Verified Convex Optimisation for Hybrid Systems

Ramon Fernández Mir

School of Informatics
University of Edinburgh

September 2022

Initial questions

Why do we care about hybrid systems?
They model safety-critical systems such as self-driving cars.

How can an interactive theorem prover help?
It's a convenient framework to formalise them and reason about their properties.

What is the role of convex optimisation?
Many properties can be *reduced* to convex optimisation queries and solved efficiently.

Convex optimisation

Optimising over $x \in \mathbb{R}^n$ and with convex f_is and affine h_is, a convex optimisation problem in *standard form* is:

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

However, solvers do not understand it! They usually want the problem in *conic form*:

$$\begin{aligned} \text{minimise} \quad & c^T y \\ \text{subject to} \quad & G_i y - h_i \in K_i, \ \ i = 1, \ldots, m; \end{aligned}$$

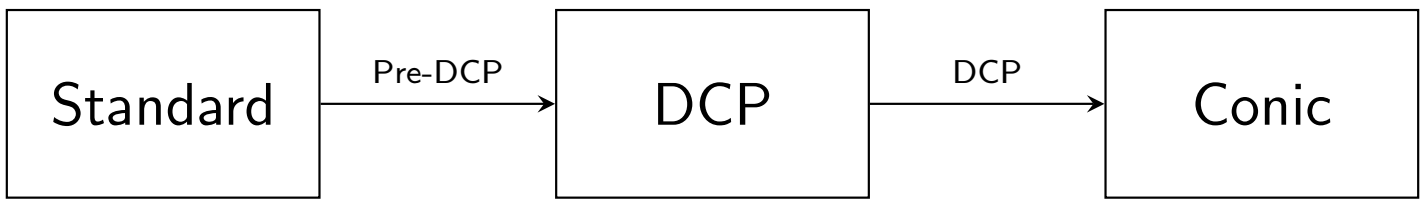
where the K_is are convex cones.

Disciplined convex programming

A system to transform problems to conic form consisting of:

- A library of well-understood base functions called *atoms*.
- Optimisation problems representing each atom called *graph implementations* e.g. $|x| \mapsto \min\{t \mid x \leq t \wedge -x \leq t\}$.
- Infrastructure to keep track of curvature, monotonicity, etc.

The current set-up:



- Pre-DCP transformations are done by hand (errors!).
- DCP transformations are done using a modelling framework such as CVXPY, CVX, Convex.jl, etc.

Problem description

More issues:

- No correctness assurance when adding new atoms.
- Solvers cannot be trusted: interior-point methods are inherently approximate, some problems are ill-conditioned, etc.

We would like a tool with the following requirements:

- High degree of **trust** in the solution.
- Potential of **extending** it without breaking it.
- Access to a large **formal mathematics** library.

Solution: `CvxLean`[1]. Essentially, CVXPY written in Lean.

[1] Joint work with Alexander Bentkamp and Jeremy Avigad.

CvxLean overview

```
noncomputable def prob (a : ℝ) :=
  optimization (x y : ℝ)
    maximize (log x) + (log y)
    subject to
      c1 : 0 < x
      c2 : 0 < y
      c3 : x + y ≤ a

#check prob 3 -- Minimization (Real × Real) Real
```

We use Lean 4's macros system to define our own language for optimisation problems.

Note: the constraints c1 and c2 will be needed to convince Lean that we can solve it.

CvxLean overview

```
noncomputable def prob (a : ℝ) :=
  optimization (x y : ℝ)
    maximize (log x) + (log y)
    subject to
      c1 : 0 < x
      c2 : 0 < y
      c3 : x + y ≤ a

noncomputable def sol : Solution (prob 3) := by
  apply Solution.mk (point := ⟨1.5, 1.5⟩)
  case feasibility => sorry
  case optimality => sorry
```

We can solve the problem by providing a point, and proofs of feasibility and optimality.

CvxLean overview

```
noncomputable def prob (a : ℝ) :=
  optimization (x y : ℝ)
    maximize (log x) + (log y)
    subject to
      c1 : 0 < x
      c2 : 0 < y
      c3 : x + y ≤ a

noncomputable def sol : Solution (prob 3) := by dcp
-- ⊢ Solution (
--  optimization (x y t.0 t.1 : Real)
--      maximize t.0 + t.1
--      subject to
--        _ : posOrthCone (3 - (x + y))
--        _ : expCone t.0 1 x
--        _ : expCone t.1 1 y
--  )
```

CvxLean overview

```
noncomputable def prob (a : ℝ) :=
  optimization (x y : ℝ)
    maximize (log x) + (log y)
    subject to
      c1 : 0 < x
      c2 : 0 < y
      c3 : x + y ≤ a

solve prob 3

#print prob.reduced -- Same as before
#eval prob.status -- "PRIMAL_AND_DUAL_FEASIBLE"
#eval prob.solution -- (1.500000, 1.500000)
```

The `solve` command reduces the problem, sends it to MOSEK, and reconstructs the point in the original domain (no proofs, yet).

Pre-DCP reductions

CvxLean

```
noncomputable def prob2 :
  optimization (x y : ℝ)
    maximize x + y
    subject to
      h : (exp x) * (exp y) ≤ 10

solve prob2
```

CVXPY

```
x, y = cp.Variable(1), cp.Variable(1)
objFun = cp.Maximize(x + y)
constr = [cp.exp(x) * cp.exp(y) <= 10]

prob2 = cp.Problem(objFun, constr)
prob2.solve(verbose=True)
```

Both fail because the constraint is not DCP...

But, clearly, we just need to note that $e^x e^y = e^{x+y}$.

Pre-DCP reductions

CvxLean

```
reduction red/prob2 :
  optimization (x y : ℝ)
    maximize x + y
    subject to
      h : (exp x) * (exp y) ≤ 10 := by
    conv_constr =>
      rw [←Real.exp_add]
    reduction

solve prob2
-- (2.302585, 0.000000)
```

CVXPY

```
x, y = cp.Variable(1), cp.Variable(1)
objFun = cp.Maximize(x + y)
constr = [cp.exp(x + y) <= 10]

prob2 = cp.Problem(objFun, constr)
prob2.solve(verbose=True)
-- [0.] [2.30258509]
```

With the `reduce` command, you can use lemmas from mathlib!

Note: we could even try to automate it.

More complex atoms

A symmetric real-valued matrix M is positive semidefinite (PSD) if for all v, we have $v^T M v \geq 0$ and we write $M \succeq 0$. The cone of PSD matrices is convex.

Provided that $A \succeq 0$, the implementation of the $log(det(A))$ is the following optimisation problem over $t \in \mathbb{R}^n$ and $Y \in \mathbb{R}^n \times \mathbb{R}^n$:

$$\begin{aligned} \text{maximise} \quad & \sum_i t_i \\ \text{subject to} \quad & (t, 1, y) \in \mathcal{E} \\ & \begin{pmatrix} D & Z \\ Z^T & A \end{pmatrix} \succeq 0. \end{aligned}$$

Proving the correctness of this atom requires some extra work such as the existence of an LDL decomosition and the theory of Schur complements.

Proofs of ϵ-feasibility and ϵ-optimality

In general, we cannot prove exact feasiblity and optimality from the result given by the solver.

But we can use it as starting point to give rigorous bounds. This requires:

- Some results about floating point errors.
- Developing a theory of convex ϵ-duality.
- Proof generation infrastructure.

Remark: In the case of semidefinite programming and *sum-of-squares*, there are a few tools that can generate proofs or exact certificates (e.g. ValidSDP and RealCertify).

Sum-of-squares certificates

Write $p = x^{2d} + p_{2d-1}x^{2d-1} + \cdots + p_1 x + p_0$. If we solve:

$$\begin{array}{ll} \text{find} & Q \\ \text{subject to} & p_k = \displaystyle\sum_{i+j=k} Q_{ij} \\ & Q \succeq 0, \end{array}$$

we can conclude that p is nonnegative. Note that the affine constrains are set up so that $p = [x]_d^T Q [x]_d$ where $[x]_d$ are the monomials of degree $\leq d$.

Encoding stability

We say that an autonomous system $\dot{x} = f(x(t))$ is *Lyapunov stable* if for every equilibrium point x_e, i.e. $f(x_e) = 0$, we have that for all $\epsilon > 0$ there exists $\delta > 0$ such that $\|x(0) - x_e\| < \delta$ implies that $\|x(t) - x_e\| < \epsilon$ for all $t \geq 0$.

It suffices to find a function $V : \mathbb{R}^n \to \mathbb{R}$ such that if, without loss, $x_e = 0$ is an equilibrium point, then:

- $V(x) \geq 0$ and $V(x) = 0$ if, and only if, $x = 0$.
- $\nabla V \cdot f(x) \leq 0$ for all $x \neq 0$.

This fits nicely in `CvxLean`.

For instance, if f is polynomial, it can be handled as a sum-of-squares problem.

Encoding safety

One way to certify safety of $\dot{x} = f(x(t))$ defined on some domain X is to come up with a *barrier certificate*.

Suppose $I \subset X$ is the set of initial states and $U \subset X$ a set of unsafe states. A barrier certificate is a differentiable function $B : X \to \mathbb{R}$ satisfying:

- $B(x) \leq 0$ for all $x \in I$.
- $B(x) > 0$ for all $x \in U$.
- $\nabla B(x) \cdot f(x) < 0$ for all $x \in X$.

Again, this fits nicely in `CvxLean`, although for either case one might need some pre-pre-DCP transformations...

Concluding remarks

Some challenges:

- Going back and forth between reals and floats.
- Working with mathport.

Next steps:

1. Package the framework and make it public (soon).
2. Work on automation to solve side conditions.
3. Generate proofs and rigorous bounds.
4. Encode interesting examples.

Thank you