CVX.jl: A Convex Modeling Environment in Julia
Jenny Hong, Karanveer Mohan, and David Zeng
EE364b: Convex Optimization II Final Project

Introduction
CVX.jl is a modeling environment for disciplined convex programming (DCP) in Julia, an open-source scientific computing language. CVX.jl parses human-readable, DCP-compliant expressions into standard forms for backend solvers. CVX.jl currently supports linear programs and second-order cone (SOC) programs with scalar, vector, or matrix variables, using the Embedded Conic Solver (ECOS) for its backend. Supported operations, also known as atoms, include:

- Arithmetic atoms: +, -, *, /
- Slicing and shaping atoms: getindex, hcat, vertcat, reshape, diag, vec, transpose
- Positive orthant atoms: abs, max, min, pos, neg, norm_inf, norm_one
- SOC atoms: sqrt, geo_mean, quad_over_lin, inv_pos, sum_squares, square_pos

Constraints and Canonicalization
Each user-declared constraint is of type CvxConstr, which holds an operator (i.e., <=, ==, or >=), a left-hand-side expression, and a right-hand-side expression. The constraints and objective of the problem must then be formulated into a canonical form for the backend solver. The ECOS backend expects the following canonical form:

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

where \( K = \mathbb{R}^l \times Q_{n_1} \times \cdots \times Q_{n_p} \) and \( Q_k = \{(t,x) \in \mathbb{R} \times \mathbb{R}^{p_k} \mid \|x\|_2 \leq t\} \) is the second-order cone of dimension \( p \). A constraint such as

\[
\text{norm}_2(k * z + b) \leq 3
\]

with variable \( z \in \mathbb{R}^p \) and data \( A \in \mathbb{R}^{m \times n} \), \( b \in \mathbb{R}^m \) will be parsed into expressions with the following relationships:

\[
\begin{align*}
t_0 & \leq 3 \\
t_0 = & \|[t_1]\|_2 \\
t_1 = & t_2 + b \\
t_2 = & Ax \\
t_3 = & 3
\end{align*}
\]

where unique identifiers \( t_0, t_1, t_2 \) have been introduced to represent each expression. These relationships between the expressions implicitly introduce additional constraints (2), (3), and (4) to the problem. These constraints can then be collected together to build the affine equality constraints \( Az = b \) and the conic inequality constraints \( Gx \preceq_h \) in the following manner:

\[
\begin{bmatrix}
0 - I_m & I_m & 0 \\
0 & 0 & -I_m \\
-1 & 0 & 0 \\
0 & -I_m & 0
\end{bmatrix}
\begin{bmatrix}
t_0 \\
t_1 \\
t_2 \\
t_3
\end{bmatrix}
\preceq
\begin{bmatrix}
0 \\
0 \\
1 \\
0
\end{bmatrix}
\]

These implicit constraints that arise during canonicalization are stored in a CanonicalConstr instance in every expression and constraint. Each expression and constraint stores both its own implicit canonicalization constraints and those of all its descendants.

Expression Types
All expressions in CVX.jl are instances of one of three types:

- Constant: Wrappers around constant-valued scalars, vectors, or matrices.
- Variable: User-declared variables of the optimization problem.
- CvxExpr: General expressions that depend on other expressions, variables, or constants.

Each expression type keeps track of its convexity, sign, size, and canonical form. An atom, such as an arithmetic operator or a function like norm_inf, combines expression arguments into a new expression, resulting in a tree structure for the expressions. Upon solving the problem, we can retrieve optimal values for Variable and CvxExpr types.

Using CVX.jl
We introduce the basic functionality of CVX.jl with example code.

- Declaring variables:
  To declare variables \( x \in \mathbb{R}, y \in \mathbb{R}^4, Z \in \mathbb{R}^{4 \times 6} \), we use the following syntax.
  \[
x = \text{Variable}() \\
y = \text{Variable}(4) \\
Z = \text{Variable}(4, 6)
\]

- Forming expressions:
  We can form expressions from CVX.jl variables and expressions, and regular Julia constants.
  \[
  \text{expr1} = \text{sum_squares}(y + 2) \\
  \text{expr2} = 2*[4] + x
\]

- Forming constraints:
  \[
  \text{const1} = 10 * \text{expr1} <= \text{sum}(x) + \text{sum}(\text{reshape}(Z, 2, 3, 8)) \\
  \text{const2} = \text{expr2} >= 3
\]

- Creating problems:
  Constraints can be added at construction or appended later.
  \[
  \text{problem} = \text{minimize} (\text{norm}_2(y), [\text{const1}, \text{const2}]) \\
  \text{problem.constraints} += (y >= 0)
\]

- Solving problems:
  \[
  \text{solve!(problem)}
  \]

- Extracting optimal values (of the problem, primal variables, user-defined expressions, and dual variables):
  \[
  \text{println} (\text{problem.optval}) \\
  \text{println}(\text{problem.constraints}) \\
  \text{println} (\text{expr1.evaluate()})
  \]

Further Work
- We have also written SCS.jl, a Julia wrapper around the Splitting Cone Solver (SCS). Using this package, we will be able to support exponential and semidefinite cone constraints, and atoms such as log_sum_exp, log_det, norm_nuclear, kl_divergence, lambda_max, lambda_min.
- We plan to perform reductions to prune our expression tree. For example, for the expression \( 4x + 3x \), we currently form three additional identifiers, one for each of the multiplications and a third for the addition. Instead, we could recognize that an equivalent expression is \( 7x \) and use only one additional identifier to represent the expression.

Acknowledgements
We thank Madeleine Udell for the project idea, for her work on starting CVX.jl, and for her continued support and direction. We also thank Stephen Boyd for his mentorship. Finally, we thank Steven Diamond for his experienced guidance as the author of CVXPY, which has been a very useful model for us.