CVXCanon: Automatic Canonicalization of Disciplined Convex Programs

John Miller, Paul Quigley, Jack Zhu
EE364b: Convex Optimization II Class Project

Introduction
Convex optimization modeling tools like CVX, CVXPY, and Convex.Jl all translate high-level problem descriptions into low-level, canonical forms that are then passed to a backend solver. We introduce CVXCanon, a software package that factors out the common operations that all such modeling systems perform into a single library with a simple C interface. CVXCanon is currently interfaced with CVXPY and provides significant performance gains for this system.

Problem Description
The abstract syntax trees representing the problem must be converted into a canonical representation for input to solvers. CVX libraries allow users to specify their convex optimization problem at a high level. From there, these libraries convert the problem to an abstract syntax tree, and then to linear atom tree. CVXCanon targets the next step, conversion from the linear atom tree to second-order cone problems. For many CVXPY problems, this step takes up the majority of the solve time.

Canonicalization
The abstract syntax trees representing the problem must be converted into a canonical representation for input to solvers. Consider the trivial minimization problem with variable \( x, y \in \mathbb{R}^2 \)

\[
\begin{align*}
\text{minimize} & \quad \|x + y\|_1 \\
\text{subject to} & \quad t_0 - t_1 = 0 \\
& \quad \begin{bmatrix} t_1 & 1 \end{bmatrix}^T \begin{bmatrix} t_2 & 2 \end{bmatrix} \\
& \quad t_2 = x + y,
\end{align*}
\]

where \( Q^2 = \{(i, x) \in \mathbb{R} \times \mathbb{R} : |i|x| \leq 1\} \).

The problem can be equivalently expressed using an expanded abstract syntax tree with only linear atoms.

Matrix Stuffing Algorithm

def build_matrix(given a list linear atom \( L \) and a map from variables to column numbers, \( M \)
\begin{align*}
\text{let} e & = 0 \\
\text{for} i \text{ in range}(n): \\
\text{e} & = e + x \\
p & = \text{Problem(Minimize(e + 1, 2)), [x >= 0])}
\end{align*}
\]

CVXPY returned the problem data in 6.81s, while CVXCanon returned the problem data in .772 ms.

Further Work
- The Julia wrapper for CVXCanon simply calls the Python wrapper, and the Convex.Jl code that produces standard form matrices, the conc_form function, must be modified slightly to take full advantage of CVXCanon. We are currently working to correct both of these issues.
- CVXPY currently passes the returned problem data through another wrapper to call a solver. We could obtain performance improvements and boost code reusability and robustness by directly calling a solver from CVXCanon after obtaining the problem matrix.

Acknowledgments
We thank Steven Diamond for the project idea and his continued support and guidance.