Alexander Leong
Software Engineer
http://alexanderleong.com

ConicSolve.jl

A modular Primal-Dual Interior-Point Method Conic Optimization Solver

JuliaCon 2025
Pittsburgh, PA

Last updated: 15/02/2025

Introduction

Mathematical optimization is used in many scientific and engineering domains, from robotics and image processing to communications. In robotics, for example, optimization helps compute collision-free trajectories; in image processing (e.g., MRI or Magnetic Resonance Imaging), it aids in denoising through matrix completion; and in network design, it enables maximizing data throughput subject to capacity constraints (e.g., Max Flow Min Cut).

ConicSolve.jl solves constrained optimization problems by applying primal-dual Interior-point Methods (IPMs) to efficiently solve them. The solver handles a variety of problem classes, including Linear Programming (LP), Quadratic Programming (QP), Second Order Cone Programming (SOCP), and Semidefinite Programming (SDP).

Installation

1. Ensure Julia is installed
2. Execute from Julia REPL:

```
using Pkg
Pkg.add("ConicSolve")
using ConicSolve
```

Features

- Solves LP (Linear Programming), QP (Quadratic Programming), SOCP (Second Order Cone Programming) and SDP (Semidefinite Programming) problems
- Supports Nonnegative Orthant, Second Order and Positive Semidefinite cone types
- Minimal project dependencies (mainly LAPACK)
- Implements common array manipulation operations to reduce complexity in defining optimization problems
- Can define Semidefinite and Sum of Squares problems with ease using predefined models
- Modular (by design) so users can bring their own KKT (Karush Kuhn Tucker) solver to utilize sparsity patterns and problem structure for optimal solver performance
- Open source (MIT License)
- Several scientific and engineering examples ready to be extended for real world application
- Native conic solver interface, reduce overhead associated with using dependencies when defining an optimization problem

Future work

- Add support for CUDA based GPUs
- Add examples of SDP applications for Quantum Information Science
- Add more example applications
- Add additional mathematical functions to realize performance gains from exploiting problem structure and sparsity
- Investigate extensions to Primal Dual Interior-point optimization methods
- Investigate supporting more algebraic modelling frontends

Conic Form

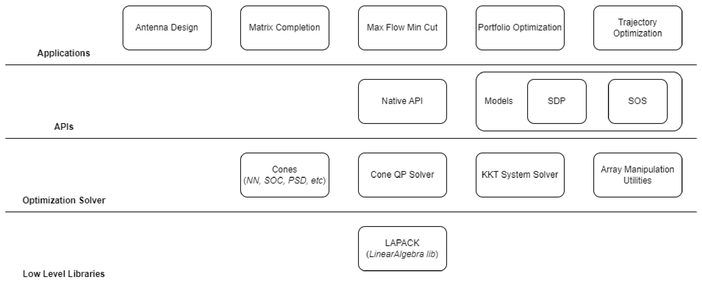
$$
\begin{aligned}
\text{minimize} \quad & (1/2)x^T P x + c^T x \\
\text{subject to} \quad & Gx + s = h \\
& Ax = b \\
& s \succeq 0
\end{aligned}
$$

x: is the decision variable
P, c: defines the objective in terms of x
A, b defines the linear equality constraints
G, h defines the linear inequality constraints
s defines x together with G, h with respect to the particular cone type

Many optimization problems can be converted into a conic form. The conic form allows the application of several cone types:

- Nonnegative Orthant
- Second Order Cone
- Positive Semidefinite Cone

Components



The bottom layer shows that the only main dependency is Julia's LinearAlgebra library. The second layer shows the various components that make up the optimization solver. Users can easily define their own cone types, provide a specific KKT solver or use array utilities to help construct an optimization problem. The third layer defines APIs for constructing various types of optimization problems.

The top layer consists of all example applications which extend from either the native or model-based APIs. The rank minimization (matrix completion) example (for example) uses the SDP model and trajectory optimization example uses the SOS model.

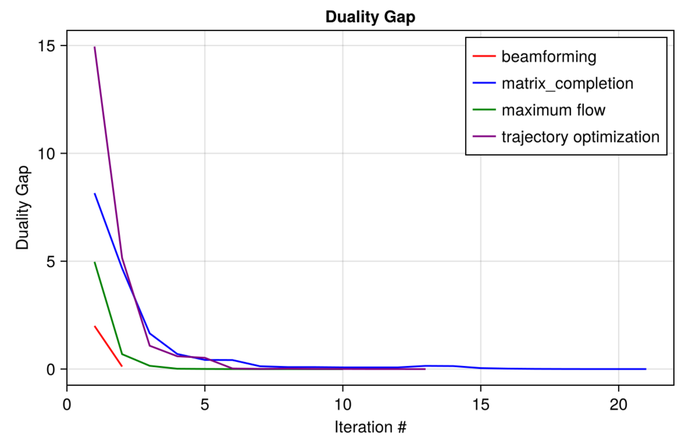
You may need to install further dependencies to run the specific examples. This modular approach allows the optimization solver to be performant, extensible for future emerging applications and versatile so it is suitable for solving large-scale problems or real-time embedded problems for example.

The high-level component diagram (see below) can serve as a guide for users and developers working with the project codebase.

Using the low-level API

```
using ConicSolve
p_1 = 16 # for example, 16 x 16 SDP matrix
p_2 = 136 # for example, 136 elements (or 16 * 17 / 2) in
lower triangular SDP matrix
A = [] # problem specific, replace with your values
G = [] # problem specific, replace with your values
P = [] # problem specific, replace with your values
b = [] # problem specific, replace with your values
c = [] # problem specific, replace with your values
h = [] # problem specific, replace with your values
cones::Vector{Cone} = []
push!(cones, PSDCone(p_1))
push!(cones, NonNegativeOrthant(p_2))
cone_qp = ConeQP(A, G, P, b, c, h, cones)
solver = Solver(cone_qp)
optimize!(solver)
```

Results



Solver parameters: η=σ, γ=1
The above plot shows rapid convergence of the duality gap with just a few iterations for many common mathematical optimization problems.

See project documentation for specific details on problem formulation:
https://github.com/alexander-leong/ConicSolve.jl

References

Ben-Tal, N. (2001). Lectures on Modern Convex Optimization.
Blekherman, P. T. (2012). Semidefinite Optimization and Convex Algebraic Geometry.
Boyd, V. (2009). Convex Optimization.
Vandenberghe. (2010). The CVXOPT linear and quadratic cone program solvers.