Automating the formulation and resolution of convex variational problems with the `fenics_optim` package

Jeremy Bleyer

Laboratoire Navier, UMR 8205
Ecole des Ponts ParisTech-Univ Gustave Eiffel-CNRS

FEniCS 2021 Conference

Convex variational problems

variational inequalities arise in presence of contact, unilateral conditions (phase-field), plasticity...

$$\begin{array}{ll} \inf\limits_{u\in V} & J(u) \\ \text{s.t.} & u \in \mathcal{K} \end{array}$$

J convex function, $\mathcal{K}$ convex set

Convex variational problems

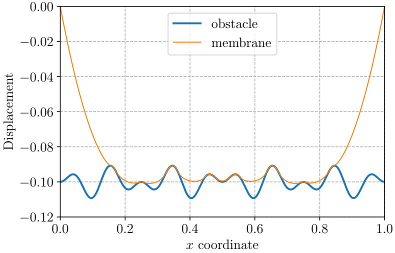
variational inequalities arise in presence of contact, unilateral conditions (phase-field), plasticity...

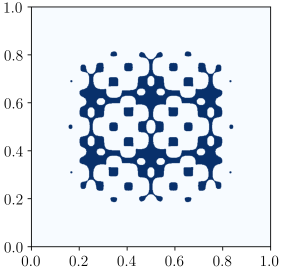
$$\begin{array}{ll} \inf\limits_{u\in V} & J(u) \\ \text{s.t.} & u \in \mathcal{K} \end{array}$$

J convex function, $\mathcal{K}$ convex set

e.g. **obstacle problem**:

$$\begin{array}{ll} \inf\limits_{u\in V} & \displaystyle\int_\Omega \frac{1}{2}\|\nabla u\|_2^2 \,\mathrm{dx} - \int_\Omega fu \,\mathrm{dx} \\ \text{s.t.} & u \geq g \text{ on } \Omega \end{array}$$





Conic optimization problems

problems become difficult to solve when J is **non-smooth** (or $\mathcal{K}$ complicated)

$$\begin{array}{ll} \inf\limits_{u\in V} & J(u) \\ \text{s.t.} & u \in \mathcal{K} \end{array}$$

Conic optimization problems

problems become difficult to solve when J is **non-smooth** (or $\mathcal{K}$ complicated)

$$\inf_{u\in V} \quad J(u) + \delta_{\mathcal{K}}(u) =: \widetilde{J}(u)$$

Conic optimization problems

problems become difficult to solve when J is **non-smooth** (or $\mathcal{K}$ complicated)

$$\begin{aligned} \inf_{u\in V,t} \quad & t \\ \text{s.t.} \quad & \widetilde{J}(u) \leq t \end{aligned}$$

Conic optimization problems

problems become difficult to solve when J is **non-smooth** (or $\mathcal{K}$ complicated)

$$\begin{aligned} \inf_{u\in V, t} \quad & t \\ \text{s.t.} \quad & \widetilde{J}(u) \leq t \end{aligned}$$

Conic optimization

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{b}_l \leq \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}_u \\ & \boldsymbol{x} \in \mathcal{K}^1 \times \ldots \times \mathcal{K}^p \end{aligned}$$

where $\mathcal{K}^j$ are **simple cones**

Conic optimization problems

problems become difficult to solve when J is **non-smooth** (or $\mathcal{K}$ complicated)

$$\begin{aligned} \inf_{u\in V,t} \quad & t \\ \text{s.t.} \quad & \widetilde{J}(u) \leq t \end{aligned}$$

Conic optimization

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{b}_l \leq \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}_u \\ & \boldsymbol{x} \in \mathcal{K}^1 \times \ldots \times \mathcal{K}^p \end{aligned}$$

where $\mathcal{K}^j$ are **simple cones**

- positive orthant : $\mathcal{K}^j = \mathbb{R}^{m+} = \{\boldsymbol{z} \in \mathbb{R}^m \text{ s.t. } z_i \geq 0\}$ => LP
- Lorentz second-order ("ice-cream") cone :

$$\mathcal{K}^j = \mathcal{Q}_m = \{\boldsymbol{z} = (z_0, \bar{\boldsymbol{z}}) \in \mathbb{R} \times \mathbb{R}^{m-1} \text{ s.t. } \|\bar{\boldsymbol{z}}\| \leq z_0\} \text{ => SOCP}$$

- cone of positive semi-definite matrix $\mathrm{X} \succeq 0$ => SDP

Conic optimization problems

problems become difficult to solve when J is **non-smooth** (or $\mathcal{K}$ complicated)

$$\begin{aligned} \inf_{u\in V,t} \quad & t \\ \text{s.t.} \quad & \widetilde{J}(u) \leq t \end{aligned}$$

Conic optimization

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{c}^\mathsf{T}\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{b}_l \leq \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}_u \\ & \boldsymbol{x} \in \mathcal{K}^1 \times \ldots \times \mathcal{K}^p \end{aligned}$$

where $\mathcal{K}^j$ are **simple cones**

- positive orthant : $\mathcal{K}^j = \mathbb{R}^{m+} = \{\boldsymbol{z} \in \mathbb{R}^m \text{ s.t. } z_i \geq 0\}$ => LP
- Lorentz second-order ("ice-cream") cone :

$$\mathcal{K}^j = \mathcal{Q}_m = \{\boldsymbol{z} = (z_0, \bar{\boldsymbol{z}}) \in \mathbb{R} \times \mathbb{R}^{m-1} \text{ s.t. } \|\bar{\boldsymbol{z}}\| \leq z_0\}$$ => SOCP

- cone of positive semi-definite matrix $\mathrm{X} \succeq 0$ => SDP

State-of-the-art **interior point** solvers: CPLEX, MOSEK, CVXOPT

A more advanced problem

$$\begin{aligned} c_\Omega = \inf_{u\in V_0} & \int_\Omega \|\nabla u\|_2 \,\mathrm{dx} \\ \text{s.t.} & \int_\Omega fu \,\mathrm{dx} = 1 \end{aligned}$$

antiplane limit analysis, Cheeger problem, first eigenvalue of the 1-Laplacian

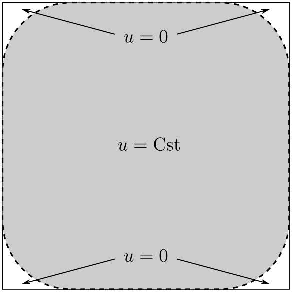
A more advanced problem

$$\begin{aligned} c_\Omega = \inf_{u\in V_0} \quad & \int_\Omega \|\nabla u\|_2 \text{ dx} \\ \text{s.t.} \quad & \int_\Omega fu \text{ dx} = 1 \end{aligned}$$

antiplane limit analysis, Cheeger problem, first eigenvalue of the 1-Laplacian

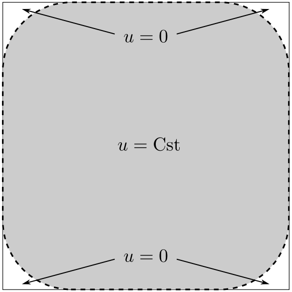
Difficulties:

- $\cancel{\|\nabla u\|_2^2}$ but $\|\nabla u\|_2 \Rightarrow$ **non-smooth**
- **discontinuous** solution $\Rightarrow$ discretization ?



A more advanced problem

$$c_\Omega = \inf_{u\in V_0} \int_\Omega \|\nabla u\|_2 \text{ dx}$$
$$\text{s.t.} \int_\Omega fu \text{ dx} = 1$$



antiplane limit analysis, Cheeger problem, first eigenvalue of the 1-Laplacian

Difficulties:

- $\cancel{\|\nabla u\|_2^2}$ but $\|\nabla u\|_2 \Rightarrow$ **non-smooth**
- **discontinuous** solution $\Rightarrow$ discretization ?

Conic reformulation:

$$\begin{aligned} \inf_{u\in V_0, \boldsymbol{z}} \quad & \int_\Omega z_0 \text{ dx} \\ \text{s.t.} \quad & \int_\Omega fu \text{ dx} = 1 \\ & \bar{\boldsymbol{z}} = \nabla u \\ & \|\bar{\boldsymbol{z}}\|_2 \leq z_0 \Leftrightarrow \boldsymbol{z} \in \mathcal{Q}_{d+1} \end{aligned}$$

(**SOCP** problem)

Conic-representable functions and the `fenics_optim` package

A convex function $F(\boldsymbol{x})$ will be *conic-representable* if it can be written as:

$$\begin{array}{rl} F(\boldsymbol{x}) = \displaystyle\min_{\boldsymbol{y}} & \boldsymbol{c}_x\boldsymbol{x} + \boldsymbol{c}_y\boldsymbol{y} \\ \text{s.t.} & \boldsymbol{b}_l \leq \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} \leq \boldsymbol{b}_u \\ & \boldsymbol{y} \in \mathcal{K}^1 \times \ldots \times \mathcal{K}^p \end{array}$$

Conic-representable functions and the `fenics_optim` package

A convex function $F(\boldsymbol{x})$ will be *conic-representable* if it can be written as:

$$\begin{array}{rl} F(\boldsymbol{x}) = \displaystyle\min_{\boldsymbol{y}} & \boldsymbol{c}_x\boldsymbol{x} + \boldsymbol{c}_y\boldsymbol{y} \\ \text{s.t.} & \boldsymbol{b}_l \leq \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} \leq \boldsymbol{b}_u \\ & \boldsymbol{y} \in \mathcal{K}^1 \times \ldots \times \mathcal{K}^p \end{array}$$

`fenics_optim` package dedicated to solving problems involving:

$$J(u) = \sum_{i=1}^{n} \int_\Omega F_i(\ell_i(u)) \, \mathrm{dx}$$

where F_i are *conic-representable* and ℓ_i are UFL-representable linear operators

Conic-representable functions and the `fenics_optim` package

A convex function $F(\boldsymbol{x})$ will be *conic-representable* if it can be written as:

$$\begin{array}{rl} F(\boldsymbol{x}) = \displaystyle\min_{\boldsymbol{y}} & \boldsymbol{c}_x\boldsymbol{x} + \boldsymbol{c}_y\boldsymbol{y} \\ \text{s.t.} & \boldsymbol{b}_l \leq \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{y} \leq \boldsymbol{b}_u \\ & \boldsymbol{y} \in \mathcal{K}^1 \times \ldots \times \mathcal{K}^p \end{array}$$

`fenics_optim` package dedicated to solving problems involving:

$$J(u) = \sum_{i=1}^{n} \int_\Omega F_i(\ell_i(u)) \,\mathrm{dx}$$

where F_i are *conic-representable* and ℓ_i are UFL-representable linear operators

Choice of a **quadrature rule**: $J(u) = \displaystyle\int_\Omega F(\ell(u)) \,\mathrm{dx} \approx \sum_{g=1}^{N_g} \omega_g F(\boldsymbol{L}_g\boldsymbol{u})$

$$\Rightarrow \qquad \begin{array}{rl} \displaystyle\min_{\boldsymbol{u}} J(\boldsymbol{u}) = \min_{\boldsymbol{u},\boldsymbol{y}_g} & \displaystyle\sum_{g=1}^{N_g} \omega_g(\boldsymbol{c}_x\boldsymbol{L}_g\boldsymbol{u} + \boldsymbol{c}_y\boldsymbol{y}_g) \\ \text{s.t.} & \boldsymbol{b}_l \leq \boldsymbol{A}\boldsymbol{L}_g\boldsymbol{x}_g + \boldsymbol{B}\boldsymbol{y}_g \leq \boldsymbol{b}_u \\ & \boldsymbol{y}_g \in \mathcal{K}^1 \times \ldots \times \mathcal{K}^p \end{array}$$

Example on the Cheeger problem

auxiliary variables will be implicitly declared on a Quadrature space

```
class L2Norm(ConvexFunction):
    def conic_repr(self, X):
        d = self.dim_x
        z = self.add_var(d+1, cone=Quad(d+1))
        zbar = as_vector([z[i]
                    for i in range(1, d+1)])
        self.add_eq_constraint(X - zbar)
        self.set_linear_term(z[0])
```

```
V = FunctionSpace(mesh, "CG", 1)
prob = MosekProblem("Cheeger problem")
u = prob.add_var(V, bc=bc)

F = L2Norm(grad(u), degree=0)
prob.add_convex_term(F)

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)

prob.optimize()
```

CG1

Example on the Cheeger problem

auxiliary variables will be implicitly declared on a Quadrature space

```
class L2Norm(ConvexFunction):
    def conic_repr(self, X):
        d = self.dim_x
        z = self.add_var(d+1, cone=Quad(d+1))
        zbar = as_vector([z[i]
                  for i in range(1, d+1)])
        self.add_eq_constraint(X - zbar)
        self.set_linear_term(z[0])
```

```
V = FunctionSpace(mesh, "CG", 1)
prob = MosekProblem("Cheeger problem")
u = prob.add_var(V, bc=bc)

F = L2Norm(grad(u), degree=0)
prob.add_convex_term(F)

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)

prob.optimize()
```

CG2

Example on the Cheeger problem

auxiliary variables will be implicitly declared on a `Quadrature` space

```
class L2Norm(ConvexFunction):
    def conic_repr(self, X):
        d = self.dim_x
        z = self.add_var(d+1, cone=Quad(d+1))
        zbar = as_vector([z[i]
                    for i in range(1, d+1)])
        self.add_eq_constraint(X - zbar)
        self.set_linear_term(z[0])
```

```
V = FunctionSpace(mesh, "CG", 1)
prob = MosekProblem("Cheeger problem")
u = prob.add_var(V, bc=bc)

F = L2Norm(grad(u), degree=0)
prob.add_convex_term(F)

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)

prob.optimize()
```

DG1

also works with **facet measures**

Example on the dual Cheeger problem

dual problem with the same objective

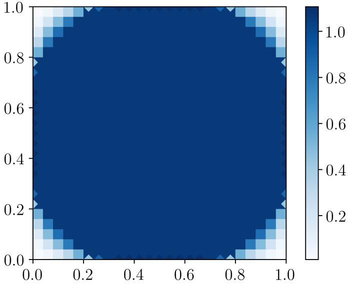
$$\begin{aligned} c_\Omega = \sup_{\lambda\in\mathbb{R},\boldsymbol{\sigma}\in W} \quad & \lambda \\ \text{s.t.} \quad & \lambda f = \operatorname{div}\boldsymbol{\sigma} \quad \text{in } \Omega \\ & \|\boldsymbol{\sigma}\|_2 \leq 1 \end{aligned}$$

Example on the dual Cheeger problem

dual problem with the same objective

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

$\Rightarrow$ $H(div)$-conforming discretization with **RT** elements (lower bound)

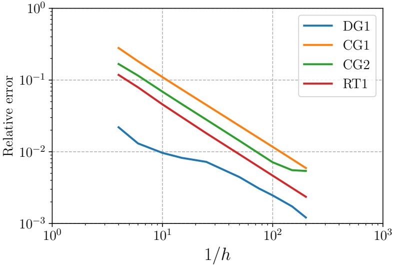



```
prob = MosekProblem("Cheeger dual")
lamb, sig = prob.add_var([R, VRT])

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

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

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



Variational cartoon/texture decomposition

Image $y = u$ (cartoon) $+ v$ (texture)
Meyer's model (TV + G-norm):

$$\begin{array}{ll} \inf\limits_{u,v} & \displaystyle\int_\Omega \|\nabla u\|_2 \,\mathrm{dx} + \alpha\|v\|_G \\ \text{s.t.} & y = u + v \end{array}$$

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

reformulated as:

$$\begin{array}{ll} \inf\limits_{u,\boldsymbol{g}} & \displaystyle\int_\Omega \|\nabla u\|_2 \,\mathrm{dx} \\ \text{s.t.} & y = u + \operatorname{div}(\boldsymbol{g}) \\ & \|\sqrt{g_1^2+g_2^2}\|_\infty \leq \alpha \end{array}$$

L_2 ad $L_{\infty,2}$-norms are **conic-representable** $\Rightarrow$ SOCP problem

Variational cartoon/texture decomposition

Image y : represented by a $DG0$ field on a 512x512 finite-element mesh
$u, \boldsymbol{g} \in \mathtt{CR} \times \mathtt{RT}$

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

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

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

prob.optimize()
```

Variational cartoon/texture decomposition

Image y : represented by a $DG0$ field on a 512x512 finite-element mesh

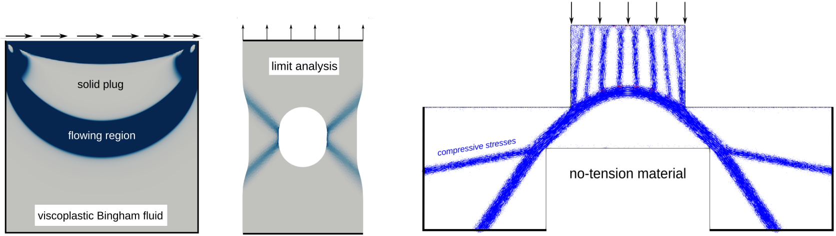
$u, \boldsymbol{g} \in \mathtt{CR} \times \mathtt{RT}$

Barbara image

Conclusions

Package available at `https://gitlab.enpc.fr/navier-fenics/fenics-optim`

- UFL syntax for **conic-representable functions**
- supports **LP**, **SOCP**, **SDP**, exponential and power cones via Mosek
- **other applications** : viscoplastic fluids, limit analysis, topology optimization, nonlinear membranes/shells, inpainting, optimal transport, etc.



Perspectives

- other IPM solvers, custom solver ?
- first-order solvers (**proximal algorithms**)
- porting to `dolfin-x`

Bleyer J., TOMS, 46(3), 1-33. 2020