CvxLean

Modeling convex optimization problems in Lean

Ramon Fernández Mir

University of Edinburgh

January 12, 2024

Motivation

Convex optimization problems are ubiquitous in engineering, industry, and finance. Some applications include

- safety and stability analysis of control systems
- power control
- portfolio optimization
- electronic circuit design

It is a generalization of linear programming that can be solved efficiently using interior-point methods.

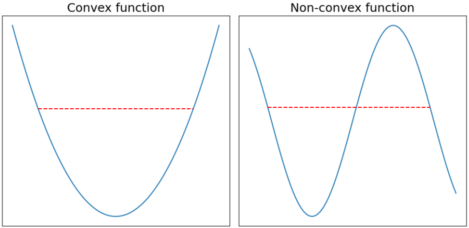
There is often a large gap between the original problem and the problem eventually solved.

Lean can help bridge this gap more reliably than existing tools.

Convex optimization

Optimizing over $x \in \mathbb{R}^n$ with convex f_is and affine h_is, a convex optimization problem in *standard form* (high-level) is

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \le 0, \quad i = 1, \ldots, k \\ & h_i(x) = 0, \quad i = 1, \ldots, l. \end{array}$$



Convex optimization

Optimizing over $x \in \mathbb{R}^n$ with convex f_is and affine h_is, a convex optimization problem in *standard form* (high-level) is

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, k \\ & h_i(x) = 0, \quad i = 1, \ldots, l. \end{array}$$

However, solvers want the problem in *conic form* (low-level)

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{K}, \end{array}$$

where $\mathcal{K}$ is a convex cone e.g. $\mathbb{R}^n_+$, $\mathcal{Q}^n$, $\mathcal{K}_{\text{exp}}$, etc.

Disciplined convex programming

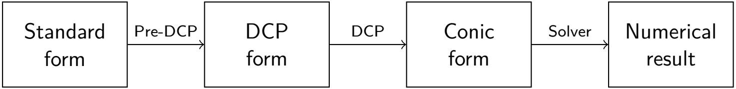
A framework to systematically reduce problems to conic form.

The framework consists of

- a library of base functions with known curvature and monotonicity called *atoms*
- top-level, product-free, sign and composition rules (*DCP form*)
- optimization problems representing each atom called *graph implementations* e.g. $e^x \mapsto \min\{t \mid (x, 1, t) \in \mathcal{K}_{\mathsf{exp}}\}$

Key idea: replace expressions not in conic form with equivalent optimization problems in conic form.

Convex optimization workflow



Before our work:

- Pre-DCP transformations are done by hand.
- DCP transformations are done using a modeling framework such as `CVXPY`, `CVX`, `Convex.jl`, etc.
- There are many conic solvers: `MOSEK`, `Clarabel`, `ECOS`, Gurobi, `SDPA`, etc.

Issues

Issues:

- Pre-DCP transformations have no guarantees.
- DCP transformations are hard-coded and syntactic, and side conditions are not checked in a principled way. They rely on properties of the atoms that are assumed but not verified.

Issues

Issues:

- Pre-DCP transformations have no guarantees.
- DCP transformations are hard-coded and syntactic, and side conditions are not checked in a principled way. They rely on properties of the atoms that are assumed but not verified.

Solution: `CvxLean`, a framework that supports verified and interactive pre-DCP and DCP transformations.

CvxLean overview

The type of minimization problems

```
structure Minimization where
  objFun : D → R
  constraints : D → Prop
```

The type of solutions

```
structure Solution (p : Minimization D R) where
  point : D
  feasibility : p.constraints point
  optimality : ∀ y, p.constraints y → p.objFun point ≤ p.objFun y
```

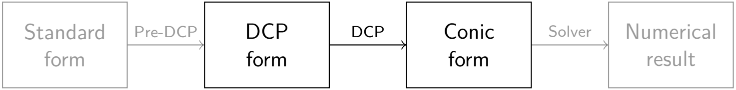
CvxLean overview

A problem defined in CvxLean

```
def prob :=
  optimization (x y : ℝ)
      minimize -sqrt (x - y)
      subject to
        c1 : y = 2*x - 3
        c2 : x^2 ≤ 2
        c3 : 0 ≤ x - y

#check prob -- Minimization (ℝ × ℝ) ℝ
```

Transformation to conic form



Algorithm: check DCP rules and iteratively replace atom applications with their graph implementations.

For example, we can replace $\sqrt{x-y}$ by a new variable t and add the constraint $(x-y, 0.5, t) \in \mathcal{Q}_r^3$, which holds iff $t^2 \leq x - y$.

The idea is that every eliminable atom includes proofs of key properties that can be combined to prove the overall equivalence.

Atom declaration

The atom declaration for $\sqrt{\cdot}$, with its four key proof obligations.

```
declare_atom sqrt [concave] (x : ℝ)+ : sqrt x :=
  vconditions (cond : 0 ≤ x)
  implementationVars (t : ℝ)
  implementationObjective (t)
  implementationConstraints (c1 : rotatedSoCone x 0.5 ![t])
  solution (t := Real.sqrt x)
  solutionEqualsAtom by ...
  -- ∀ a, 0 ≤ a → sqrt a = sqrt a
  feasibility (c1 : by ...)
  -- ∀ a, 0 ≤ a → rotatedSoCone a 0.5 ![sqrt a]
  optimality by ...
  -- ∀ v a a', a ≤ a' → rotatedSoCone a 0.5 ![v] → v ≤ sqrt a'
  vconditionElimination (cond : by ...)
  -- ∀ v a a', a ≤ a' → rotatedSoCone a 0.5 ![v] → 0 ≤ a'
```

Atom declaration

The atom declaration for $\sqrt{\cdot}$, with its four key proof obligations.

```
declare_atom sqrt [concave] (x : ℝ)+ : sqrt x :=
  vconditions (cond : 0 ≤ x)
  implementationVars (t : ℝ)
  implementationObjective (t)
  implementationConstraints (c1 : rotatedSoCone x 0.5 ![t])
  solution (t := Real.sqrt x)
  solutionEqualsAtom by ...
  -- ∀ a, 0 ≤ a → sqrt a = sqrt a
  feasibility (c1 : by ...)
  -- ∀ a, 0 ≤ a → rotatedSoCone a 0.5 ![sqrt a]
  optimality by ...
  -- ∀ v a a', a ≤ a' → rotatedSoCone a 0.5 ![v] → v ≤ sqrt a'
  vconditionElimination (cond : by ...)
  -- ∀ v a a', a ≤ a' → rotatedSoCone a 0.5 ![v] → 0 ≤ a'
```

Example

The original problem

```
optimization (x y : ℝ)
  minimize -sqrt (x - y)
  subject to
    c1 : y = 2*x - 3
    c2 : x^2 ≤ 2
    c3 : 0 ≤ x - y
```

is reduced to

```
optimization (x y t.0 t.1 : ℝ)
  minimize -t.0
  subject to
    c1' : zeroCone (2*x - 3 - y)             -- 2*x - 3 - y = 0
    c2' : posOrthCone (2 - t.1)              -- 0 ≤ 2 - t.1
    c4' : rotatedSoCone (x - y) 0.5 ![t.0]   -- t.0^2 ≤ x - y
    c5' : rotatedSoCone t.1 0.5 ![x]         -- x^2 ≤ t.1
```

Example

The original problem

```
optimization (x y : ℝ)
  minimize -sqrt (x - y)
  subject to
    c1 : y = 2*x - 3
    c2 : x^2 ≤ 2
    c3 : 0 ≤ x - y
```

is reduced to

```
optimization (x y t.0 t.1 : ℝ)
  minimize -t.0
  subject to
    c1' : zeroCone (2*x - 3 - y)              -- 2*x - 3 - y = 0
    c2' : posOrthCone (2 - t.1)               -- 0 ≤ 2 - t.1
    c4' : rotatedSoCone (x - y) 0.5 ![t.0]    -- t.0^2 ≤ x - y
    c5' : rotatedSoCone t.1 0.5 ![x]          -- x^2 ≤ t.1
```

Example

The original problem

```
optimization (x y : ℝ)
  minimize -sqrt (x - y)
  subject to
    c1 : y = 2*x - 3
    c2 : x^2 ≤ 2
    c3 : 0 ≤ x - y
```

is reduced to

```
optimization (x y t.0 t.1 : ℝ)
  minimize -t.0
  subject to
    c1' : zeroCone (2*x - 3 - y)              -- 2*x - 3 - y = 0
    c2' : posOrthCone (2 - t.1)               -- 0 ≤ 2 - t.1
    c4' : rotatedSoCone (x - y) 0.5 ![t.0]    -- t.0^2 ≤ x - y
    c5' : rotatedSoCone t.1 0.5 ![x]          -- x^2 ≤ t.1
```

Example

The original problem

```
optimization (x y : ℝ)
  minimize -sqrt (x - y)
  subject to
    c1 : y = 2*x - 3
    c2 : x^2 ≤ 2
    c3 : 0 ≤ x - y
```

is reduced to

```
optimization (x y t.0 t.1 : ℝ)
  minimize -t.0
  subject to
    c1' : zeroCone (2*x - 3 - y)              -- 2*x - 3 - y = 0
    c2' : posOrthCone (2 - t.1)               -- 0 ≤ 2 - t.1
    c4' : rotatedSoCone (x - y) 0.5 ![t.0]    -- t.0^2 ≤ x - y
    c5' : rotatedSoCone t.1 0.5 ![x]          -- x^2 ≤ t.1
```

The solve command

```
solve prob

#print prob.reduced   -- shows the reduced problem

#eval prob.status     -- "PRIMAL_AND_DUAL_FEASIBLE"
#eval prob.value      -- 2.101003
#eval prob.solution   -- (-1.414214, -5.828427)
```

The `solve` command reduces the problem, sends it to MOSEK, and reconstructs the point in the original domain.

New feature: multilevel atom declarations

The atom for the Huber loss function

$$h(x) := \begin{cases} x^2 & \text{if } |x| \leq 1 \\ 2x - 1 & \text{if } |x| \geq 1 \end{cases}$$

is declared as follows:

```
declare_atom huber [convex] (x : ℝ)? : huber x :=
  vconditions
  implementationVars (v : ℝ) (w : ℝ)
  implementationObjective (2*v + w^2)
  implementationConstraints
    (c1 : |x| ≤ v + w)
    (c2 : w ≤ 1)
    (c3 : 0 ≤ v)
  solution
    (v := if |x| ≤ 1 then 0 else |x| - 1)
    (w := if |x| ≤ 1 then |x| else 1)
  ...
```

New feature: multilevel atom declarations

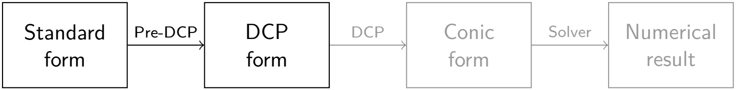
The atom for the Huber loss function

$$h(x) := \begin{cases} x^2 & \text{if } |x| \leq 1 \\ 2x - 1 & \text{if } |x| \geq 1 \end{cases}$$

is declared as follows:

```
declare_atom huber [convex] (x : ℝ)? : huber x :=
  vconditions
  implementationVars (v : ℝ) (w : ℝ)
  implementationObjective (2*v + w^2)
  implementationConstraints
    (c1 : |x| ≤ v + w)
    (c2 : w ≤ 1)
    (c3 : 0 ≤ v)
  solution
    (v := if |x| ≤ 1 then 0 else |x| - 1)
    (w := if |x| ≤ 1 then |x| else 1)
  ...
```

New feature: automatic transformation to DCP form



Take the following problem:

```
optimization (x : ℝ)
  minimize x
  subject to
    c1 : 1 / 1000 ≤ x
    c2 : 1 / sqrt x ≤ exp x
```

It is not DCP because `c2` is not of the form "convex" $\leq$ "concave".

The equivalent constraint `exp (-x) ≤ sqrt x` is DCP.

We have support to rewrite it manually, but can we automate it?

E-graphs for optimization problems

An e-graph compactly represents a set of equivalent terms w.r.t. some rewrite rules. Each rule application updates it by adding nodes and merging equivalence classes.

They are useful when the rewriting system is complex, has no normal forms, and there is no clear heuristic to guide the search.

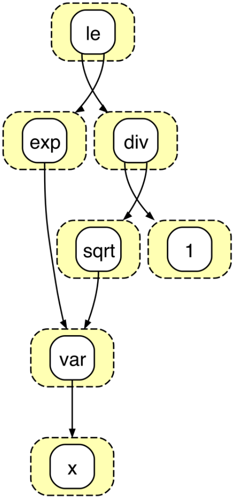
We use egg, a high-performance e-graph library written in Rust.

We provide:

- A language to encode optimization problems.
- A list of (currently) 67 rewrite rules that we found useful.
- Support for conditional rewrites.
- Mechanisms to detect terms in DCP form.

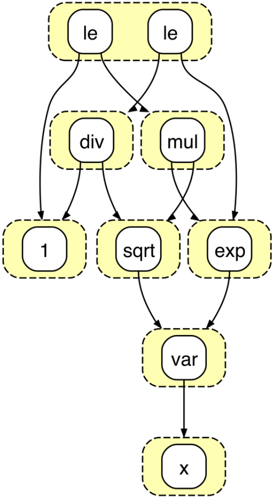
Building an e-graph step by step

Initial e-graph representing $1/\sqrt{x} \leq e^x$.



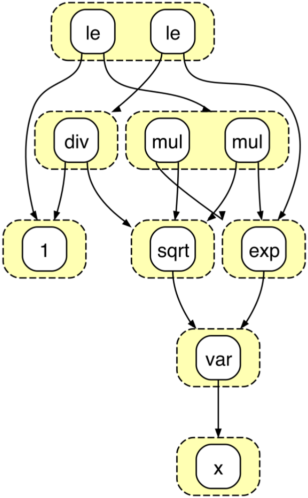
Building an e-graph step by step

After rewriting $1/\sqrt{x} \leq e^x$ into $1 \leq e^x\sqrt{x}$.



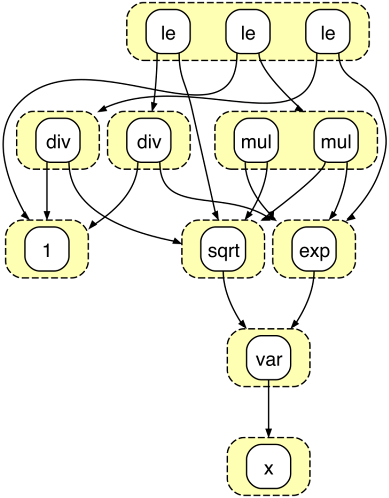
Building an e-graph step by step

After rewriting $e^x\sqrt{x}$ into $\sqrt{x}e^x$.



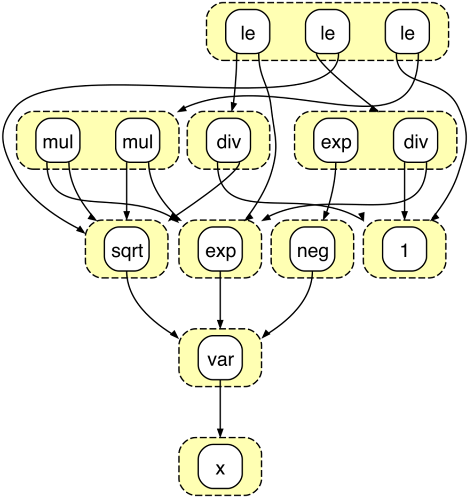
Building an e-graph step by step

After rewriting $1 \leq \sqrt{x}e^x$ into $1/e^x \leq \sqrt{x}$.



Building an e-graph step by step

After rewriting $1/e^x$ into e^{-x}.



Extracting the "best" problem

The e-graph represents a set of equivalent optimization problems.

We can extract the term that minimizes some cost. In our case, the cost is the curvature, calculated following the DCP rules. If one of the problems is tagged as convex, then it is in DCP form, and egg gives us the sequence of rewrites to get there, e.g.,

$$\frac{1}{\sqrt{x}} \leq e^x \quad \rightsquigarrow \quad 1 \leq e^x\sqrt{x} \quad \rightsquigarrow \quad 1 \leq \sqrt{x}e^x$$
$$\cdots \quad \rightsquigarrow \quad \frac{1}{e^x} \leq \sqrt{x} \quad \rightsquigarrow \quad e^{-x} \leq \sqrt{x}.$$

Replaying the proof in Lean

See `https://github.com/opencompl/egg-tactic-code` by Andrés Goens and Siddharth Bhat.

Every rewrite rule corresponds to a lemma. For example, the first step in the previous slide involves applying `div_le_iff`.

```
theorem div_le_iff (hb : 0 < b) : a / b ≤ c ↔ a ≤ c * b := ...
```

Note that it is a conditional rewrite. We keep track of the range of each expression in egg using interval arithmetic, so it can detect that $0 < \sqrt{x}$ in our example.

In Lean, we do our best to discharge the side condition using `positivity` and `nlinarith` (maybe `by_approx` in the future?).

Demo

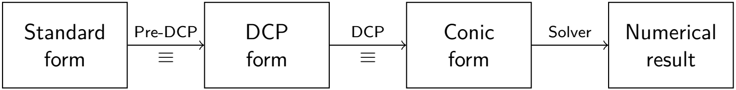
More information about CvxLean

Alexander Bentkamp and Jeremy Avigad started the project and wrote “Verified optimization” (FMM 21).

Bentkamp, Avigad, and I explain the proof-producing DCP algorithm in “Verified reductions for optimization” (TACAS 23).

The code is publicly available at `https://github.com/verified-optimization/CvxLean`.

Conclusion and future steps



Next steps:

1. Improve ergonomics.
2. Documentation and tutorials.
3. Formalize more examples, for instance, from `https://www.cvxpy.org/examples/index.html`.