Automating the formulation and resolution of convex variational problems: applications from image processing to computational mechanics

Jeremy Bleyer

Automating the formulation and resolution of convex variational problems: applications from image processing to computational mechanics

JEREMY BLEYER, Laboratoire Navier UMR 8205 (ENPC-IFSTTAR-CNRS), Université Paris-Est, France

Convex variational problems arise in many fields ranging from image processing to fluid and solid mechanics communities. Interesting applications usually involve non-smooth terms which require well-designed optimization algorithms for their resolution. The present manuscript presents the Python package called `fenics_optim` built on top of the FEniCS finite element software which enables to automate the formulation and resolution of various convex variational problems. Formulating such a problem relies on FEniCS domain-specific language and the representation of convex functions, in particular non-smooth ones, in the conic programming framework. The discrete formulation of the corresponding optimization problems hinges on the finite element discretization capabilities offered by FEniCS while their numerical resolution is carried out by the interior-point solver Mosek. Through various illustrative examples, we show that convex optimization problems can be formulated using only a few lines of code, discretized in a very simple manner and solved extremely efficiently.

CCS Concepts: • **Mathematics of computing → Convex optimization**; • **Computing methodologies → Symbolic and algebraic algorithms**;

Additional Key Words and Phrases: convex optimization, conic programming, finite element method, FEniCS

ACM Reference Format:
Jeremy Bleyer. 2019. Automating the formulation and resolution of convex variational problems: applications from image processing to computational mechanics. 1, 1 (November 2019), 31 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Convex variational problems represent an important class of mathematical abstractions which can be used to model various physical systems or provide a natural way of formulating interesting problems in different areas of applied mathematics. Moreover, they also often arise as a relaxation of more complicated non-convex problems. Optimality conditions of constrained convex variational problems correspond to variational inequalities which have been the topic of a large amount of work in terms of analysis or practical applications [25, 30, 45].

In this work, we consider convex variational problems defined on a domain $\Omega \subset \mathbb{R}^d$ ($d = 2, 3$ for typical applications) with convex constraints of the following kind:

$$\begin{aligned} \inf_{u \in V} \quad & J(u) \\ \text{s.t.} \quad & u \in \mathcal{K} \end{aligned} \tag{1}$$

where u belongs to a suitable functional space V, J is a convex function and $\mathcal{K}$ a convex subset of V. Some variational inequality problems are formulated naturally in this framework such as the classical Signorini obstacle problem in which $\mathcal{K}$ encodes the linear inequality constraint that a membrane displacement cannot interpenetrate a fixed obstacle (see section 3.1). An important

Author's address: Jeremy Bleyer, jeremy.bleyer@enpc.fr, Laboratoire Navier UMR 8205 (ENPC-IFSTTAR-CNRS), Université Paris-Est, 6-8 av Blaise Pascal, Cité Descartes, Champs-sur-Marne, France, 77455.

class of situations concerns the case where J can be decomposed as the sum of a smooth and a non-smooth term. Such a situation arises in many variational models of image processing problems such as image denoising, inpainting, deconvolution, decomposition, etc. In some cases, such as limit analysis problems in mechanics for instance, smooth terms in J are absent so that numerical resolution of (1) becomes very challenging [27, 45]. Important problems in applied mathematics such as optimal control [36] or optimal transportation [9, 41, 42, 46] can also be formulated, in some circumstances, as convex variational problems. This is also the case for some classes of topology optimization problems [10], which can also be extended to non-convex problems involving integer optimization variables [28, 48]. Finally, robust optimization in which optimization is performed while taking into account uncertainty in the input data of (1) has been developed in the last decade [7, 8]. It leads, in some cases, to tractable optimization problems fitting the same framework, possibly with more complex constraints.

The main goal of this paper is to present a numerical toolbox for automating the formulation and the resolution of discrete approximations of (1) using the finite-element method. The large variety of areas in which such problems arise makes us believe that there is a need for a versatile tool which will aim at satisfying three important features:

- straightforward formulation of the problem, mimicking in particular the expression of the continuous functional;
- automated finite-element discretization, supporting not only standard Lagrange finite elements but also DG formulations and $H(\mathrm{div})/H(\mathrm{curl})$ elements;
- efficient solution procedure for all kinds of convex functionals, in particular non-smooth ones.

In our proposal, the first two points will rely extensively on the versatility and computational efficiency of the FEniCS open-source finite element library [2, 32]. FEniCS is now an established collection of components including the DOLFIN C++/Python Interface [34, 35], the Unified Form Language [1, 3], the FEniCS Form Compiler [31, 33], etc. Using the high-level DOLFIN interface, the user is able to write short pieces of code for automating the resolution of PDEs in an efficient manner. For all these reasons, we decided to develop a Python package called `fenics_optim` as an add-on to the FEniCS library. We will also make use of Object Oriented Programming possibilities offered by Python for defining easily our problems (see 4.4). Our proposal can therefore be considered to be close to high-level optimisation libraries based on *disciplined convex programming* such as CVX[1] for instance. However, here we really concentrate on the integration within an efficient finite-element library offering symbolic computation capabilities. As mentioned later, integration with other high-level optimisation libraries will be an interesting development perspective.

Concerning the last point on solution procedure, we will here rely on the state-of-the art conic programming solver named Mosek [38], which implements extremely efficient primal-dual interior-point algorithms [4]. Let us mention first that there is no ideal choice concerning solution algorithms which mainly depends on the desired level of accuracy, the size of the considered problem, the sparsity of the underlying linear operators, the type of convex functionals involved, etc. In particular, in the image processing community, first-order proximal algorithms are widely used since they work well in practice for large scale problems discretized on uniform grids [41]. Moreover, high accuracy on the computed solution is usually not required since one aims mostly at achieving some decrease of the cost function but not necessarily an accurate computation of the optimal point. In contrast, interior-point methods can achieve a desired accuracy on the solution in polynomial time and, in practice, in quasi-linear time since the number of final iterations is

[1] http://cvxr.com/cvx/

often observed to be nearly independent on the problem size. However, as a second-order method, each iteration is costly since it requires to solve a Newton-like system. Iterative solvers are also difficult to use in this context due to the strong increase of the Newton system conditioning when approaching the solution. As result, such solvers usually rely on direct solvers for factorizing the resulting Newton system, therefore requiring important memory usage. Nevertheless, interior-point solvers are extremely robust and quite efficient even compared to first order methods in some cases. For these reasons, the present paper will not focus on comparing different solution procedures and we will use only the Mosek solver but including first-order algorithms in the `fenics_optim` package will be an interesting perspective for future work. Providing interfaces to other interior-point solvers, especially open-source ones (CVXOPT, ECOS, Sedumi, etc.), is also planned for future releases.

The manuscript is organized as follows: section 2 introduces the conic programming framework and the concept of conic representable functions. The formulation and discretization of convex variational problems is discussed in section 3 by means of a simple example. Section 4 discusses further aspects by considering a more advanced example. Finally, 5 provides a gallery of illustrative examples along with their formulation and some numerical results.

The `fenics_optim` package can be downloaded from https://gitlab.enpc.fr/navier-fenics/fenics-optim. It contains test files as well as demo files corresponding to the examples discussed in the present paper.

2 CONIC PROGRAMMING FRAMEWORK

2.1 Conic programming in Mosek

The Mosek solver is dedicated to solving problems entering the *conic programming framework* which can be written as:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{b}_l \leq \mathbf{A}\mathbf{x} \leq \mathbf{b}_u \\ & \mathbf{x} \in \mathcal{K} \end{aligned} \tag{2}$$

where vector $\mathbf{c}$ defines a linear objective functional, matrix $\mathbf{A}$ and vectors $\mathbf{b}_u, \mathbf{b}_l$ define linear inequality (or equality if $\mathbf{b}_u = \mathbf{b}_l$) constraints and where $\mathcal{K} = \mathcal{K}^1 \times \mathcal{K}^2 \times \ldots \times \mathcal{K}^p$ is a product of cones $\mathcal{K}^i \subset \mathbb{R}^{d_i}$ so that $\mathbf{x} \in \mathcal{K} \Leftrightarrow \mathbf{x}^i \in \mathcal{K}^i \; \forall i = 1, \ldots, p$ where $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^p)$. These cones can be of different kinds:

- $\mathcal{K}^i = \mathbb{R}^{d_i}$ i.e. no constraint on $\mathbf{x}^i$
- $\mathcal{K}^i = (\mathbb{R}^+)^{d_i}$ is the positive orthant i.e. $\mathbf{x}^i \geq 0$
- $\mathcal{K}^i = \mathcal{Q}_{d_i}$ the quadratic Lorentz cone defined as:

$$\mathcal{Q}_{d_i} = \{\mathbf{z} \in \mathbb{R}^{d_i} \text{ s.t. } \mathbf{z} = (z_0, \bar{\mathbf{z}}) \text{ and } z_0 \geq \|\bar{\mathbf{z}}\|_2\} \tag{3}$$

- $\mathcal{K}^i = \mathcal{Q}^r_{d_i}$ the rotated quadratic Lorentz cone defined as:

$$\mathcal{Q}^r_{d_i} = \{\mathbf{z} \in \mathbb{R}^{d_i} \text{ s.t. } \mathbf{z} = (z_0, z_1, \bar{\mathbf{z}}) \text{ and } 2z_0 z_1 \geq \|\bar{\mathbf{z}}\|_2^2\} \tag{4}$$

- $\mathcal{K}^i = \mathcal{S}_{d_i}$ is the vectorial representation of the cone of semi-definite positive matrices $\mathbb{S}^+_{n_i}$ of dimension n_i if $d_i = n_i(n_i+1)/2$ i.e.

$$\begin{aligned} \mathcal{S}_{d_i} &= \{\mathbf{vec}(M) \text{ s.t. } M \in \mathbb{S}^+_{n_i}\} \\ \text{where } \mathbb{S}^+_{n_i} &= \{M \in \mathbb{R}^{n_i \times n_i} \text{ s.t. } M = M^{\mathrm{T}} \text{ and } M \geq 0\} \end{aligned} \tag{5}$$

  and in which the **vec** operator is the half-vectorization of a symmetric matrix obtained by collecting in a column vector the upper triangular part of the matrix. The elements are

arranged in a column-wise manner and off-diagonal elements are multiplied by a $\sqrt{2}$ factor. For instance, for a 3×3 matrix:

$$\mathbf{vec}(M) = (M_{1,1}, \sqrt{2}M_{1,2}, \sqrt{2}M_{1,3}, M_{2,2}, \sqrt{2}M_{2,3}, M_{3,3}) \tag{6}$$

Note that this representation is such that $\mathbf{vec}(A)^{\mathrm{T}}\,\mathbf{vec}(B) = \langle A, B\rangle_F$ where $\langle\cdot,\cdot\rangle_F$ denotes the Froebenius inner product of two matrices.

If $\mathcal{K}$ contains only cones of the first two kinds, then the resulting optimization problem (2) belongs to the class of Linear Programming (LP) problems. If, in addition, $\mathcal{K}$ contains quadratic cones $\mathcal{Q}_{d_i}$ or $\mathcal{Q}^r_{d_i}$, then the problem belongs to the class of Second-Order Cone Programming (SOCP) problems. Finally, when cones of the type $\mathcal{S}_{d_i}$ are present, the problem belongs to the class of Semi-Definite Programming (SDP) problems. Note that Quadratic Programming (QP) problems consisting of minimizing a quadratic functional under linear constraints can be seen as a particular instance of an SOCP problem as we will later discuss.

There obviously exist dedicated algorithms for some classes of problem (e.g. the simplex method for LP, projected conjugate gradient methods for bound constrained QP, etc.). However, interior-point algorithms prove to be extremely efficient algorithms for all kinds of problems from LP up to difficult problems like SDP. It also turns out that a large variety of convex problems can be reformulated into an equivalent problem of the previously mentioned categories so that interior-point algorithms can be used to solve, in a robust and efficient manner, a large spectrum of convex optimization problems.

2.2 Conic reformulations

Most conic programming solvers other than Mosek (CVXOPT, Sedumi, SDPT3) use a default format similar to (2). Aiming at optimizing a convex problem using a conic programming framework therefore requires a first reformulation step to fit into format (2). In the following examples, we will consider a purely discrete setting in which optimization variables are in $\mathbb{R}^n$.

2.2.1 L^2-norm constraint. Let us consider the following L^2-norm constraint:

$$\|\mathbf{x}\|_2 \leq 1 \tag{7}$$

This can be readily observed to be the following quadratic cone constraint $(1, \mathbf{x}) \in \mathcal{Q}_{n+1}$. However, for this constraint to fit the general format of (2), one must introduce an additional scalar variable y such that the previous constraint can be equivalently written:

$$\begin{aligned} y &= 1 \\ \|\mathbf{x}\|_2 \leq y &\Leftrightarrow (y, \mathbf{x}) \in \mathcal{Q}_{n+1} \end{aligned} \tag{8}$$

2.2.2 L^1-norm constraint. Let us consider the following L^1-norm constraint:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i| \leq 1 \tag{9}$$

To reformulate this constraint, we introduce n scalar auxiliary variables y_i such that:

$$\begin{aligned} \sum_{i=1}^{n} y_i &= 1 \\ |x_i| &\leq y_i \quad \forall i \end{aligned} \tag{10}$$

then each constraint with an absolute value can be written using two linear inequality constraints $x_i - y_i \leq 0$ and $0 \leq x_i + y_i$.

2.2.3 Quadratic constraint. Let us consider the case of a quadratic inequality constraint such as:

$$\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} \leq b \tag{11}$$

Matrix $\mathbf{Q}$ must necessarily be semi-definite positive for the constraint to be convex. In this case, introducing the Cholesky factor $\mathbf{C}$ of $\mathbf{Q}$ such that $\mathbf{Q} = \mathbf{C}^{\mathrm{T}}\mathbf{C}$, one has:

$$\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} = \frac{1}{2}\|\mathbf{C}\mathbf{x}\|_2^2 \leq b \tag{12}$$

Introducing an auxiliary variable $\mathbf{y}$, the previous constraint can be equivalently reformulated as:

$$\begin{aligned} \mathbf{C}\mathbf{x} - \mathbf{y} &= 0 \\ \|\mathbf{y}\|_2^2 &\leq 2b \end{aligned} \tag{13}$$

Finally adding two others scalar variables z_0 and z_1, we have:

$$\begin{aligned} \mathbf{C}\mathbf{x} - \mathbf{y} &= 0 \\ z_0 &= b \\ z_1 &= 1 \\ \|\mathbf{y}\|_2^2 &\leq 2z_0z_1 \end{aligned} \tag{14}$$

where the last constraint is also the rotated quadratic cone constraint $(z_0, z_1, \mathbf{y}) \in \mathcal{Q}^r_{n+2}$

2.2.4 Minimizing a L^2-norm. Let us now consider the problem of minimizing the L^2-norm of $\mathbf{B}\mathbf{x}$ under some affine constraints:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \|\mathbf{B}\mathbf{x}\|_2 \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned} \tag{15}$$

As such this problem does not fit (2) since the objective function is non-linear. In order to circumvent this, one needs to consider the epigraph of $F(\mathbf{x}) = \|\mathbf{B}\mathbf{x}\|_2$ defined as $\operatorname{epi} F = \{(t, \mathbf{x}) \text{ s.t. } F(\mathbf{x}) \leq t\}$. Minimizing F is then equivalent to minimizing t under the constraint that $(t, \mathbf{x}) \in \operatorname{epi} F$. For the present case, we therefore have:

$$\begin{aligned} \min_{\mathbf{x},t} \quad & t \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \|\mathbf{B}\mathbf{x}\|_2 \leq t \end{aligned} \tag{16}$$

Introducing an additional variable $\mathbf{y}$ we have:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y},t} \quad & t \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{B}\mathbf{x} - \mathbf{y} = 0 \\ & \|\mathbf{y}\|_2 \leq t \end{aligned} \tag{17}$$

where the last constraint is again a quadratic Lorentz cone constraint. Problem (15) is now a linear problem of the augmented optimization variables $(\mathbf{x}, \mathbf{y}, t)$ under linear and conic constraints.

2.3 Conic representable sets and functions

As previously mentioned, minimizing a convex function $F(\mathbf{x})$ can be turned into a linear problem with a convex non-linear constraint involving the epigraph of F. We will thus consider the class of

conic representable functions as the class of convex functions which can be expressed as follows:

$$\begin{aligned} F(\mathbf{x}) = \min_{\mathbf{y}} \quad & \mathbf{c}_x^{\mathrm{T}}\mathbf{x} + \mathbf{c}_y^{\mathrm{T}}\mathbf{y} \\ \text{s.t.} \quad & \mathbf{b}_l \leq \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{y} \leq \mathbf{b}_u \\ & \mathbf{y} \in \mathcal{K} \end{aligned} \tag{18}$$

in which $\mathcal{K}$ is again a product of cones of the kinds detailed in section 2.1.

For instance, consider the case of the L^1-norm, we have:

$$\begin{aligned} \|\mathbf{x}\|_1 = \min_{\mathbf{y}\in\mathbb{R}^n} \quad & \mathbf{e}^{\mathrm{T}}\mathbf{y} \\ \text{s.t.} \quad & \mathbf{0} \leq \mathbf{x} + \mathbf{y} \\ & \mathbf{x} - \mathbf{y} \leq \mathbf{0} \end{aligned} \tag{19}$$

where $\mathbf{e} = (1, \dots, 1)$ whereas for the L^2-norm we have:

$$\begin{aligned} \|\mathbf{x}\|_2 = \min_{\mathbf{y}\in\mathbb{R}^{n+1}} \quad & y_0 \\ \text{s.t.} \quad & \mathbf{x} - \bar{\mathbf{y}} = \mathbf{0} \\ & \mathbf{y} \in \mathcal{Q}_{n+1} \end{aligned} \tag{20}$$

where $\mathbf{y} = (y_0, y_1, \dots, y_n)$ and $\bar{\mathbf{y}} = (y_1, \dots, y_n)$. In this example, it can be seen that the representation (18) is not necessarily optimal in terms of number of additional variables, one could perfectly eliminate the $\mathbf{y}$ variable. However, in most practical cases, functions like the L^2-norm will quite often be composed with some linear operator (gradient, interpolation, etc.) so that introducing such additional variables will be necessary to fit format (2).

Obviously, if F is the indicator function of a convex set, then we have a similar notion of *conic representable sets* for which only the constraints in (18) are relevant.

3 VARIATIONAL PROBLEMS AND THEIR DISCRETE VERSION

3.1 A first illustrative example

Before describing the framework of variational formulation and discretization, let us first introduce a classical example of variational inequality, namely the obstacle problem. Let Ω be a bounded domain of $\mathbb{R}^2$ and $u \in V = H_0^1(\Omega)$, $f \in H^{-1}(\Omega)$ and $g \in H^1(\Omega) \cap C^0$ such that $y \leq 0$ on $\partial\Omega$. The obstacle problem consists in solving:

$$\begin{aligned} \inf_{u\in V} \quad & \int_\Omega \frac{1}{2}\|\nabla u\|_2^2 \,\mathrm{d}x - \int_\Omega f u \,\mathrm{d}x \\ \text{s.t.} \quad & u \in \mathcal{K} \end{aligned} \tag{21}$$

where $\mathcal{K} = \{v \in H_0^1(\Omega) \text{ s.t. } v \geq g \text{ on } \Omega\}$. Physically, this problem corresponds to that of a membrane described by an out-of-plane deflection u and loaded by a vertical load f which may potentially enter in contact with a rigid obstacle located on the surface $z = -g(x, y)$.

3.2 Discretization

Let us now consider some finite element discretization of Ω using a mesh $\mathcal{T}_h$ of N_e triangular cells. For the displacement field u, we consider a Lagrange piecewise linear interpolation represented by the discrete functional space $V_h = \{v \in C^0(\Omega) \text{ s.t. } v|_T \in \mathbb{P}^1(T) \quad \forall T \in \mathcal{T}_h\}$ of dimension N. Interpolating the obstacle position y on the same space V_h, a discrete approximation $\mathcal{K}_h$ of $\mathcal{K}$ consists in a pointwise inequality on the vectors $\mathbf{v}, \mathbf{g} \in \mathbb{R}^N$ of degrees of freedom of $v_h, g_h \in V_h$: $\mathcal{K}_h = \{v_h \in V_h \text{ s.t. } \mathbf{v} \geq \mathbf{g}\}$. Finally, introducing a quadrature formula with M quadrature points for

the first integral in (21), the discrete obstacle problem is now:

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^N} \quad & \sum_{g=1}^{M} \omega_g \frac{1}{2} \|\mathbf{B}_g \mathbf{u}\|_2^2 - \mathbf{f}^{\mathrm{T}}\mathbf{u} \\ \text{s.t.} \quad & \mathbf{u} \geq \mathbf{g} \end{aligned} \tag{22}$$

In (22), $\mathbf{B}_g\mathbf{u} \in \mathbb{R}^2$ denotes the discrete gradient evaluated at the current quadrature point g and ω_g is the associated quadrature weight. Note that since u_h is linear, its gradient is piecewise-constant so that only one point per triangle T with $\omega_g = |T|$ is sufficient for exact evaluation of the integral ($M = N_e$ in this case). Finally, $\mathbf{f}$ is the assembled finite-element vector corresponding to the linear form $L(u) = \int_\Omega f u \,\mathrm{dx}$.

The quadratic term in the objective function is now rewritten following section 2.2 as follows:

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^N} \quad & \sum_{g=1}^{M} \omega_g y_{g,0} - \mathbf{f}^{\mathrm{T}}\mathbf{u} \\ \text{s.t.} \quad & \mathbf{u} \geq \mathbf{g} \\ & y_{g,1} = 1 \\ & \mathbf{B}_g\mathbf{u} - \begin{bmatrix} y_{g,2} \\ y_{g,3} \end{bmatrix} = 0 \qquad \forall g = 1, \dots, M \\ & (y_{g,0}, y_{g,1}, y_{g,2}, y_{g,3}) \in Q_4^r \end{aligned} \tag{23}$$

Collecting the $4M$ auxiliary variables $\mathbf{y}_g = (y_{g,0}, y_{g,1}, y_{g,2}, y_{g,3})$ into a global vector $\widehat{\mathbf{y}} = (\mathbf{y}_1, \dots \mathbf{y}_M) \in \mathbb{R}^{4M}$, the previous problem can be rewritten as:

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^N, \widehat{\mathbf{y}}\in\mathbb{R}^{4M}} \quad & \mathbf{c}^{\mathrm{T}}\widehat{\mathbf{y}} - \mathbf{f}^{\mathrm{T}}\mathbf{u} \\ \text{s.t.} \quad & \mathbf{u} \geq \mathbf{g} \\ & \mathbf{A}_u\mathbf{u} + \mathbf{A}_y\widehat{\mathbf{y}} = \mathbf{b} \\ & \widehat{\mathbf{y}} \in Q_4^r \times \cdots \times Q_4^r \end{aligned} \tag{24}$$

where $\mathbf{c} = (\omega_1, 0, 0, 0, \omega_2, 0 \dots, \omega_M, 0, 0, 0)$, $\mathbf{A}_u = \begin{bmatrix} 0 \\ \mathbf{B}_1 \\ 0 \\ \mathbf{B}_2 \\ \vdots \\ 0 \\ \mathbf{B}_M \end{bmatrix}$, $\mathbf{A}_y = \begin{bmatrix} -\mathbf{I} & & \\ & \ddots & \\ & & -\mathbf{I} \end{bmatrix}$ with $\mathbf{I} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $\mathbf{b} = (1, 0, 0, 1, 0 \dots, 1, 0, 0)$. This last formulation enables to see that problem (22) indeed fits into the general conic programming framework (2) but in a specific fashion since it possesses a block-wise structure induced by the quadrature rule. Indeed, each 4-dimensional block of auxiliary variables y_g is decoupled from each other and is linked to the main unknown variable $\mathbf{u}$ through the evaluation of the discrete gradient at each point g. The conic reformulation performed in (22) is in fact the same for all quadrature points.

This observation motivates us to rewrite the initial continuous problem as:

$$\begin{aligned} \inf_{u\in V} \quad & \int_\Omega F(\nabla u)\,\mathrm{dx} - \int_\Omega f u \,\mathrm{dx} \\ \text{s.t.} \quad & u \in \mathcal{K} \end{aligned} \tag{25}$$

with $F(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|_2^2$ which is conic representable as follows[2]:

$$\begin{aligned} F(\mathbf{x}) = \min_{\mathbf{y}\in\mathbb{R}^4} \quad & y_0 \\ \text{s.t.} \quad & y_1 = 1 \\ & \mathbf{x} - \begin{bmatrix} y_2 \\ y_3 \end{bmatrix} = \mathbf{0} \\ & \mathbf{y} \in \mathcal{Q}_4^r \end{aligned} \tag{26}$$

Introducing now the previously mentioned discretization and the quadrature formula, we aim at solving:

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^N} \quad & \sum_{g=1}^{M} \omega_g F(\mathbf{B}_g\mathbf{u}) - \mathbf{f}^{\mathrm{T}}\mathbf{u} \\ \text{s.t.} \quad & \mathbf{u} \geq \mathbf{g} \end{aligned} \tag{27}$$

which will be equivalent to (22) when injecting (26) into (27) since, for all M evaluations of F, a 4-dimensional auxiliary vector $\mathbf{y}_g$ will be introduced as an additional minimization variable.

As a consequence, the `fenics_optim` package has been particularly designed for a sub-class of problems of type (1) in which $J(u) = \int_\Omega j(u)\,\mathrm{d}x$ for which integration will be handled by the FEniCS machinery and in which the user must specify the local conic representation of j.

3.3 FEniCS formulation

In the following, we present the main part of a `fenics_optim` script. More details on how an optimization problem is defined are discussed in A. In particular, it is possible to write *manually* the discretized version of the obstacle problem based on (24) (see A.2). However, `fenics_optim` also provides a more user-friendly way of modelling such problems which is based on (25) and (26) and will now be presented.

First, a simple unit square mesh and $\mathbb{P}^1$ Lagrange function space `V` is defined using basic FEniCS commands. Homogeneous Dirichlet boundary conditions are also defined in variable `bc`. Finally, `obstacle` is the interpolant on V of

$$g(x,y) = g_0 + a\sin(2\pi k_1 x)\cos(2\pi k_1 y)\sin(2\pi k_2 x)\cos(2\pi k_2 y)$$

In the following simulations, we took $g_0 = -0.1$, $a = 0.01$, $k_1 = 2$ and $k_2 = 8$. The loading is also assumed to be uniform and given by $f = -5$. The main part of the script starts by instantiating a `MosekProblem` object and adding a first optimization variable `u` living in the function space `V`, subject to Dirichlet boundary conditions `bc`. The `add_var` method also enables to define a lower bound (resp. an upper bound) on an optimization variable by specifying a value for the `lx` (resp. `ux`) keyword. For the present case, we use `lx=obstacle` for enforcing $\mathbf{u} \geq \mathbf{g}$.

```
prob = MosekProblem("Obstacle problem")
u = prob.add_var(V, bc=bc, lx=obstacle)

prob.add_obj_func(-dot(load,u)*dx)
```

where we also added the linear part of the objective function through the `add_obj_func` method.

The next step consists in defining the quadratic part of the objective function. For this purpose, we define a class inheriting from the base `MeshConvexFunction` class which must be instantiated by specifying on which previously defined optimization variable[3] this function will act (here the only possible variable is `u`). Moreover, we also specify the degree of the quadrature necessary

[2]Note that it would have been possible to work directly with function $\widetilde{F}(u) := \frac{1}{2}\|\nabla u\|_2^2$ which is also conic-representable
[3]see the notion of block-variables discussed in A.

for integrating the function (one-point quadrature used by default but written explicitly in the code snippet below). We must also define the `conic_repr` method which will encode the conic representation (26). We add a local optimization variable `Y` of dimension 4 which will belong to the cone `RQuad(4)` representing Q_4^r. Equality constraints are then added using the `add_eq_constraint` by specifying, as in (18), a block matrix $\begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}$ and a right-hand side `b` (= 0 by default). Note that both equality constraints could also have been written in a single one of row dimension 3. Finally, the local linear objective $(\mathbf{c}_x, \mathbf{c}_y)$ vector is defined using the `set_linear_term` method.

```
class QuadraticTerm(MeshConvexFunction):
    def conic_repr(self, X):
        Y = self.add_var(4, cone=RQuad(4))
        self.add_eq_constraint([None, Y[1]], b=1)
        self.add_eq_constraint([X, -as_vector([Y[2], Y[3]])])
        self.set_linear_term([None, Y[0]])

F = QuadraticTerm(u, degree=0)
F.set_term(grad(u))
prob.add_convex_term(F)
```

Note that constraints and linear objectives are all defined in a block-wise manner, these blocks consisting of, first, the main variable which has been specified at instantiation (`u` in this case), then the additional local variables (`Y` here). Besides, these blocks are represented in terms of their action on the block variables using UFL expressions.

The `set_term` method enables to evaluate F for the gradient of u using the UFL `grad` operator. This function is then added to the global optimization problem. Finally, optimization (minimization by default) is performed by calling the `optimize` method of the `MosekProblem` object:

```
prob.optimize()
```

For validation and performance comparison, the obstacle problem has been solved for various mesh sizes using the `fenics_optim` toolbox as well as using PETSc's TAO quadratic bound-constrained solver [5, 39] which is particularly well suited for this kind of problems. We used the Trust Region Newton Method (TRON) and an ILU-preconditioned conjugate gradient solver for the inner iterations. Results in terms of optimal objective function value, total optimization time and number of iterations have been reported for both methods in table 1. Note that default convergence tolerances have been used in both cases and that total optimization time includes the presolve step of Mosek which can efficiently eliminate redundant linear constraints for instance. It can be observed that both approach yield close results in terms of optimal objective values and that TAO's solver is more efficient than Mosek in terms of optimization time as expected, mainly because of the small number of iterations needed to reach convergence but also because no additional variables are introduced when using TAO. However, Mosek surprisingly becomes quite competitive for large-scale problems because of its number of iterations scaling quite weakly with the problem size, contrary to the TRON algorithm. Membrane displacement along the line $y = 0.5$ and contact area for $h = 1/400$ have been represented in Figure 1.

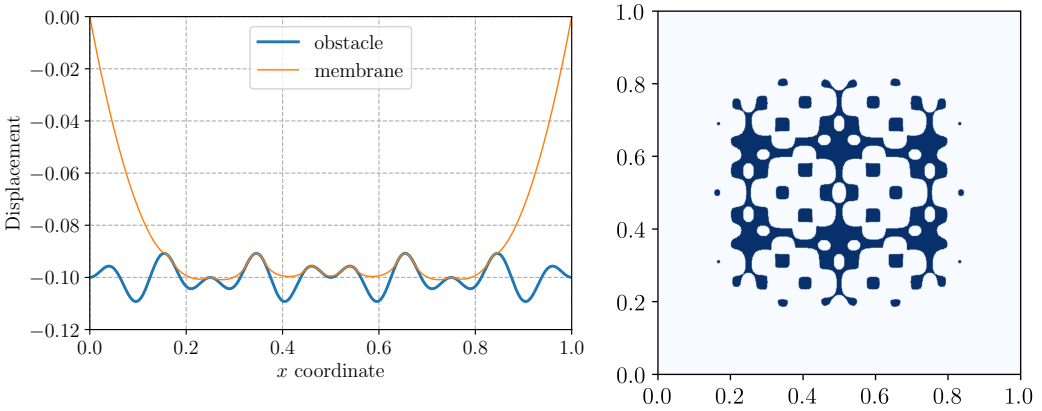
4 A MORE ADVANCED EXAMPLE

Let us now consider the following problem:

$$\begin{array}{ll}\inf\limits_{u\in V} & \int_\Omega \|\nabla u\|_2 \,\mathrm{dx} \\ \text{s.t.} & \int_\Omega f u \,\mathrm{dx} = 1\end{array} \tag{28}$$

Mesh size	Interior point (Mosek)			TRON algorithm (TAO)		
	Objective	Opt. time	iter.	Objective	Opt. time	iter.
$h = 1/25$	-0.265081	0.13 s	14	-0.265082	0.09 s	5
$h = 1/50$	-0.264932	0.56 s	15	-0.264932	0.22 s	6
$h = 1/100$	-0.264883	2.27 s	16	-0.264884	1.04 s	10
$h = 1/200$	-0.264867	10.04 s	19	-0.264871	6.03 s	14
$h = 1/400$	-0.264864	48.95 s	20	-0.264868	47.79 s	22

Table 1. Comparison between the `fenics_optim` implementation of the obstacle problem relying on the interior point Mosek solver and TAO's bound-constrained TRON solver.



(a) Membrane displacement in contact with the obstacle along $y = 0.5$

(b) Contact area in blue

Fig. 1. Results of the obstacle problem for $h = 1/400$

This problem is known to be related to antiplane limit analysis problems in mechanics as well as to the Cheeger problem and the eigenvalue fo the 1-Laplacian when $f = 1$ [16, 17, 20]. In this particular case, the solution of (28) can indeed be shown to be proportional to the characteristic function of a subset $C_\Omega \subseteq \Omega$ known as the Cheeger set of Ω which is the solution of:

$$C_\Omega := \underset{\omega \subseteq \Omega}{\operatorname{argmin}} \quad \frac{|\partial\omega|}{|\omega|} \tag{29}$$

that is the subset minimizing the ratio of perimeter over area, the associated optimal value of this ratio c_Ω being known as the *Cheeger constant*.

This problem is not strictly convex and is particularly difficult to solve using standard algorithms due to the highly non-smooth objective term. Again, introducing a $\mathbb{P}^k$ Lagrange discretization for u, we aim at solving the following discrete problem:

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^N} \quad & \sum_{g=1}^{M} \omega_g F(\mathbf{B}_g \mathbf{u}) \\ \text{s.t.} \quad & \mathbf{f}^T\mathbf{u} = 1 \end{aligned} \tag{30}$$

where $F(\mathbf{x}) = \|\mathbf{x}\|_2$ with its conic representation being given by (20). Similarly to the obstacle problem, choosing a $\mathbb{P}^1$ discretization requires only a one-Gauss point quadrature rule for the

objective function evaluation. For $\mathbb{P}^k$ with $k \geq 2$, the quadrature is always inexact and Gaussian quadrature is not necessarily optimal. For the particular case $k = 2$, one can choose a *vertex* quadrature scheme on the simplex triangle to ensure that the discrete integral is approximated by excess:

$$\int_T \|\mathbf{r}(x, y)\| \, \mathrm{dx} \lesssim \frac{|T|}{3} \sum_{i=1}^{3} \|\mathbf{r}(x_i, y_i)\| \tag{31}$$

where (x_i, y_i) denote the simplex vertices. The choice of the quadrature scheme can also be made when defining the corresponding `MeshConvexFunction`:

```
class L2Norm(MeshConvexFunction):
    """ Defines the L2-norm function ||x||_2 """
    def conic_repr(self, X):
        d = self.dim_x
        Y = self.add_var(d+1, cone=Quad(d+1))
        Ybar = as_vector([Y[i] for i in range(1, d+1)])
        self.add_eq_constraint([X, -Ybar])
        self.set_linear_term([None, Y[0]])

prob = MosekProblem("Cheeger problem")
u = prob.add_var(V, bc=bc)

if degree == 1:
    F = L2Norm(u)
elif degree == 2:
    F = L2Norm(u, "vertex")
else:
    F = L2Norm(u, degree = degree)
F.set_term(grad(u))
prob.add_convex_term(F)
```

In the previous code, `degree` denotes the polynomial degree k of function space `V`. If $k = 1$, the default one-point quadrature rule is used, if $k = 2$ the above-mentioned *vertex* scheme is used, otherwise a default Gaussian quadrature rule for polynomials of degree k is used. `Quad(d+1)` corresponds to the quadratic Lorentz cone Q_{d+1} of dimension $d + 1$ where $d =$ `self.dim_x` is the dimension of the `X` variable.

In the Cheeger problem, a normalization constraint must also be added. This can again be done by adding a convex term including only the corresponding constraint or it can also be added directly to the `MosekProblem` instance by defining the function space for the Lagrange multiplier corresponding to the constraint (here it is scalar so we use a `"Real"` function space) and passing the corresponding constraint in its weak form as follows:

```
f = Constant(1.)
R = FunctionSpace(mesh, "Real", 0)
def constraint(l):
    return [l*f*u*dx]
prob.add_eq_constraint(R, A=constraint, b=1)
```

4.1 Discontinuous Galerkin discretization

Problem (28) can be discretized using standard Lagrange finite elements but also using Discontinous Galerkin discretization, in this case the gradient L^2-norm objective term is completed by

absolute values of the jumps of u:

$$\begin{aligned} \min_{u\in V} \quad & \int_\Omega \|\nabla u\|_2 \,\mathrm{dx} + \int_\Gamma |[\![u]\!]| \,\mathrm{dS} + \int_{\Gamma_D} |u| \,\mathrm{dS} \\ \text{s.t.} \quad & \int_\Omega f u \,\mathrm{dx} = 1 \end{aligned} \tag{32}$$

where Γ denotes the set of internal edges, Γ_D the Dirichlet boundary part and $[\![u]\!] = u^+ - u^-$ is the jump across Γ.

The discretized version using discontinuous $\mathbb{P}_d^k$ Lagrange finite elements reads as:

$$\begin{aligned} \min_{\mathbf{u}\in\mathbb{R}^N} \quad & \sum_{g=1}^{M} \omega_g F(\mathbf{B}_g\mathbf{u}) + \sum_{g_e=1}^{M_e} \omega_{g_e} G(\mathbf{J}_{g_e}\mathbf{u}) + \sum_{g_d=1}^{M_d} \omega_{g_d} G(\mathbf{T}_{g_d}\mathbf{u}) \\ \text{s.t.} \quad & \mathbf{f}^{\mathrm{T}}\mathbf{u} = 1 \end{aligned} \tag{33}$$

where $G(x) = |x|$, g_e (resp. g_d) denotes a current quadrature point on the internal (resp. Dirichlet) facets, M_e (resp. M_d) denoting the total number of such points and ω_{g_e} (resp. ω_{g_d}) the associated quadrature weights. Finally, $\mathbf{J}_{g_e}\mathbf{u}$ denotes the evaluation of $[\![u]\!]$ at the quadrature point g_e and $\mathbf{T}_{g_d}\mathbf{u}$ the evaluation of u at g_d.

Similarly to the previously introduced `MeshConvexFunction`, we define a `FacetConvexFunction` corresponding to the conic representable convex function $G(x)$:

```
class AbsValue(FacetConvexFunction):
    def conic_repr(self, X):
        Y = self.add_var()
        self.add_ineq_constraint(A=[X, -Y], bu=0)
        self.add_ineq_constraint(A=[-X, -Y], bu=0)
        self.set_linear_term([None, Y])
```

When instantiating such a `FacetConvexFunction`, integration will (by default) be performed both on internal edges (in FEniCS the corresponding integration measure symbol is `dS`) and on external edges (FEniCS symbol being `ds`). If the Dirichlet boundary does not cover the entire boundary, then the `ds` measure can be restricted to the corresponding part. Again, the optimization variable on which acts the function must be specified and the desired quadrature rule can also be passed as an argument when instantiating the function. The `set_term` method can now take a list of UFL expression associated to the different integration measures. In the present case, G is evaluated for $[\![u]\!]$ on `dS` and for u on `ds`:

```
G = AbsValue(u)
G.set_term([jump(u), u])
prob.add_convex_term(G)
```

By default, facet integrals are evaluated using the *vertex* scheme.

4.2 Numerical example

We consider the problem of finding the Cheeger set of the unit square $\Omega = [0;1]^2$. The exact solution of this problem is known to be the unit square rounded by circles of radius $\rho = \dfrac{1}{2+\sqrt{\pi}}$ in its four corners, the associated Cheeger constant being $c_\Omega = 1/\rho$ [40, 44]. Results of the optimal field u for various discretization schemes have been represented on Figure 2. For all the retained discretization choices, the obtained Cheeger constant estimates are necessarily upper bounds to the exact one, in particular because of the choice of *vertex* quadrature schemes ensuring upper bound estimations such as (31). It can be seen on Figure 2 that all schemes yield a correct approximation

of the Cheeger set, except for the DG-0 scheme which is too stiff and produces straight edges in the corners, following the structured mesh edges.

4.3 A $H(\mathrm{div})$-conforming discretization for the dual problem

It can be easily shown through Fenchel-Rockafellar duality that problem (28) is equivalent to the following dual problem (see [17] for instance):

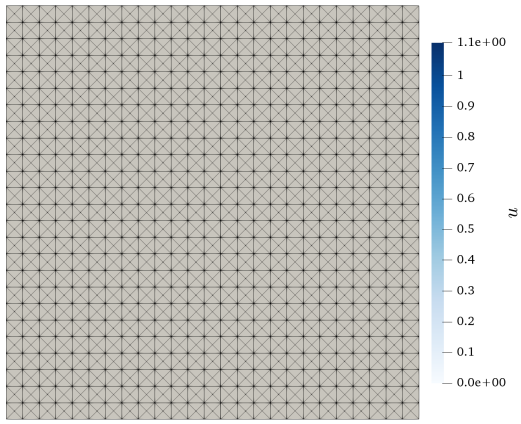
$$\begin{aligned} \sup_{\lambda\in\mathbb{R},\boldsymbol{\sigma}\in W} \quad & \lambda \\ \text{s.t.} \quad & \lambda f = \operatorname{div}\boldsymbol{\sigma} \quad \text{in } \Omega \\ & \|\boldsymbol{\sigma}\|_2 \leq 1 \end{aligned} \tag{34}$$

A natural discretization strategy for such a problem is to use $H(\mathrm{div})$-conforming elements such as the Raviart-Thomas element. Here, we will use the lowest Raviart-Thomas element, noted RT_1 by the FEniCS definition [32]. For the `fenics_optim` implementation, two minimization variables are defined: λ belonging to a scalar `"Real"` function space and $\boldsymbol{\sigma} \in RT_1$. Since for $\boldsymbol{\sigma} \in RT_1$, $\operatorname{div}\boldsymbol{\sigma} \in \mathbb{P}^0$, we write the constraint equation using $\mathbb{P}^0$ Lagrange multipliers:

```
N = 50
mesh = UnitSquareMesh(N, N, "crossed")

VRT = FunctionSpace(mesh, "RT", 1)
R = FunctionSpace(mesh, "Real", 0)
VDG0 = FunctionSpace(mesh, "DG", 0)

prob = MosekProblem("Cheeger dual")
lamb, sig = prob.add_var([R, VRT])

f = Constant(1.)
def constraint(u):
    return [lamb*f*u*dx, -u*div(sig)*dx]
prob.add_eq_constraint(VDG0, A=constraint, name="u")
```

Finally, since $\boldsymbol{\sigma} \in \mathbb{P}^1$ on a triangle, if the constraint $\|\boldsymbol{\sigma}\|_2$ is satisfied at the three vertices, it is satisfied everywhere by convexity. We here define a `MeshConvexFunction` representing the characteristic function of a L^2-ball constraint and select the `"vertex"` quadrature scheme so that the constraint will be indeed satisfied at the three vertices. Finally, the objective function is defined through the `add_obj_func` method of the problem instance:

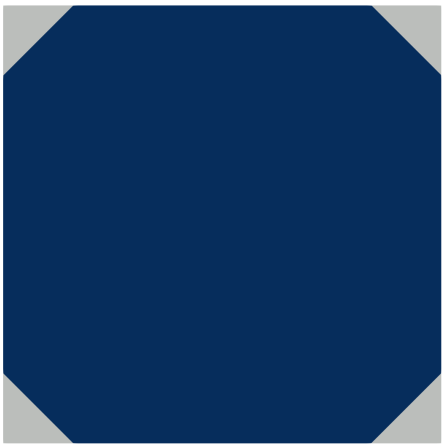
```
class L2Ball(MeshConvexFunction):
    """ Defines the L2-ball constraint ||x||_2 <= 1 """
    def conic_repr(self, X):
        d = self.dim_x
        Y = self.add_var(d+1, cone=Quad(d+1))
        Ybar = as_vector([Y[i] for i in range(1, d+1)])
        self.add_eq_constraint([X, -Ybar])
        self.add_eq_constraint([None, Y[0]], b=1)

F = L2Ball(sig, "vertex")
F.set_term(sig)
prob.add_convex_term(F)

prob.add_obj_func([1, None])
```



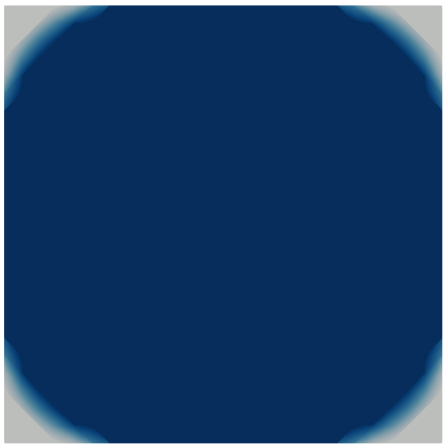
(a) 25×25 mesh

(b) CG-1

(c) CG-2



(d) DG-0



(e) DG-1

Fig. 2. Results of the Cheeger problem for various discretizations on the unit square: continuous Galerkin (CG) and discontinuous Galerkin (DG) of degrees $k = 0, 1$ or 2.

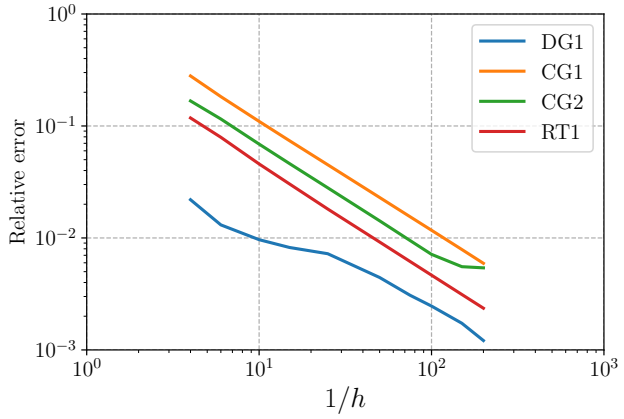


Fig. 3. Convergence results on the Cheeger problem

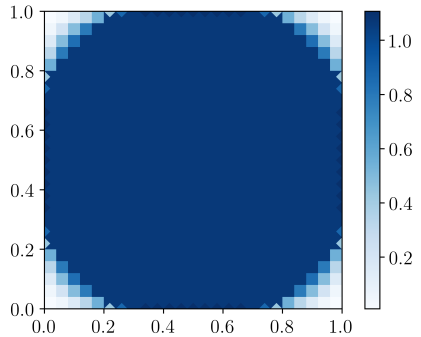


Fig. 4. Optimal u field from the RT discretization

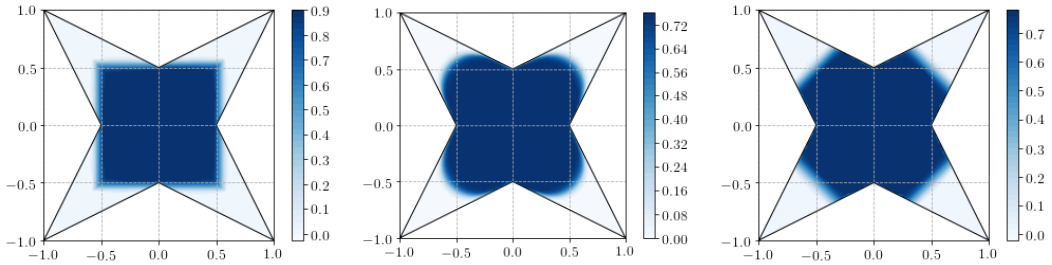
With the above-mentioned discretization and quadrature choice, it can easily be shown that the discrete version of (34) will produce a lower bound of the exact Cheeger constant. For instance, for a 25×25 mesh, we obtained:

$$c_\Omega^{RT_1} \approx 3.704 \leq c_\Omega \approx 3.772 \leq c_\Omega^{DG1} \approx 3.800 \tag{35}$$

Convergence results of the numerical Cheeger constant estimate $c_{\Omega,h}$ obtained with the previous CG/DG discretizations as well as with the present RT discretization have been reported in Figure 3. The relative error is computed as $\epsilon(c_{\Omega,h}/c_\Omega - 1)$ where $\epsilon = -1$ for the RT discretization and $\epsilon = 1$ otherwise. We observe in particular that the DG1 scheme is the most accurate and that all schemes have the same convergence rate in $O(h)$. Finally, primal-dual solvers such as Mosek also provide access to the optimal values of constraint Lagrange multipliers. The Lagrange multiplier associated with the constraint $\lambda f = \operatorname{div} \boldsymbol{\sigma}$ can be interpreted as the field u from the primal problem. This Lagrange multiplier, which belongs to a DG0 space, has been represented in Figure 4.

4.4 A library of convex representable functions

In the `fenics_optim` library, instead of defining each time the conic representation of usual functions, a library of common convex functions has been already implemented, including:



(a) L^1-norm (b) L^2-norm (c) L^∞-norm

Fig. 5. Generalized Cheeger sets of a star-shaped domain using different norms

- linear functions $F(\mathbf{x}) = \mathbf{c}^\mathrm{T}\mathbf{x}$
- quadratic functions $F(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^\mathrm{T}\mathbf{Q}^\mathrm{T}\mathbf{Q}(\mathbf{x}-\mathbf{x}_0)$
- absolute value $F(x) = |x|$
- L^1, L^2 and L^∞ norms
- L^1, L^2 and L^∞ balls characteristic functions

These functions inherit from the composite class `ConvexFunction` which, by default, behaves like a `MeshConvexFunction`. To use them as `FacetConvexFunction`, they can be instantiated as `F = L2Norm.on_facet(u)`. Using such predefined functions, many problems can be formulated in an extremely simple manner, without even worrying about the conic reformulation. For instance, we revisited the Cheeger problem on a star-shaped domain but with anisotropic norms [29] such as L^1 and L^∞ instead of L^2 in (28)[4], the resulting sets are represented on Figure 5.

5 A GALLERY OF ILLUSTRATIVE EXAMPLES

We now give a series of examples which illustrate the versatility of the `fenics_optim` package for formulating and solving problems taken from the fields of solid and fluid mechanics, image processing and applied mathematics. The last two examples involve, in particular, time-dependent problems. Let us again point out that discretization choices or solver strategies using interior-point methods are not necessarily the most optimal ones for each of these problems and that many other approaches which have been proposed in the literature may be much more appropriate. We just aim at illustrating the potential of the package to formulate and solve various problems.

5.1 Limit analysis of thin plates in bending

The first problem consists in finding the ultimate load factor that a thin plate in bending can sustain given a predefined strength criterion and boundary condition. This limit analysis problem has been studied in [12, 22]. In the present case, we consider a unit square plate made of a von Mises material of uniform bending strength m and subjected to a uniformly distributed loading f. The thin plate limit analysis problem consists in solving the following problem:

$$\begin{aligned} \inf_{u\in \mathrm{HB}_0(\Omega)} \quad & \int_\Omega \pi(\nabla^2 u)\,\mathrm{d}x \\ \text{s.t.} \quad & \int_\Omega f u\,\mathrm{d}x = 1 \end{aligned} \tag{36}$$

[4]Note that in the general case of an L^p-norm for the gradient term, the corresponding jump term in (32) is $\int_\Gamma |[\![u]\!]| \cdot \|\boldsymbol{n}\|_p \,\mathrm{d}S$ where $\boldsymbol{n}$ is the facet normal and similarly for the Dirichlet boundary term.

where HB_0 is the space of bounded Hessian functions [23] with zero trace on $\partial\Omega$ and $\pi(M) = \frac{2m}{\sqrt{3}}\sqrt{M_{11}^2 + M_{22}^2 + M_{12}^2 + M_{11}M_{22}}$ for any $M \in \mathbb{S}_2^+$. One can notice that problem (36) shares some similar structure with the Cheeger problem (28) except that we are now dealing with the Hessian operator and a different norm through function π.

Contrary to elastic bending plate problems involving functions with C^1-continuity, we deal here with functions in HB which are continuous but may have discontinuities in their normal gradient $\partial_n u$, in particular we can consider again a Lagrange interpolation for u with jumps of $\partial_n u$ across all internal facets $F \in \Gamma_h$ of unit normal $\boldsymbol{n}$. The π-function being some generalized total variation for $\nabla^2 u$, we have explicitly [13]:

$$\begin{aligned} \inf_{u \in \mathrm{HB}_0(\Omega)} \quad & \sum_{T \in \mathcal{T}_h} \int_T \pi(\nabla^2 u)\,\mathrm{dx} + \sum_{F \in \Gamma_h} \int_F \pi([\![\partial_n u]\!]\boldsymbol{n} \otimes \boldsymbol{n})\,\mathrm{dS} \\ \text{s.t.} \quad & \int_\Omega f u\,\mathrm{dx} = 1 \end{aligned} \tag{37}$$

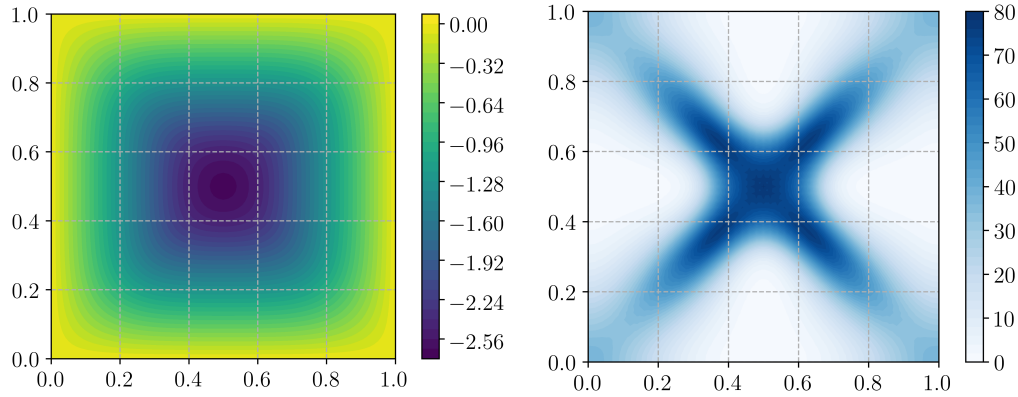
where it happens that in fact $\pi([\![\partial_n u]\!]\boldsymbol{n} \otimes \boldsymbol{n}) = |[\![\partial_n u]\!]|\pi(\boldsymbol{n} \otimes \boldsymbol{n}) = |[\![\partial_n u]\!]|\frac{2m}{\sqrt{3}}$. Following B, we have the following formulation of the bending plate problem for a $\mathbb{P}^2$ interpolation:

```
prob = MosekProblem("Bending plate limit analysis")

V = FunctionSpace(mesh, "CG", 2)
bc = DirichletBC(V, Constant(0.), boundary)
u = prob.add_var(V, bc = bc)

R = FunctionSpace(mesh, "R", 0)
def Pext(lamb):
    return [lamb*dot(load,u)*dx]
prob.add_eq_constraint(R, A=Pext, b=1)

J = as_matrix([[2., 1., 0.],
               [0, sqrt(3.), 0.],
               [0, 0, 1]])
def Chi(v):
    chi = sym(grad(grad(v)))
    return as_vector([chi[0,0], chi[1,1], 2*chi[0, 1]])
pi_c = L2Norm(u, "vertex", degree=1)
pi_c.set_term(m/sqrt(3)*dot(J, Chi(u)))
prob.add_convex_term(pi_c)

pi_h = L1Norm.on_facet(u)
pi_h.set_term([jump(grad(u), n)], k=2/sqrt(3)*m)
prob.add_convex_term(pi_h)

prob.optimize()
```

The reference solution for this problem is known to be $25.02m/f$ [15], whereas we find $25.05m/f$ for a 50×50 structured mesh. The corresponding solutions for u and $\pi(\nabla^2 u)$ are represented in Figure 6.

5.2 Viscoplastic yield stress fluids

Viscoplastic (or yield stress) fluids [6, 21] are a particular class of non-Newtonian fluids which, in their most simple form, namely the Bingham model, behave like a purely rigid solid when the



(a) Optimal collapse mechanism u

(b) Curvature dissipation density $\pi(\nabla^2 u)$

Fig. 6. Results for the simply supported von Mises square plate

shear stress is below a critical yield stress τ_0 and flow like a Newtonian fluid when the shear stress is above τ_0. They appear in many applications ranging from civil engineering, petroleum, cosmetics or food industries. The solution of a steady state viscoplastic fluid flow under Dirichlet boundary conditions and a given external force field $\boldsymbol{f}$ can be obtained as the unique solution to the following convex variational principle [26]:

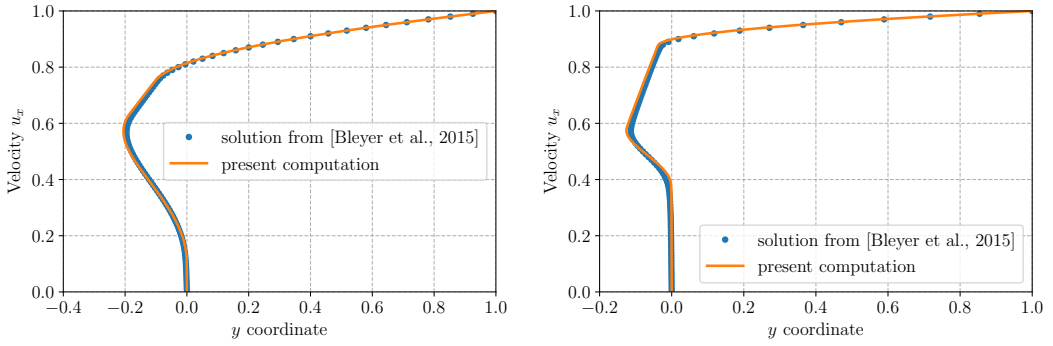
$$\begin{aligned} \inf_{\boldsymbol{u}\in\boldsymbol{H}^1(\Omega;\mathbb{R}^d)} \quad & \int_\Omega (\mu\|\nabla\boldsymbol{u}\|_2^2 + \sqrt{2}\tau_0\|\nabla\boldsymbol{u}\|_2)\,\mathrm{dx} - \int_\Omega \boldsymbol{f}\cdot\boldsymbol{u}\,\mathrm{dx} \\ \text{s.t.} \quad & \operatorname{div}\boldsymbol{u} = 0 \text{ in } \Omega \\ & \boldsymbol{u} = \boldsymbol{g} \text{ on } \partial\Omega \end{aligned} \tag{38}$$

where μ is the fluid viscosity. Typical solutions of problem (38) involve rigid zones in which $\nabla\boldsymbol{u} = 0$ and flowing regions where $\|\nabla\boldsymbol{u}\| \neq 0$, the locations of which are *a priori* unknown. Note that when $\tau_0 = 0$, we recover the classical viscous energy of Stokes flows and optimality conditions of problem (38) reduce to a linear problem. The FE discretization is quite classical, we adopt Taylor-Hood $\mathbb{P}^2/\mathbb{P}^1$ discretization for the velocity $\boldsymbol{u}$ and the pressure p which is the Lagrange multiplier of constraint $\operatorname{div}\boldsymbol{u} = 0$.

The considered problem is the classical lid-driven unit-square cavity, with $\boldsymbol{f} = 0$, $\boldsymbol{u} = 0$ everywhere on $\partial\Omega$, except on the top boundary $y = 1$ where $\boldsymbol{u} = (U, 0)$ with U the imposed constant velocity. Different solutions to problem (38) are then obtained depending on the value of the non-dimensional Bingham number $\mathtt{Bi} = \dfrac{\mu U}{\tau_0 L}$ with the characteristic length $L = 1$ for the present case. When $\mathtt{Bi} = 0$, the solution is that of a Newtonian fluid and when $\mathtt{Bi} \to \infty$ it corresponds to that of a purely plastic material.

Implementation in `fenics_optim` is straightforward once the symmetric tensor $\nabla\boldsymbol{u}$ has been represented as a vector of $\mathbb{R}^3$ through the `strain` function [11].

```
prob = MosekProblem("Viscoplastic fluid")

V = VectorFunctionSpace(mesh, "CG", 2)
bc = [DirichletBC(V, Constant((1.,0.)), top),
      DirichletBC(V, Constant((0.,0.)), sides)]
```



(a) Bi = 2 (b) Bi = 20

Fig. 7. Horizontal velocity profile $u_x(y)$ on the middle plane $x = 0.5$, comparison with results from [14]

```
u = prob.add_var(V, bc=bc)

Vp = FunctionSpace(mesh, "CG", 1)
def mass_conserv(p):
    return [p*div(u)*dx]
prob.add_eq_constraint(Vp, mass_conserv)

def strain(v):
    E = sym(grad(v))
    return as_vector([E[0, 0], E[1, 1], sqrt(2)*E[0, 1]])
visc = QuadraticTerm(u, degree=2)
visc.set_term(strain(u))
plast = L2Norm(u, degree=2)
plast.set_term(strain(u))

prob.add_convex_term(2*mu*visc)
prob.add_convex_term(sqrt(2)*tau0*plast)

prob.optimize()
```

The obtained optimal velocity field is compared on Figure 7 with that from a previous independent implementation described in [14]. Finally, if $\boldsymbol{d} = \nabla \boldsymbol{u} \neq 0$, then the stress inside the fluid is given by $\boldsymbol{\tau} = 2\mu\boldsymbol{d} + \sqrt{2}\tau_0 \dfrac{\boldsymbol{d}}{\|\boldsymbol{d}\|_2}$ and $\|\boldsymbol{\tau}\|_2 > \sqrt{2}\tau_0$. In Figure 8, $\|\boldsymbol{\tau}\|_2$ has been plotted with a colormap ranging from $\sqrt{2}\tau_0$ to $1.01\sqrt{2}\tau_0$, thus exhibiting the transition from solid regions (white) to liquid regions (blue).

5.3 Total Variation inpainting

In this example, we consider an image processing problem called *inpainting*, consisting in recovering an image which has been deteriorated. In the present case, we consider a color RGB image in which a fraction η of randomly chosen pixels have been lost (black). The inpainting problem consists in recovering the three color channels $\mathbf{U} = (\mathbf{u}_j)$ for $j \in \{R, G, B\}$ such that it matches the original color for pixels which have not been corrupted and minimizing a given energy for the remaining pixels. An efficient choice of energy for the inpainting problem is the L^2

(a) `Bi = 2` (b) `Bi = 20`

Fig. 8. Transition between solid (white) and liquid (regions). The bottom solid region is arrested and the central region rotates like a rigid body.

total variation norm $TV(u) = \int_\Omega \|\nabla u\|_2 \,\mathrm{dx}$ for a given color channel u. For an image, the discrete gradient can be computed by finite differences. Here, as we work with a FE library, the image will be represented using a Crouzeix-Raviart (CR) interpolation [18] on a structured finite element mesh. The inpainting problem therefore reads as:

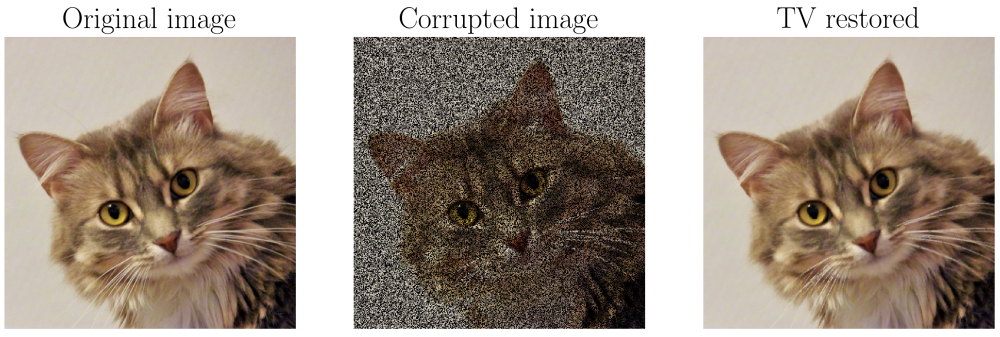
$$\begin{aligned} \min_{\mathbf{U}\in(\mathbb{R}^N)^3} \quad & \sum_{j\in\{R,G,B\}} \int_\Omega \|\nabla u_j\|_2 \,\mathrm{dx} \\ \text{s.t.} \quad & U_{i,j} = U^{\text{orig}}_{i,j} \quad \forall i \notin I_c, \ \ \forall j \in \{R,G,B\} \end{aligned} \tag{39}$$

where I_c denotes the set of corrupted pixels. Again, problem (39) can be defined very easily as follows:

```
prob = MosekProblem("TV inpainting")
u = prob.add_var(V, ux=ux, lx=lx)

for i in range(3):
    tv_norm = L2Norm(u)
    tv_norm.set_term(grad(u[i]))
    prob.add_convex_term(tv_norm)

prob.optimize()
```

where `V` is the space $(CR)^3$ and `ux` (resp. `lx`) denote functions of `V` equal to the original image on cells corresponding to uncorrupted pixels and which take $+\infty$ (resp. $-\infty$) values on I_c, so that `lx` $\leq$ `u` $\leq$ `ux` amounts to enforcing fidelity with the uncorrupted values. Finally, an L^2-norm term on the gradient of each channel is added to the problem. Results for a 512×512 image discretized using a triangular mesh of identical resolution (each pixel is split into two triangles) are represented in Figure 9 for two corruption levels. It must be noted that optimization took roughly one minute for both cases.

(a) $\eta = 25\%$ corruption level



(b) $\eta = 50\%$ corruption level

Fig. 9. Inpainting problem of a corrupted image using TV restoration

5.4 Cartoon+Texture Variational Image Decomposition

The next image processing example we consider is that of decomposing an image $y = u + v$ into a cartoon-like component u and a texture component v (here we assume that the image is not noisy). The cartoon layer u captures flat regions separated by sharp edges, whereas the texture component v contains the high frequency oscillations. There are many existing models to perform such a decomposition, in the following, we implement the model proposed by Y. Meyer [37, 47]:

$$\begin{aligned} &\inf_{u,v} \quad \int_\Omega \|\nabla u\|_2 \, \mathrm{dx} + \alpha \|v\|_G \\ &\text{s.t.} \quad y = u + v \end{aligned} \tag{40}$$

$$\text{where} \quad \|v\|_G = \inf_{\boldsymbol{g} \in L^\infty(\Omega;\mathbb{R}^2)} \{ \| \sqrt{g_1^2 + g_2^2} \|_\infty \text{ s.t. } v = \operatorname{div} \boldsymbol{g} \}$$

Fig. 10. Cartoon-texture decomposition with α = 2e-4

This model favors flat regions in u due to the use of the TV norm and oscillatory regions in v since $\|v\|_G$ increases for characteristic functions. Following [47], we reformulate the model as:

$$\begin{aligned} \inf_{u,\boldsymbol{g}} \quad & \int_\Omega \|\nabla u\|_2 \,\mathrm{dx} \\ \text{s.t.} \quad & y = u + \operatorname{div}(\boldsymbol{g}) \\ & \sqrt{g_1^2 + g_2^2} \leq \alpha \end{aligned} \tag{41}$$

The original image (512×512) is here represented on a triangular finite-element mesh of similar mesh size and we adopt a Crouzeix-Raviart interpolation for u and a Raviart-Thomas interpolation for $\boldsymbol{g}$. The constraint $y = u + \operatorname{div}(\boldsymbol{g})$ is enforced weakly on the CR space. The implementation reads as:

```
prob = MosekProblem("Cartoon/texture decomposition")
Vu = FunctionSpace(mesh, "CR", 1)
Vg = FunctionSpace(mesh, "RT", 1)
u, g = prob.add_var([Vu, Vg])

def constraint(l):
    return [dot(l, u)*dx, dot(l, div(g))*dx]
def rhs(l):
    return dot(l, y)*dx
prob.add_eq_constraint(Vu, A=constraint, b=rhs)

tv_norm = L2Norm(u)
tv_norm.set_term(grad(u))
prob.add_convex_term(tv_norm)

g_norm = L2Ball(g)
g_norm.set_term(g, k=alpha)
prob.add_convex_term(g_norm)

prob.optimize()
```

Results for the Barbara image decomposition are shown in Figure 10.

5.5 Time-dependent sandpile growth

In this example, we consider the time-dependent evolution model of a sandpile characterized by its height h. Since sand can fall off the table domain Ω, Dirichlet boundary conditions are prescribed. Layers of sand having a slope larger than the critical angle at rest $\tan\alpha$ will fall down the slope and can be modelled by Prighozin evolutionary PDE [43]:

$$\begin{aligned} \partial_t h - \operatorname{div}(m\nabla h) &= f \quad \text{in } \Omega \times [0;T] \\ m \geq 0,\ \|\nabla h\| &\leq \tan\alpha \\ m(\|\nabla h\| - \tan\alpha) &= 0 \end{aligned} \tag{42}$$

with $h(\boldsymbol{x}, t) = 0$ for $\boldsymbol{x} \in \partial\Omega$ and $h(\boldsymbol{x}, 0) = h_0(\boldsymbol{x})$ as the initial sandpile height. In the above, $-m\nabla h$ denotes the horizontal material flux of collapsing sand layers and f is a potential source term. This model has been also linked with the Monge-Kantorovitch problem of optimal mass transportation. Performing a backward implicit Euler discretization of the time derivative at each time step t_n and knowing the previous height configuration $h_{n-1}(\boldsymbol{x})$ at time $t = t_{n-1}$, finding $h_n(\boldsymbol{x})$ amounts to solving the following variational problem [24]:

$$\begin{aligned} \inf_{h} \quad & \frac{1}{2}\int_\Omega (h - g_n)^2 \,\mathrm{dx} \\ \text{s.t.} \quad & \|\nabla h\| \leq \tan\alpha \end{aligned} \tag{43}$$

where $g_n = \Delta t f + h_{n-1}$ and $\Delta t = T/N$ is the time interval of each N time increments discretization of interval $[0;T]$. Adopting a standard Lagrange $\mathbb{P}^1$ interpolation for h, problem (43) is solved N times with values of g_n updated from the previous solution. Figure 11 illustrate the results obtained with $\alpha = 30°$, no source term $f = 0$ and an initial unstable configuration for h_0 since $\|\nabla h_0\| > \tan\alpha$.

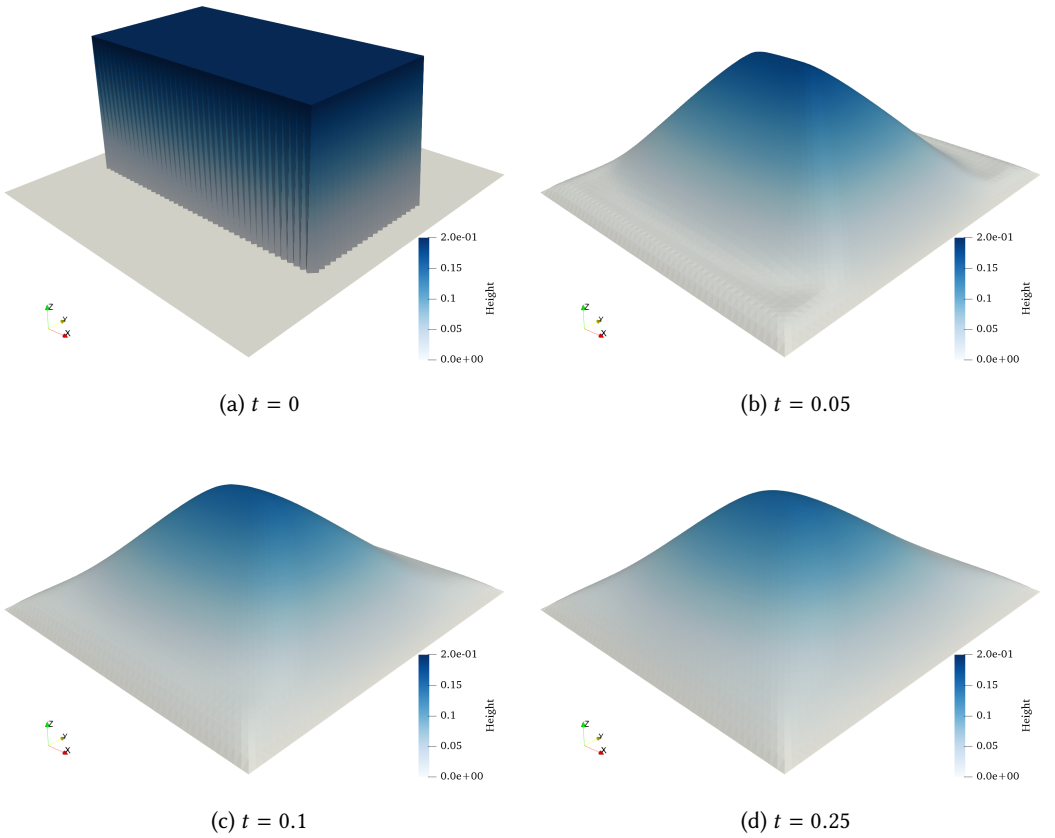
5.6 Optimal transport with space-time finite elements

Finally, we consider the Brenier-Benamou dynamic formulation [9] of quadratic cost optimal transport between two distributions ρ_0 and ρ_1 which reads as:

$$\begin{aligned} \inf_{\rho,\boldsymbol{v}} \quad & \frac{1}{2}\int_0^1\int_\Omega \rho(\boldsymbol{x},t)\|\boldsymbol{v}(\boldsymbol{x},t)\|_2^2 \,\mathrm{dx}\,\mathrm{dt} \\ \text{s.t.} \quad & \partial_t\rho + \operatorname{div}_x(\rho\boldsymbol{v}) = 0 \\ & \rho(\boldsymbol{x}, t=0) = \rho_0(\boldsymbol{x}) \\ & \rho(\boldsymbol{x}, t=1) = \rho_1(\boldsymbol{x}) \\ & \boldsymbol{v}\cdot\boldsymbol{n} = 0 \text{ on } \partial\Omega \end{aligned} \tag{44}$$

The change of variable $(\rho, \boldsymbol{m}) := (\rho, \rho\boldsymbol{v})$ proposed in [9] enables to obtain the following convex optimization problem:

$$\begin{aligned} \inf_{\rho,\boldsymbol{m}} \quad & \int_0^1\int_\Omega c(\rho,\boldsymbol{m}) \,\mathrm{dx}\,\mathrm{dt} \\ \text{s.t.} \quad & \partial_t\rho + \operatorname{div}_x \boldsymbol{m} = 0 \\ & \rho(\boldsymbol{x}, t=0) = \rho_0(\boldsymbol{x}) \\ & \rho(\boldsymbol{x}, t=1) = \rho_1(\boldsymbol{x}) \\ & \boldsymbol{m}\cdot\boldsymbol{n} = 0 \text{ on } \partial\Omega \end{aligned} \tag{45}$$



(a) $t = 0$ (b) $t = 0.05$

(c) $t = 0.1$ (d) $t = 0.25$

Fig. 11. Sandpile growth evolution starting from an initial unstable configuration (height amplification by factor 2)

where the cost function is $c(\rho, \boldsymbol{m}) = \begin{cases} \dfrac{\|\boldsymbol{m}\|_2^2}{2\rho} & \text{if } \rho > 0 \\ 0 & \text{if } (\rho, \boldsymbol{m}) = (0, 0) \\ +\infty & \text{otherwise} \end{cases}$. This function is convex and, observing that $c(\rho, \boldsymbol{m}) \leq t$ is equivalent to $2\rho t \geq \|\boldsymbol{m}\|_2^2$, is conic representable as follows:

$$\begin{aligned} c(\rho, \boldsymbol{m}) = \min_{\mathrm{y}} \quad & y_0 \\ \text{s.t.} \quad & \begin{bmatrix} \rho \\ m_1 \\ m_2 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = 0 \\ & \mathrm{y} \in \boldsymbol{Q}_4^r \end{aligned} \tag{46}$$

The numerical approximation is performed by relying on a space-time finite element discretization of $Q = [0; 1]^2 \times [0; T]$ with $T = 1$. We adopt $\mathbb{P}^2$ Lagrange finite elements for the 3d-vector $(\rho, \boldsymbol{m})$. Initial and boundary conditions are imposed on the different boundaries of the space-time cube. The mass conservation equation is replaced by a relaxed inequality between $\pm\epsilon$ to allow for small deviations because of errors induced by the space-time discretization. It is written as:

```
prob = MosekProblem("Optimal transport")
u = prob.add_var(V, bc=bc)

conserv = InequalityConstraint(u, degree=2)
rho, mx, my = u[0], u[1], u[2]
eps = 1e-6
conserv.set_term(rho.dx(2)+mx.dx(0)+my.dx(1), bl=-eps, bu=eps)
prob.add_convex_term(conserv)
```

where `dx(0)` and `dx(1)` stand for derivation along both spatial directions and `dx(2)` stands for derivation along the third time direction. Finally, the cost function term is added following reformulation (46):

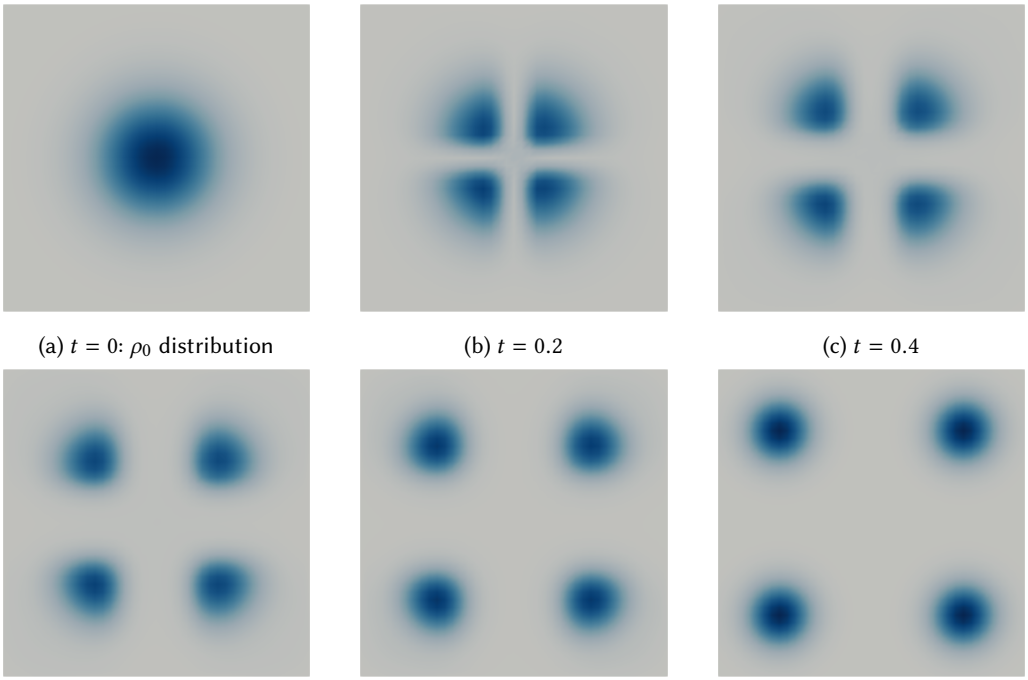
```
class CostFunction(ConvexFunction):
    def conic_repr(self, X):
        Y = self.add_var(dim=4, cone=RQuad(4))
        Ybar = as_vector([Y[i] for i in range(1, 4)])
        self.add_eq_constraint([X, -Ybar])
        self.set_linear_term([None, Y[0]])

c = CostFunction(u, degree=2)
c.set_term(u)
prob.add_convex_term(c)
```

Numerical results for $\rho(\boldsymbol{x}, t)$ at different time slices t are represented in Figure 12 for ρ_0 being a Gaussian distribution of standard deviation 0.2 and ρ_1 being four identical Gaussian distributions of standard deviation 0.1 located on four opposite points of $[0;1]^2$. It can be observed how the optimal transport splits the initial distribution ρ_0 into four parts driving towards ρ_1.

6 CONCLUSIONS AND PERSPECTIVES

With the Python package `fenics_optim` based on the FEniCS project, we propose a way to easily formulate convex variational problems arising in many applications of applied mathematics, image processing or mechanics. Convex optimization problems are formulated to fit into the conic programming framework in order to use efficient interior-point solvers especially tailored for such classes of problem. In the current form of the project, we use Mosek as the interior-point solver but other solvers could well be interfaced with the obtained discrete problems. The key point for fitting into the conic programming framework relies on a conic reformulation of convex functions. We have shown that many elementary convex functions such as L^p norms arising in applications can be indeed reformulated in such a way. In the gallery of examples we tackled, we showed that various problems can be formulated with the `fenics_optim` library in a very condensed manner. Besides, despite the fact that interior-point solvers are not necessarily the method of choice for all the considered examples, in particular for image processing applications, they are still very efficient and robust. They are therefore a good choice for a general-purpose solver for the present package. Finally, the versatility of FEniCS in terms of discretization solutions allowed to formulate very easily different discretization strategies, in particular including DG finite-elements or $H(\mathrm{div})$-conforming elements which naturally arise in dual problems.

Obviously, there exist many aspects for improving the scope of the library or its efficiency. In particular, the current implementation of `fenics_optim` does not allow for problems involving SDP matrix variables (Semi-Definite Programming problems), there are important applications in 3D computational mechanics which would benefit from such a feature. Besides, since version 9



(a) $t = 0$: ρ_0 distribution (b) $t = 0.2$ (c) $t = 0.4$

(d) $t = 0.6$ (e) $t = 0.8$ (f) $t = 1$: ρ_1 distribution

Fig. 12. Optimal transport between two distributions

of Mosek, power and exponential cones [19] are also available, which would broaden even more the class of conic representable functions including, for instance, L^p-norms with $p \notin \{1, 2, \infty\}$, exponential, entropy functions, etc[5]. Including such a feature would therefore be a huge added value for many applications.

As regards computational efficiency, we mentioned that interior-point solvers, although being efficient and robust, have important memory requirements for large-scale problems since they rely on solving Newton-like systems using direct solvers. Image processing applications usually rely on proximal algorithms for solving the corresponding optimization problems, it would therefore be interesting to implement such algorithms in the package. Finally, there are some internal limitations due to the current status of the FEniCS library (e.g. Lagrange multipliers cannot be defined on one sub-part, the boundary for instance, of the domain) that could be improved. Fortunately, the FEniCS project is currently experiencing a major redevelopment to bring new functionality and improve efficiency[6]. We will therefore aim at taking advantage of these new developments in the later versions of the `fenics_optim` package.

ACKNOWLEDGMENTS

The author would like to thank Gabriel Peyré for his useful advices when preparing the manuscript and F. Bleyer for providing some of the illustrative examples input material.

[5] see https://docs.mosek.com/modeling-cookbook/index.html

[6] https://github.com/FEniCS/dolfinx and https://github.com/FEniCS/ffcx/

A GENERAL PRINCIPLES OF FENICS_OPTIM INTERNAL STRUCTURE

A.1 Block-structure of the problem

The formulation of an optimization problem using `fenics_optim` relies on a block-structure definition of variables and constraints. Let us consider, for instance formulation (24). The conic reformulation leads to the introduction of variables $\widehat{\mathbf{y}}$ in addition to $\mathbf{u}$. Problem (24) therefore contains a block-structure of $p = 2$ variables $(\mathbf{x}^1, \mathbf{x}^2) = (\mathbf{u}, \widehat{\mathbf{y}}) \in \mathbb{R}^N \times \mathbb{R}^{4M}$. The internal machinery of `fenics_optim` works by adding sequentially new optimization variables, possibly associated with bound or conic constraints, and new linear equality or inequality constraints. Pseudo-code for defining such a block-wise structure would look like:

```
# Problem initialization
prob = MosekProblem("My problem")

# Adding a first block variable x1
x1 = prob.add_var(V1, lx=lx1, ux=ux1, cone=K1)
# Adding a first linear constraint
prob.add_ineq_constraint(W1, A=[a1], bl=bl1, bu=bu1)
```

At this stage, the `prob` instance represents the following problem:

$$\begin{aligned} \min_{\mathbf{x}^1 \in V_1} \quad & 0 \\ \text{s.t.} \quad & \mathbf{l}_x^1 \leq \mathbf{x}^1 \leq \mathbf{u}_x^1 \\ & \mathbf{b}_l^1 \leq \mathbf{A}^1 \mathbf{x}^1 \leq \mathbf{b}_u^1 \end{aligned} \tag{47}$$

where V_1 would be $\mathbb{R}^{d_1}$ in a purely discrete setting but will, in fact, be the variable `FunctionSpace` in the FEniCS FE-discretization setting. Bounds like $\mathbf{l}_x^1, \mathbf{u}_x^1, \mathbf{b}_l^1, \mathbf{b}_u^1$ are $\pm\infty$ by default (`None` in Python) and can be ignored in such case. $\mathcal{K}^1$ is a `Cone` object describing the type of cone to which the variable belongs (again `None` by default if there is no conic constraint). Finally, the linear constraint matrix $\mathbf{A}^1$ is represented by a bilinear form $a^1(y^1, x^1)$ on $W_1 \times V_1$ where y^1 is the constraint Lagrange multiplier and W_1 its corresponding `FunctionSpace`. The bilinear form a^1 is then assembled by FEniCS to produce the discrete matrix $\mathbf{A}^1$ stored in sparse format.

Adding a second variable is then similar, except that constraints must now include the block-structure of both variables such as:

```
# Problem initialization
prob = MosekProblem("My problem")

# Adding a first block variable x1
x1 = prob.add_var(V1, lx=lx1, ux=ux1, cone=K1)
# Adding a first linear constraint
prob.add_ineq_constraint(W1, A=[a1], bl=bl1, bu=bu1)

# Adding a second block variable x2
x2 = prob.add_var(V2, lx=lx2, ux=ux2, cone=K2)
# Adding a second linear constraint
prob.add_ineq_constraint(W2, A=[a21, a22], bl=bl2, bu=bu2)
```

where we now have two bilinear forms $a^{21}(y^2, x^1)$ on $W_2 \times V_1$ and $a^{22}(y^2, x^2)$ on $W_2 \times V_2$ leading to:

$$\begin{aligned} \min_{(\mathbf{x}^1,\mathbf{x}^2)\in V_1\times V_2} \quad & 0 \\ \text{s.t.} \quad & \mathbf{l}_x^1 \leq \mathbf{x}^1 \leq \mathbf{u}_x^1 \\ & \mathbf{b}_l^1 \leq \mathbf{A}^1\mathbf{x}^1 \leq \mathbf{b}_u^1 \\ & \mathbf{l}_x^2 \leq \mathbf{x}^2 \leq \mathbf{u}_x^2 \\ & \mathbf{b}_l^2 \leq \mathbf{A}^{21}\mathbf{x}^1 + \mathbf{A}^{22}\mathbf{x}^2 \leq \mathbf{b}_u^2 \end{aligned} \tag{48}$$

Finally, a linear objective term $(\mathbf{c}^1)^{\mathrm{T}}\mathbf{x}^1 + (\mathbf{c}^2)^{\mathrm{T}}\mathbf{x}^2$ can be added as

```
prob.add_obj_fun([c1, c2])
```

The final block-structure for a problem with p blocks will therefore look like:

$$\begin{aligned} \min_{\mathbf{x}=(\mathbf{x}^1,\ldots,\mathbf{x}^p)\in V_1\times\ldots\times V_p} \quad & (\mathbf{c}^1,\ldots,\mathbf{c}^p)^{\mathrm{T}}(\mathbf{x}^1,\ldots,\mathbf{x}^p) \\ \text{s.t.} \quad & \mathbf{l}_x \leq \mathbf{x} \leq \mathbf{u}_x \\ & \mathbf{b}_l \leq \begin{bmatrix} \mathbf{A}^{11} & 0 & \ldots & 0 \\ \mathbf{A}^{21} & \mathbf{A}^{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{p1} & \mathbf{A}^{p2} & \ldots & \mathbf{A}^{pp} \end{bmatrix} \mathbf{x} \leq \mathbf{b}_u \end{aligned} \tag{49}$$

Note that when defining sequentially the block-constraints until variable $\mathbf{x}^i$, the blocks $\mathbf{A}^{ij}$ with $j > i$ are automatically zero since variables $\mathbf{x}^j$ have not been defined yet. This is by no means a restriction since one could perfectly define the constraint matrices once all variables have been defined. This lower-triangular structure allows however for an easier definition of the constraints in many cases. Note also that empty blocks can also be written with `0` or `None` in Python. Such symbols must be explicitly used for all $\mathbf{A}^{ij}$ with $j \leq i$.

A.2 Explicit construction of problem (24)

Going back to problem (24), one could do first:

```
# Problem initialization
prob = MosekProblem("Obstacle problem")

# Adding a first block variable u
u = prob.add_var(Vu, lx=g)
```

creating only variable $\mathbf{u}$ and its lower bound constraint $\mathbf{u} \geq \mathbf{g}$.

Auxiliary variable $\widehat{\mathbf{y}}$ corresponds to a 4-dimensional vectorial field with degrees of freedom located at quadrature points. FEniCS provides such a functional space through the concept of `Quadrature` elements. We will use one, noted `V2`, of dimension 4 for $\widehat{\mathbf{y}}$ and one, noted `W`, of dimension 3 for the Lagrange multipliers corresponding to constraints:

$$\begin{aligned} y_{g,1} &= 1 \\ \mathbf{B}_g\mathbf{u} - \begin{bmatrix} y_{g,2} \\ y_{g,3} \end{bmatrix} &= 0 \end{aligned}$$

Indeed, satisfying the above constraints for all Gauss points g is equivalent to writing:

$$a^{12}(z, u) + a^{22}(z, y) = b(z) \quad \forall z \in W \tag{50}$$

where

$$a^{12}(z,u) = \int_\Omega \begin{pmatrix} z_2 \\ z_3 \end{pmatrix} \cdot \nabla u \, \mathrm{dx}$$

$$a^{22}(z,y) = -\int_\Omega z \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \mathrm{dx}$$

$$b(z) = \int_\Omega z \cdot \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} \mathrm{dx}$$

in which the integrals are computed using the same quadrature used for defining $y \in V_2$ and $z \in W$. This results in the following code:

```
def quad_element(degree=0, dim=1):
    return VectorElement("Quadrature", mesh.ufl_cell(),
            degree=degree, dim=dim, quad_scheme="default")
V2 = FunctionSpace(mesh, quad_element(degree=0, dim=4))
W = FunctionSpace(mesh, quad_element(degree=0, dim=3))
y = prob.add_var(V2, cone = RQuad(4))

dxq = Measure("dx", metadata={"quadrature_scheme":"default",
                              "quadrature_degree":0})
def constraint(z):
    g = grad(u)
    a21 = dot(z, as_vector([0, g[0], g[1]]))*dxq
    a22 = -dot(z, as_vector([y[1], y[2], y[3]]))*dxq
    return [a21, a22]
def rhs(z):
    return -z[0]*dxq
prob.add_eq_constraint(W, A=constraint, b=rhs)
```

where $y \in V_2$ is created by specifying that it also belongs to a rotated quadratic cone Q_r^4. This statement is understood point-wise, meaning that at each degree of freedom (Gauss point) location x_g, the local 4-d vector $y(x_g)$ belongs to Q_r^4. The `dxq` measure is used to enforce a one-point quadrature on each cell, the same used for the definition of `V2` and `W`. Finally, the constraint matrix is passed as a function (`constraint`) of the Lagrange multiplier $z \in W$ and returns a list of 2 bilinear forms corresponding to both blocks in u and y, while the constraint right-hand side is also passed as a function of z (`rhs`) and returns a single linear form in z. A similar syntax would be used for inequality constraints.

Finally, the objective term is set as a list of two linear forms in u and y respectively:

```
prob.add_obj_func([-dot(load, u)*dx, y[0]*dxq])
```

Note again the use of the one-point quadrature measure for the second term.

One role of `ConvexFunction` classes described in 3.3 is to avoid for the user to explicitly define function spaces for the additional variables and Lagrange multipliers. The complete script of this implementation can be found in `demos/obstacle/obstacle_problem_explicit_construction`.

B CONIC REFORMULATION OF PROBLEM (36)

We consider function $\pi : M \in \mathbb{S}^2 \longmapsto \frac{2m}{\sqrt{3}}\sqrt{M_{11}^2 + M_{22}^2 + M_{12}^2 + M_{11}M_{22}}$. Expressing $M \in \mathbb{S}^2$ as $\mathbf{X} = (M_{11}, M_{22}, 2M_{12})$, we have that:

$$\pi(M) = \widehat{\pi}(\mathbf{X}) = \frac{m}{\sqrt{3}}\sqrt{\mathbf{X}^{\mathrm{T}}\mathbf{C}\mathbf{X}} \text{ with } \mathbf{C} = \begin{bmatrix} 4 & 2 & 0 \\ 2 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{51}$$

Computing the Cholesky factor $\mathbf{J} = \begin{bmatrix} 2 & 1 & 0 \\ 0 & \sqrt{3} & 0 \\ 0 & 0 & 1 \end{bmatrix}$ of matrix $\mathbf{C}$, we have that $\widehat{\pi}(\mathbf{X}) = \frac{m}{\sqrt{3}}\sqrt{\mathbf{X}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\mathbf{X}} = \frac{m}{\sqrt{3}}\|\mathbf{J}\mathbf{X}\|_2$.

REFERENCES

[1] Alnæs, M. S. (2012). *UFL: a Finite Element Form Language*, chapter 17. Springer.

[2] Alnæs, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N. (2015). The fenics project version 1.5. *Archive of Numerical Software*, 3(100).

[3] Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N. (2014). Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software*, 40(2).

[4] Andersen, E. D., Roos, C., and Terlaky, T. (2003). On implementing a primal-dual interior-point method for conic quadratic optimization. *Mathematical Programming*, 95(2):249–277.

[5] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2016). PETSc Web page. http://www.mcs.anl.gov/petsc.

[6] Balmforth, N. J., Frigaard, I. A., and Ovarlez, G. (2014). Yielding to stress: recent developments in viscoplastic fluid mechanics. *Annual Review of Fluid Mechanics*, 46:121–146.

[7] Ben-Tal, A., El Ghaoui, L., and Nemirovski, A. (2009). *Robust optimization*, volume 28. Princeton University Press.

[8] Ben-Tal, A. and Nemirovski, A. (2002). Robust optimization–methodology and applications. *Mathematical Programming*, 92(3):453–480.

[9] Benamou, J.-D. and Brenier, Y. (2000). A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393.

[10] Bendsoe, M. P. and Sigmund, O. (2004). *Topology Optimization: Theory, Methods and Applications*. Springer.

[11] Bleyer, J. (2018). Advances in the simulation of viscoplastic fluid flows using interior-point methods. *Computer Methods in Applied Mechanics and Engineering*, 330:368–394.

[12] Bleyer, J., Carlier, G., Duval, V., Mirebeau, J.-M., and Peyré, G. (2016). A γ-convergence result for the upper bound limit analysis of plates. *ESAIM: Mathematical Modelling and Numerical Analysis*, 50(1):215–235.

[13] Bleyer, J. and de Buhan, P. (2013). On the performance of non-conforming finite elements for the upper bound limit analysis of plates. *International Journal for Numerical Methods in Engineering*, 94(3):308–330.

[14] Bleyer, J., Maillard, M., De Buhan, P., and Coussot, P. (2015). Efficient numerical computations of yield stress fluid flows using second-order cone programming. *Computer Methods in Applied Mechanics and Engineering*, 283:599–614.

[15] Capsoni, A. and Corradi, L. (1999). Limit analysis of plates- a finite element formulation. *Structural Engineering and Mechanics*, 8(4):325–341.

[16] Carlier, G., Comte, M., Ionescu, I., and Peyré, G. (2011). A projection approach to the numerical analysis of limit load problems. *Mathematical Models and Methods in Applied Sciences*, 21(06):1291–1316.

[17] Carlier, G., Comte, M., and Peyré, G. (2009). Approximation of maximal cheeger sets by projection. *ESAIM: Mathematical Modelling and Numerical Analysis*, 43(1):139–150.

[18] Chambolle, A. and Pock, T. (2018). Crouzeix-raviart approximation of the total variation on simplicial meshes. hal-01787012.

[19] Chares, R. (2009). *Cones and interior-point algorithms for structured convex optimization involving powers andexponentials*. PhD thesis, Ph. D. Thesis, UCL-Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

[20] Cheeger, J. (1969). A lower bound for the smallest eigenvalue of the laplacian. In *Proceedings of the Princeton conference in honor of Professor S. Bochner*.

[21] Coussot, P. (2016). Bingham's heritage. *Rheologica Acta*, 6(3):163–176.

[22] Demengel, F. (1983). Problemes variationnels en plasticité parfaite des plaques. *Numerical Functional Analysis and Optimization*, 6(1):73–119.
[23] Demengel, F. (1984). Fonctions à hessien borné. In *Annales de l'institut Fourier*, volume 34, pages 155–190.
[24] Dumont, S. and Igbida, N. (2009). On a dual formulation for the growing sandpile problem. *European Journal of Applied Mathematics*, 20(2):169–185.
[25] Duvaut, G. and Lions, J. L. (2012). *Inequalities in mechanics and physics*, volume 219. Springer Science & Business Media.
[26] Glowinski, R. (1984). *Numerical Methods for Nonlinear Variational Problems*, volume 158 of *Springer Series in Computational Physics*. Springer-Verlag Berlin Heidelberg.
[27] Kanno, Y. (2011). *Nonsmooth mechanics and convex optimization*. Crc Press.
[28] Kanno, Y. and Guo, X. (2010). A mixed integer programming for robust truss topology optimization with stress constraints. *International Journal for Numerical Methods in Engineering*, 83(13):1675–1699.
[29] Kawohl, B. and Novaga, M. (2008). The p-laplace eigenvalue problem as p→ 1 and cheeger sets in a finsler metric. *Journal of Convex Analysis*, 15(3):623.
[30] Kinderlehrer, D. and Stampacchia, G. (1980). *An introduction to variational inequalities and their applications*, volume 31. Siam.
[31] Kirby, R. C. and Logg, A. (2006). A compiler for variational forms. *ACM Transactions on Mathematical Software*, 32(3).
[32] Logg, A., Mardal, K.-A., and Wells, G. (2012a). *Automated solution of differential equations by the finite element method: The FEniCS book*, volume 84. Springer Science & Business Media.
[33] Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N. (2012b). *FFC: the FEniCS Form Compiler*, chapter 11. Springer.
[34] Logg, A. and Wells, G. N. (2010). Dolfin: Automated finite element computing. *ACM Transactions on Mathematical Software*, 37(2).
[35] Logg, A., Wells, G. N., and Hake, J. (2012c). *DOLFIN: a C++/Python Finite Element Library*, chapter 10. Springer.
[36] Malanowski, K. (1982). Convergence of approximations vs. regularity of solutions for convex, control-constrained optimal-control problems. *Applied Mathematics and Optimization*, 8(1):69–95.
[37] Meyer, Y. (2001). *Oscillating patterns in image processing and nonlinear evolution equations: the fifteenth Dean Jacqueline B. Lewis memorial lectures*, volume 22. American Mathematical Soc.
[38] MOSEK ApS, . (2018). *The MOSEK optimization API for Python 8.1.0*.
[39] Munson, T., Sarich, J., Wild, S., Benson, S., and McInnes, L. C. (2012). Tao 2.0 users manual.
[40] Overton, M. L. (1985). Numerical solution of a model problem from collapse load analysis. In *Proc. of the sixth int'l. symposium on Computing methods in applied sciences and engineering, VI*, pages 421–437. North-Holland Publishing Co.
[41] Papadakis, N., Peyré, G., and Oudet, E. (2014). Optimal transport with proximal splitting. *SIAM Journal on Imaging Sciences*, 7(1):212–238.
[42] Peyré, G., Cuturi, M., et al. (2019). Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607.
[43] Prigozhin, L. (1996). Variational model of sandpile growth. *European Journal of Applied Mathematics*, 7(3):225–235.
[44] Strang, G. (1979). A minimax problem in plasticity theory. In *Functional analysis methods in numerical analysis*, pages 319–333. Springer.
[45] Trémolières, R., Lions, J.-L., and Glowinski, R. (2011). *Numerical analysis of variational inequalities*, volume 8. Elsevier.
[46] Villani, C. (2003). *Topics in optimal transportation*. Number 58. American Mathematical Soc.
[47] Weiss, P., Blanc-Féraud, L., and Aubert, G. (2009). Efficient schemes for total variation minimization under constraints in image processing. *SIAM journal on Scientific Computing*, 31(3):2047–2080.
[48] Yonekura, K. and Kanno, Y. (2010). Global optimization of robust truss topology via mixed integer semidefinite programming. *Optimization and Engineering*, 11(3):355–379.