LEigOpt: Fast Eigenvalue Optimization for Spectral-Element PDEs

Guillermo Angeris and John Sholar
EE364b: Convex Optimization II Class Project

Introduction

Eigenvalue optimization is a common task in many control problems, inverse design, and, recently, in inference problems on continuous spaces. For this project, we attempted a few methods for fast solutions for eigenvalue problems arising from spectral element discretizations of eigenvalue partial differential equations (PDEs).

Spectral Element Methods and Laplacians

A *spectral element method* (SEM) is a finite-element method for solving PDEs which makes use of high-degree polynomials to approximate functions on a compact domain. The idea is to construct a quadrature rule such that all polynomials of lesser degree have exact integrals when sampled at a particular set of points, and generate some basis polynomials of degree less than the maximum.

In general, the Laplacians of these SEM-discretized problems—while sparse—usually incur a large computational cost on modern optimization packages for even a modest number of points. This is because the matrix generated by the spectral elements method is essentially a block-diagonal matrix with small blocks, each of which overlap with the previous block by exactly one element (see Figure 1), so no completely trivial decomposition can be applied.

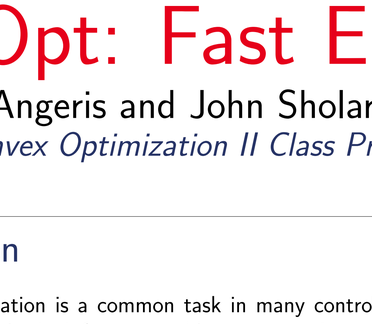


Figure 1: A simple example of the overlapping diagonal elements with three blocks. The darker box indicates a single overlapping element, e.g., $(A_1)_{nn} = (A_2)_{11}$ if $A_1 \in \mathbf{R}^{n\times n}$.

Our project takes advantage of the structure of these approximately-block-diagonal matrices for fast solutions of eigenvalue problems.

Problem

Let $\mathscr{L} : \mathbf{R}^n \to \mathbf{S}^m$ be affine and have the approximately-block-diagonal structure described in figure 1 and let $x \in \mathbf{R}^n$, then the problem of interest can be written as the dual of a standard-form SDP,

$$\begin{array}{ll} \underset{x,\,t}{\text{maximize}} & b^T x + t \\ \text{subject to} & \mathscr{L}(x) \succeq tI, \\ & Cx = d, \\ & x \succeq 0. \end{array} \tag{1}$$

The first inequality is with respect to the semidefinite cone, while the latter is with respect to the positive orthant.

Alternating Projections

We can rewrite (1) as

$$\begin{array}{ll} \underset{x,\,t}{\text{minimize}} & c^T x \\ \text{subject to} & \mathscr{L}(x) = \sum_i E_i Z_i E_i^T, \\ & Cx = d, \\ & x \succeq 0, \\ & Z_i \succeq 0,\ i \in \{1, 2, \ldots, b\}, \end{array}$$

where $E_i \in \mathbf{R}^{n\times b}$ are selector matrices which project $Z_i \in \mathbf{R}^{b\times b}$ into their appropriate position aligned with $\mathscr{L}(x) - tI$ in $\mathbf{R}^{n\times n}$. If $b \ll n$ projecting the Z_i into the PSD cone is potentially much faster than projecting the complete matrix.

In order to construct the projections in question, we derive conditions for optimality as follows. From strong duality we have,

$$b^T x + d^T \eta - \operatorname{tr}(\Lambda A_0) = 0,$$

while, from dual feasibility we have,

$$c_i + \operatorname{tr}(\Lambda A_i) + (C^T \eta)_i = \xi_i \ \ \forall i, \qquad -E_j^T \Lambda E_j = \Sigma_j \ \ \forall j.$$

Primal feasibility implies,

$$\mathscr{L}(x) = \sum_j E_j Z_j E_j^T, \qquad x \succeq 0, \qquad Z_j \succeq 0 \ \ \forall j, \qquad Cx = d,$$

while dual feasibility implies,

$$\Sigma_j \succeq 0 \ \ \forall j, \qquad \xi_i \geq 0 \ \ \forall i.$$

These are all either affine spaces or convex cones, so we can find a point in the intersection of all sets via alternating projections. Since the affine projections can be computed and cached in the pre-solve, and projecting into PSD cones is simple for $Z_j, \Sigma_j \in \mathbf{R}^{b\times b}$. As $b \ll n$, this formulation yields a speed up in computational time.

Optimizations

Many of our projections involve projecting a variable y onto $\{x \mid Cx = d\}$. One standard projection in this case is given by

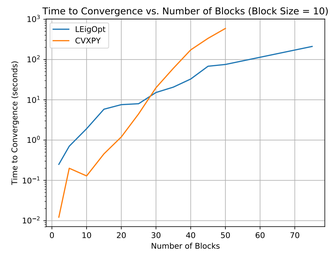
$$\Pi_{\{x|Cx=d\}}(y) = C^\dagger d + (I - C^\dagger C) y$$

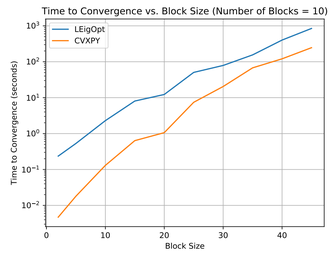
However, in the case that $C \in \mathbf{R}^{k\times n}$, where $y \in \mathbf{R}^n$, this can be computed much more efficiently, specifically by computing the QR factorization of C^T and noting that given $QR = C^T$ we have

$$\begin{aligned} C^\dagger C &= C^T (C C^T)^{-1} C \\ &= QR(R^T Q^T Q R)^{-1} R^T Q^T \\ &= QQ^T \end{aligned}$$

Results

To compare the efficiency of our methods to the standard SCS solver, we test the performance of both on an eigenvalue derivation problem, which can be formulated as a special case of (1).





- Fixing the number of blocks N to be 10 and varying b, SCS outperforms LEigOpt consistently, but the gap narrows from a factor of 100 ($b = 2$) to a factor of 3 ($b = 45$).
- Fixing $b = 10$ and varying N, SCS initially outperforms LEigOpt, but is surpassed at around $N = 30$. As the block count increases, LEigOpt consistently outperforms SCS—more specifically, convergence times for LEigOpt grow roughly linearly, while those for SCS grow roughly quadratically, if not cubically. SCS did not finish at $N = 75$.
- It should also be noted that large portions of our solver are written in Python (interpreted, slow) on top of NumPy libraries, while the CVXPY SCS solver is written in C (compiled, fast).

Future Work: Interior-Point Methods

For interior point methods, rewriting the problem using a barrier method is straightforward:

$$\begin{array}{ll} \underset{x,\,t}{\text{maximize}} & b^T x + t + \mu \log\det(\mathscr{L}(x) - tI) \\ \text{subject to} & Cx = d, \\ & x \succeq 0. \end{array}$$

The $\mathscr{L}(x)$ matrix, with the given sparsity pattern, can be forced to have block-diagonal structure on the top-left block, with narrow bands on the bottom and rights sides, via some permutation matrix P, with

$$\hat{\mathscr{L}}(x) = P \mathscr{L}(x) P^T.$$

This matrix, $\hat{\mathscr{L}}(x)$ has a simple Cholesky decomposition, which can be exploited for fast computation of the barrier function and its first and second derivatives for use with a Newton or Newton-like method.