Project Midterm Report

Zach Rosenof, Mike Sosa, Mitch Perry

April 2017

1 Problem Description

Our objective was to write python code which implements James Renegar's framework for transforming convex conic optimization problems into simpler convex formulations to solve with subgradient methods. This framework could be useful for solving many problems more easily, but as of yet, it has not been implemented in any coding language.

Originally, we hoped to implement the framework for cones in general. Upon further consideration, we decided that was an unreasonable goal. Simply writing code to recognize all of the many kinds of cones and their Cartesian products would be enough work for a project in itself. So we decided to limit our scope to implementing this algorithm for only semi-definite programs.

Renegar's method works as follows. Given a problem of the form,

$$\min\ c \cdot x$$

$$\text{s.t. } x \in \text{Affine}$$
$$x \in K$$

First, we find a distinguished direction e which is in the intersection of the interior of the cone and the set Affine. From this, we define the function $\lambda_{\text{min}}(x)$:

$$\lambda_{\text{min}}(x) := \inf\{\lambda : x - \lambda e \notin K\}$$

Then, we find some z such that $z < c \cdot e$ where $\cdot$ represents the inner product associated with the particular problem. With this z, we define a new affine space: $x \in \text{Affine}_z$ if $x \in \text{Affine}$ and $c \cdot x = z$. Now, we write the problem:

$$\max\ \lambda_{\text{min}}(x)$$

$$\text{s.t. } x \in \text{Affine}_z$$

Renegar shows that solving this problem is equivalent to solving the first. He also provides a way to move between a solution of one problem to the solution of the other.

We are concerned with this formulation for semidefinite cones. If the identity matrix is a feasible point, it can be used as the distinguished direction. With the identity matrix as the distinguished direction, Renegar shows that $\lambda_{\min}(X)$ is the minimum eigenvalue of X.

The feasible region of this program is convex and Lipschitz-continuous, so we can use a supgradient method to solve the problem.

2 Current Progress

So far we've implemented the core functionality for the case where the problem is an SDP and the identity matrix is in the set Affine. Namely, we can

- Check to see if the given input is formatted correctly
- Find an initial feasible solution to the modified problem
- Run a supgradient method on the modified problem and convert the optimal solution to a solution of the original problem.

3 Roadblocks

So far we've encountered two roadbloacks. The first was realizing that implementing this algorithm for general cones was out of the scope of this project. We got around this by restricting ourselves to the positive semidefinite cone, and, more specifically, to the case where the identity matrix is a feasible solution to the problem. The second roadblock we faced was implementing the supgradient method needed to solve the problem. More specifically, we had trouble with projecting matrices (rather than projecting column vectors).

4 Testing Plan

We plan on writing two classes of tests. The first class of tests will ensure that the program is operating correctly. The second will compare the speed of our code to others.

For our correctness tests, we will first check to make sure that the code doesn't behave strangely when we input bizarre parameters. For example, matrices with words in them, or matrices with dimensions that do not line up. Then we will try various problems to make sure that the results are consistent with those found by established solvers. We will then focus on ensuring that it works on corner case problems.

For the speed tests, we will compare the computation time to those of cvxpy. This way, the speed of the coding language itself will not affect the results. We will look at different sizes of problems, both in terms of the number of variables and how restrictive the affine set is. Through this, we will see where our implementation could be most useful.

5 Future Work

The following is an ambitious list of things we'd like to attempt, roughly in order of which we'd like to do first.

5.1 Broader class of SDPs

We hope to extend the method to general SDPs where the identity matrix is not necessarily a valid distinguished direction.

5.2 More interpretations

Currently, we only accept one formulation of SDPs. We assume that the user has reformulated their problem so it's of the form

$$\min C \cdot X$$

$$\text{s.t. } A_i \cdot X = b_i \quad \forall i = 1, ..., m$$

$$X \succeq 0$$

Where C, X, and all A_i are (n x n)-dimensional matrices. We hope to extend our code to interpret multiple ways of writing SDPs and convert them into versions on which we can run our algorithm.

5.3 Optimize

We would like to optimize the code for speed. On our first pass, we were more concerned with everything working as intended than making it run as quickly as possible. This could be done in a number of ways. We could rewrite time-consuming pieces of code with speed in mind, we could look to find a better starting feasible solution, and we could also consider how our choice of subgradient at each iteration affects the speed of the method.

5.4 Other Cones

We think it could be beneficial to implement Renegar's method for other classes of optimization problems. Renegar gives an analysis of his method for linear programs, so it seems natural to try to add the capability to solve LPs.