dune-acfem documentation. *http://www.ians.uni-stuttgart.de/nmh/stage/documentation/software/dune-acfem/doxygen/*, 2014.

[15] Michael Hintermüller and Karl Kunisch. Total bounded variation regularization as a bilaterally constrained optimization problem. *SIAM Journal on Applied Mathematics*, 64(4):1311–1333, 2004.

[16] Michael Hintermüller and Andreas Langer. Subspace correction methods for a class of nonsmooth and nonadditive convex variational problems with mixed L^1/L^2 data-fidelity in image processing. *SIAM J. Imaging Sci.*, 6(4):2134–2173, 2013.

[17] Michael Hintermüller and Monserrat Rincon-Camacho. An adaptive finite element method in L^2-TV-based image denoising. *Inverse Problems and Imaging*, 8(3):685–711, 2014.

[18] Michael Hintermüller and Georg Stadler. An infeasible primal-dual algorithm for total bounded variation-based inf-convolution-type image restoration. *SIAM J. Sci. Comput.*, 28(1):1–23, 2006.

[19] Andreas Langer. Automated parameter selection in the L^1-L^2-TV model for removing Gaussian plus impulse noise. *accepted by Inverse Problems*, 2016.

[20] Andreas Langer. Automated parameter selection for total variation minimization in image restoration. *Journal of Mathematical Imaging and Vision*, 57(2):239–268, 2017.

[21] David Tschumperlé. The cimg library - c++ template image processing toolkit. *http://www.cimg.eu/reference/*.

