CVXPY: A Rewriting System for Convex Optimization

Akshay Agrawal, Steven Diamond, Stephen Boyd

Stanford University

February 23, 2021
Convex optimization
Optimization problem

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad g_i(x) = 0, \quad i = 1, \ldots, p
\end{align*}
\]

- $x \in \mathbb{R}^n$ is (vector) variable to be chosen
- $f_0$ is the \textit{objective function}, to be minimized
- $f_1, \ldots, f_m$ are the \textit{inequality constraint functions}
- $g_1, \ldots, g_p$ are the \textit{equality constraint functions}

**Goal:** find a value for $x$ that minimizes $f_0(x)$, while satisfying constraints
Convex optimization problem

\[\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad Ax = 0
\end{align*}\]

- equality constraint functions are affine
- \(f_0, \ldots, f_m\) are convex: for \(\theta \in [0, 1]\),

\[f_i(\theta x + (1 - \theta)y) \leq \theta f_i(x) + (1 - \theta)f_i(y)\]

i.e., \(f_i\) curve upward
Why convex optimization?

- solution algorithms that work well, in theory and practice
- **many applications** in
  - machine learning, statistics
  - control
  - signal, image processing
  - networking
  - engineering design
  - finance

...and many more
How do you solve a convex optimization problem?

use someone else’s (‘standard’) solver
▶ your problem \textit{must} be written in a standard form
▶ analogous to writing machine code

write your own (custom) solver
▶ lots of work, but can take advantage of special structure

\textbf{this talk: use a domain-specific language}
▶ transforms user-friendly format into solver-friendly standard form
▶ extends reach of problems solvable by standard solvers
Domain-specific languages
Domain-specific languages (DSLs)

- DSLs make it easy to specify and solve convex problems
- Grammar and semantics based on a rule from convex analysis [GBY06]
- Examples: CVXPY, CVXR, Convex.jl, CVX
Example

CVXPY is a Python-embedded DSL [DB16; AvD+18]

```python
import cvxpy as cp

x = cp.Variable()
y = cp.Variable()

objective = cp.Minimize(cp.maximum(x + y + 2, -x - y))
constraints = [x <= 0, y == -0.5]

problem = cp.Problem(objective, constraints)
assert problem.is_dcp()
optimal_value = problem.solve()
```
Example

▶ First, CVXPY **analyzes** the problem and checks that it's valid (convex)
▶ Next, CVXPY **reduces** the problem into a low-level form
  e.g., problem is equivalent to a linear program, with form

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Gx \leq h \\
& \quad Ax = b,
\end{align*}
\]

\[
G = \begin{bmatrix} 1 & 1 & -1 \\ -1 & -1 & -1 \\ 1 & 0 & 0 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad h = \begin{bmatrix} -2 \\ 0 \\ 0 \end{bmatrix}, \quad b = -0.5
\]

▶ Finally, CVXPY solves the problem via a numerical solver
Analysis
Grammar

- CVXPY’s grammar consists of atomic functions (atoms) and a rule for combining them.
- Atoms have known curvature (convex, concave, affine) and monotonicity (increasing, decreasing) (log, exp, square, sum, ...).
- Rule guarantees that compositions of atoms have known curvature.
- Grammar is called disciplined convex programming.

Composition rule: $h(f_1(x), \ldots, f_k(x))$ is convex when $h$ is convex and for each $i$,
- $h$ is increasing in argument $i$, and $f_i$ is convex, or
- $h$ is decreasing in argument $i$, and $f_i$ is concave, or
- $f_i$ is affine.
Analysis

- A problem object is represented as a collection of expression trees
- Nodes are atoms; leaves are variables and numeric constants
- Each tree represents a composition of atoms
- CVXPY checks whether a problem is DCP by recursively checking the composition rule for each tree
Analysis

- Convex optimization problems can be organized into a hierarchy of classes
- Different solvers support different classes
- Each supported class has its own grammar (typically a subset of DCP)
- By default, CVXPY targets the most specific class possible
Other grammars

- CVXPY is easily extended to support grammars other than DCP
- *e.g.*, DGP [ADB19] for geometric programs, DQCP [AB20] for quasiconvex programs, ...  
- Grammar-checking implementation uncoupled from atom implementation  
- Decision to add new grammar based on various factors  
  - usefulness
  - implementation ease
  - maintainability
  - taste / aesthetics
Reductions
Reductions

- CVXPY transforms the original problem via a sequence of reductions
- A reduction converts a problem into a different but equivalent problem
- Each reduction has three methods:
  - accepts, which defines the class of problems the reduction can accept
  - reduce, which takes a problem and reduces it to another
  - retrieve, which retrieves a solution to the original problem from a solution to the emitted problem
Reductions

Some examples:
- FlipObjective
- Complex2Real
- Dcp2Cone
- Dgp2Dcp
- Dqcp2Dcp
- EliminatePwl
- Qp2SymbolicQp
- MatrixStuffing, ConeMatrixStuffing, QpMatrixStuffing
- change of variables, presolves, and more...
A simple example: the FlipObjective reduction.

Accepts:

\[
\begin{align*}
\text{maximize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad Ax = 0
\end{align*}
\]

Reduces to:

\[
\begin{align*}
\text{minimize} & \quad -f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad Ax = 0
\end{align*}
\]

Retrieval: a no-op (solutions are identical)
Standard form

- A chain of reductions ends with a targeted standard form
- The modern standard form is the *cone program*
  
  \[
  \begin{align*}
  \text{minimize} & \quad c^T x \\
  \text{subject to} & \quad Ax = b, \quad x \in K
  \end{align*}
  \]

  where $K$ is a Cartesian product of convex cones
- Special cases include linear programs, semidefinite programs
- There are several solvers for cone programs (SCS, ECOS, MOSEK, ... )
Other standard forms

Reduction system makes it easy to add other problem classes

▶ quadratic programs:

\[
\text{minimize } \frac{1}{2} x^T P x + q^T x \\
\text{subject to } l \leq A x \leq u
\]

▶ linearly constrained least squares:

\[
\text{minimize } \| A x - b \|_2^2 \\
\text{subject to } F x = g
\]

▶ nonlinear programs:

\[
\text{minimize } f(x)
\]

where \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is differentiable
Reduction chains

Reductions are chained together to target a standard form

- Qp2SybmolicQp $\rightarrow$ QpMatrixStuffing $\rightarrow$ OSQP
- Complex2Real $\rightarrow$ Dcp2Cone $\rightarrow$ ConeMatrixStuffing $\rightarrow$ ECOS
- Dgp2Dcp $\rightarrow$ Dcp2Cone $\rightarrow$ ConeMatrixStuffing $\rightarrow$ SCS

CVXPY builds these reduction chains automatically, behind-the-scenes
Recent extensions
Parametrized programs

\[
\begin{align*}
\text{minimize} & \quad f_0(x; \theta) \\
\text{subject to} & \quad f_i(x; \theta) \leq 0, \quad i = 1, \ldots, p \\
& \quad A(\theta)x = b(\theta)
\end{align*}
\]

- Objective function and constraints often depend on some numerical parameters \( \theta \)
- With some mild assumptions, the mapping from \( \theta \) to problem data of the final reduced-to problem is \textit{affine}
- We can represent CVXPY’s rewriting by multiplication with a sparse matrix
- This fact enables two new features: efficiently differentiating through convex optimization problems, and code generation
Differentiating through CVXPY

Solution map of a parametrized CVXPY problem: $x^*(\theta) = (R \circ S \circ C)(\theta)$

- Problem is *canonicalized* (C) to a standard form
- The canonicalized problem is *solved* (S)
- A solution for the original problem is *retrieved* (R)
- We can efficiently differentiate through $C$, $S$, and $R$ [AAB+19]
Exporting to PyTorch and TensorFlow

cvxpylayers: an open-source library for exporting CVXPY problems to PyTorch and TensorFlow

\[ x^*(\theta) = \operatorname{argmin}_x f(x; \theta) \]
subject to \[ g(x; \theta) \leq 0 \]
\[ h(x; \theta) = 0 \]

https://github.com/cvxgrp/cvxpylayers
Tuning a Markowitz policy

Figure: left: untuned; right: policy with tuned constraints, mean and covariance
Tracking a vehicle trajectory

**Figure:** left: untrained path; middle: trained path; right: expected cost histogram.
Code generation

- Extracts sparse matrices representing the rewriting
- Generates C code that takes parameters, calls solver, and returns solution
- Lets you deploy solvers in real-time applications
- Coming soon . . .
Summary

CVXPY is a modular rewriting system for convex optimization that makes convex optimization more accessible to researchers and engineers alike by abstracting away low-level numerical solvers.

- Simple grammar lets users specify problems that are verifiably convex
- Analysis phase matches a high-level problem with a low-level problem class
- Reduction system makes it easy to add new grammars and solvers
- Rewriting is amenable to optimizations when parameters are used

https://cvxpy.org
References


