Verified Semidefinite Programming

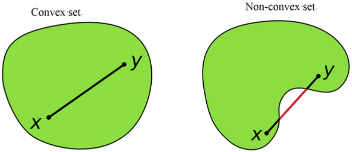
Ramon Fernández Mir

University of Edinburgh

ramon.fernandezmir@ed.ac.uk

October 2021

What is convex optimisation?



Convex function

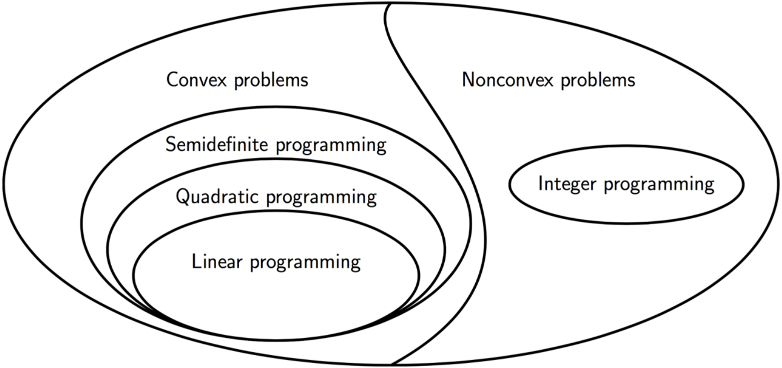
We say that $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for $\theta \in [0,1]$ and $x, y \in \mathbb{R}^n$ we have that $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$.

Convex optimisation problem

Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ and $g_i : \mathbb{R}^n \to \mathbb{R}$ be convex functions and $h_i : \mathbb{R}^n \to \mathbb{R}$ affine functions. A convex optimisation problem is:

$$\begin{aligned} &\text{minimise } f(x) \\ &\text{subject to } g_i(x) \leq 0 \text{ and } h_i(x) = 0. \end{aligned}$$

What is convex optimisation?



Applications: control syntehsis, electronic circuit design, signal processing, finance, etc.

Semidefinite programming

Positive semidefinite matrix

A matrix $M \in \mathbb{R}^{n\times n}$ is positive semidefinite if for all $x \in \mathbb{R}^n$ we have that $x^T Mx \geq 0$. We write $X \succeq 0$. Equivalently, it has a *Cholesky decomposition* $X = L^T L$.

Semidefinite program

A semidefinite program has the form:

$$\begin{array}{ll} \text{minimise} & Tr(C^T X) \\ \text{subject to} & Tr(A_i^T X) = b_i, \;\; i = 1, \ldots, k \\ & X \succeq 0, \end{array}$$

where $b_1, \ldots, b_k \in^n$ and $C, A_1, \ldots, A_k \in \mathbb{S}^n$, i.e. they are real symmetric matrices.

Semidefinite programming

How are they solved? There are several ways but interior point methods are widely used. The idea is roughly the following:

1. Consider the dual problem.
2. By strong duality, the primal and dual problems attain the same value.
3. We follow the so-called central path in the direction here the distance between the primal and dual problem decreases.
4. We specify a tolerance and when the values of the two problems are close enough, return a solution.

Semidefinite programming

Checking whether a polynomial $p = x^{2d} + p_{2d-1}x^{2d-1} + \cdots + p_1 x + p_0$ is nonnegative is an NP-hard problem. However, checking whether it is a sum of squares can be solved efficiently by encoding it as a SDP. If we solve:

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

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

Semidefinite programming

Issues:

1. Many steps skipped when encoding a problem into a SDP.
2. Result is only approximate, how can we make sure it actually solves our problem?

Semidefinite programming

Solution to both issues: Use a theorem prover!

Solution to issue 1: Formalise the problems and the allowed translations.

Solution to issue 2: Try to find the Cholesky decomposition of the result matrix. Two ways:

- Brute-force search of rational nearby solutions.
- Use knowledge about how rounding errors are introduced by the Cholesky factorisation algorithm.

Cholesky decompositon

If we're lucky, we only have that $A \simeq L^T L$, which is not good enough for a theorem prover. What do we do??

- Apply Cholesky on $A' = A - \alpha I$ and obtain L.
- Let $E = A' - L^T L$ and check that $E + \alpha I$ is diagonally dominant (which implies positive definiteness).
- We have that $A = L^T L + (E + \alpha I)$ and the sum of two PSD matrices is PSD so we are done.

All we need to do is find the appropriate α, which is possible if A is strictly positive definite and we work with arbitrary precision.

The Lean theorem prover

Brief history:

- The project began in 2013 in Microsoft Research led by Leonardo de Moura.
- Major refactor in 2017. Lean 3 and mathlib released.[1]
- Another major refactor in 2021. Lean 4 released.
- Lots of interesting maths formalised: schemes, perfectoid spaces, liquid tensors, etc.

[1] https://leanprover-community.github.io/mathlib_stats.html

The Lean theorem prover

Notable features:

- Based on a powerful dependent type theory.
- Small trusted kernel written in C++ (most of Lean is written in Lean).
- Supports constructive reasoning, quotients (natively) and classical reasoning.
- Powerful metaprogramming framework.
- (L4) Hygienic macros system.
- (L4) Built for extensibility.
- (L4) Efficient code generation.
- (L4) Tabled typeclass resolution.

The Lean theorem prover

The Lean mathematical library:

- Smaller than the standard libraries of other systems but exponentially growing.
- We have smooth manifolds, p-adics, lots of category theory, set theory, main results in linear algebra and analysis, etc.
- Backward compatibility issues are being solved by tools like mathport, which allows to use Lean 3 objects in Lean 4.
- A $20 million donation was recently announced to create the Hoskinson Centre for Formal Mathematics, which will focus largely on extending mathlib.

The Lean theorem prover

```
/-- A Lie group is a group and a smooth manifold at the same time in which
the multiplication and inverse operations are smooth. -/
-- See note [Design choices about smooth algebraic structures]
@[ancestor has_smooth_mul, to_additive]
class lie_group {𝕜 : Type*} [nondiscrete_normed_field 𝕜]
  {H : Type*} [topological_space H]
  {E : Type*} [normed_group E] [normed_space 𝕜 E] (I : model_with_corners 𝕜 E H)
  (G : Type*) [group G] [topological_space G] [charted_space H G]
  extends has_smooth_mul I G : Prop :=
(smooth_inv : smooth I I (λ a:G, a⁻¹))
```

```
/-- The unit circle in `ℂ` is a Lie group. -/
instance : lie_group (𝓡 1) circle :=
{ smooth_mul := begin
    let c : circle → ℂ := coe,
    have h₁ : times_cont_mdiff _ _ _ (prod.map c c) :=
      times_cont_mdiff_coe_sphere.prod_map times_cont_mdiff_coe_sphere,
    have h₂ : times_cont_mdiff (𝓘(ℝ, ℂ).prod 𝓘(ℝ, ℂ)) 𝓘(ℝ, ℂ) ∞ (λ (z : ℂ × ℂ), z.fst * z.snd),
    { rw times_cont_mdiff_iff,
      exact (continuous_mul, λ x y, (times_cont_diff_mul.restrict_scalars ℝ).times_cont_diff_on) },
    exact (h₂.comp h₁).cod_restrict_sphere _,
  end,
  smooth_inv := (complex.conj_cle.times_cont_diff.times_cont_mdiff.comp
    times_cont_mdiff_coe_sphere).cod_restrict_sphere _,
  .. metric.sphere.smooth_manifold_with_corners }
```

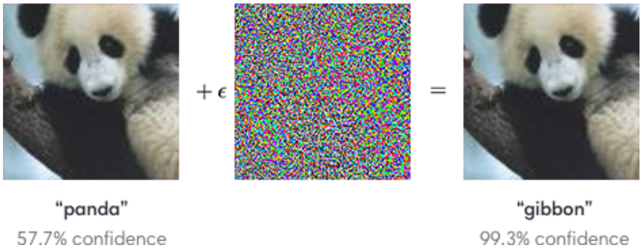
LeanSDP

Goals of the project:

- Link Lean with a convex optimiser.
- Formalise the theory of convex optimisation focusing on problem transformations.
- Check in Lean that the output satisfies the constraints.
- Use this framework to verify real-world systems.

Neural Network Verification

Consider a trained deep and feed-forward neural network used for classification. The network computes a function $f : \mathbb{R}^n \to \mathbb{R}^m$. We want to certify adversarial robustness. The network is δ-locally-robust at $x \in \mathbb{R}^n$ if for any $y \in \mathbb{R}^n$ with $\|x - y\| < \delta$ we have that $\|f(x) - f(y)\| < \epsilon$ for some small ϵ.



Neural Network Verification

This can be stated as an optimisation problem!

$$\begin{aligned} \text{maximise} \quad & \|f(x) - f(y)\| \\ \text{subject to} \quad & x^i = \mathrm{ReLU}(W^{i-1}x^{i-1}) \\ & \|x - y\| < \delta \end{aligned}$$

Solving it in this form is computationally expensive. The next step is to relax this problem to a semidefinite program that we can solve efficiently. The key observation is that a ReLU $z = \max(x, 0)$ can be expressed as the quadratically constrained quadratic program

$$z(z - x) = 0 \wedge z \geq x \wedge z \geq 0.$$

Thank you