Differentiating Through a Quadratic Cone Program

Quill Healey* Parth Nobel† Stephen Boyd†

August 28, 2025

Abstract

Quadratic cone programs are rapidly becoming the standard canonical form for convex optimization problems. In this paper we address the question of differentiating the solution map for such problems, generalizing previous work for linear cone programs. We follow a similar path, using the implicit function theorem applied to the optimality conditions for a homogenous primal-dual embedding. Along with our proof of differentiability, we present methods for efficiently evaluating the derivative operator and its adjoint at a vector. Additionally, we present an open-source implementation of these methods, named `diffqcp`, that can execute on CPUs and GPUs. GPU-compatibility is already of consequence as it enables convex optimization solvers to be integrated into neural networks with reduced data movement, but we go a step further demonstrating that `diffqcp`'s performance on GPUs surpasses the performance of its CPU-based counterpart for larger quadratic cone programs.

*healeyquill@gmail.com

†{ptnobel, boyd}@stanford.edu, Department of Electrical Engineering, Stanford University

Contents

1 Introduction

A *quadratic cone program* (QCP) is an optimization problem which minimizes a convex quadratic function over the intersection of a subspace and a convex cone. Quadratic cone programming is the generalization of both quadratic programming and (linear) cone programming, which date to the 1950s [16] and 1990s [23, Chapter 4], respectively. Specifically, a *quadratic program* (QP) is a QCP whose cone is restricted to the product of $\{0\}$, $\mathbf{R}$, and $\mathbf{R}_+$, while a (linear) *cone program* is a QCP restricted to having a linear objective.

Quadratic programs, despite their limited modeling power, have been studied extensively as they arise ubiquitously across many disciplines—from classical engineering contexts to finance. Cone programming, on the other hand, has been studied for its generality—all convex optimization problems can be equivalently written as a cone program. Along with their rich theory, signficant development has gone into specialized solvers for both quadratic programs [34, 8] and cone programs [14, 28]. Moreover, domain specific languages, such as CVXPY [5] and CVXR [18], have been designed to enable easy modeling with both classes of programs [20].

Perturbation and sensitivity analysis has also been thoroughly developed for QPs and cone programs. Clasically, this analysis centered on the Lagrange multipliers [32, 33, 31]. In recent years, *differentiable optimization*—the derivative of the solution map between an optimization problem's parameters and its solution—has been developed. The gradients of the solution map of a quadratic program (with respect to the problem data) were derived in [7] by exploiting the problem structure. Subsequently, [4] proposed a technique for differentiating the solution map of a cone program using a more general approach based on the implicit function theorem. Differentiable optimization has found applications across energy systems [13], statistics [24, 25], control [3, 9], and in neural networks [2, 26].

In recent years, specialized solvers for QCPs have been developed and have demonstrated significant speedups on problems previously solved via cone programs or quadratic programs [27, 19]. As a result, QCPs are emerging as a practical alternative to cone programs for many convex optimization problems. Further, there has been success at GPU-accelerating these QCP solvers [12]. However, the theory of differentiating the solution map of QCPs has remained undeveloped.

1.1 Our contribution

Closely following [4], we derive conditions for when the derivative, and its adjoint, of the primal-dual solution map to a QCP with respect to the QCP's parameters exists. We then present an extension to [4] to evaluate Jacobian-vector and vector-Jacobian products with these derivatives via projection onto cones and sparse linear system solves. We then describe our GPU-accelerated Python implementation of this method in §3, which forms the derivative of the solution map as an abstract linear operator. Additionally, in §A we present a unified reference of cone projection operators and their derivatives. Notably, this reference and our implementation includes the power cone, which has previously been neglected in the differentiable optimization literature.

2 Solution map and its derivative

Following [4], we consider the mapping from the numerical data defining the primal and dual problems of a QCP to its solutions. This *solution map* is in general set-valued, but in neighborhoods where it is single-valued it is an implicit function of the problem data. In the sequel, we present a system of equations that implicitly define the solution map of a QCP when it is single-valued. Applying the implicit function theorem to this system, we obtain regularity conditions on the problem data that guarantee when the solution map is single-valued and its derivative exists. Finally, we provide an expression for the derivative at points where these conditions are satisfied.

2.1 QCPs and implicit functions

The primal and dual problems for a (convex) QCP are

$$\begin{array}{llll} \text{(P)} & \text{minimize} & \frac{1}{2}x^T Px + q^T x \\ & \text{subject to} & Ax + s = b \\ & & s \in \mathcal{K}, \end{array} \qquad \begin{array}{lll} \text{(D)} & \text{maximize} & -\frac{1}{2}x^T Px - b^T y \\ & \text{subject to} & Px + A^T y = -q \\ & & y \in \mathcal{K}^*, \end{array} \qquad (1)$$

where $x \in \mathbf{R}^n$ is the *primal* variable, $y \in \mathbf{R}^m$ is the *dual* variable, and $s \in \mathbf{R}^m$ is the primal *slack* variable. We assume that $\mathcal{K} \subseteq \mathbf{R}^m$ is a nonempty, closed, convex cone with *dual cone* $\mathcal{K}^*$. The *problem data* are $P \in \mathbf{S}^n_+$, $A \in \mathbf{R}^{m \times n}$, $q \in \mathbf{R}^n$, and $b \in \mathbf{R}^m$. (The convex cone can also be problem data, but for our purposes we fix $\mathcal{K}$.) To simplify the subsequent discussion, we define the set

$$\Theta = \left\{\theta = (P, A, q, b) \,\middle|\, (P, A, q, b) \in \mathbf{S}^n \times \mathbf{R}^{m \times n} \times \mathbf{R}^n \times \mathbf{R}^m \right\}.$$

That is, θ is the concatenation of problem data (relaxed to allow $P \not\succeq 0$)—a change from [4] which embeds the problem data into a skew-symmetric matrix.

Optimality conditions. The optimality conditions for (1) are

$$Ax + s = b, \quad Px + A^T y = -q, \quad s \in \mathcal{K}, \quad y \in \mathcal{K}^*, \quad s^T y = 0. \tag{2}$$

Note that $s^T y$ is the duality gap, *i.e.*, at any point that satisfies the first four equalities and inclusions, $s^T y = \hat{p} - \hat{d}$ where $\hat{p}$ is the primal objective at (x, s) and $\hat{d}$ is the dual objective value at (x, y). Also note that (2) is an implicit system that defines the solution map to a QCP when it is single-valued.

Homogenous embedding. By applying $s^T y = \hat{p} - \hat{d}$, a solution that satisfies (2) is equivalent to a solution of the following nonlinear systems of equations

$$\begin{bmatrix} -q \\ s \\ 0 \end{bmatrix} = \begin{bmatrix} P & A^T \\ -A & 0 \\ -q^T & -b^T \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ b \\ -x^T P x \end{bmatrix}, \quad (x, s, y) \in \mathbf{R}^n \times \mathcal{K} \times \mathcal{K}^*. \tag{3}$$

However, because this system is not guaranteed to be feasible (*e.g.*, when the problem is primal or dual infeasible), we instead consider the *homogeneous embedding* (as defined in [19])

$$\begin{gathered} \begin{bmatrix} 0 \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} Px + A^T y + \tau q \\ -Ax + \tau b \\ -(1/\tau) x^T P x - q^T x - b^T y \end{bmatrix}, \\ (x, s, y, \tau, \kappa) \in \mathbf{R}^n \times \mathcal{K} \times \mathcal{K}^* \times \mathbf{R}_+ \times \mathbf{R}_+, \quad \tau + \kappa > 0, \end{gathered} \tag{4}$$

where τ and κ are new real-valued variables. Unlike (3), this embedding is guaranteed to be (asymptotically) feasible even when (1) is primal or dual infeasible.

Applying a change of variable with $N = n + m + 1$, the sets

$$K = \mathbf{R}^n \times \mathcal{K}^* \times \mathbf{R}_+, \quad K^* = \{0\}^n \times \mathcal{K} \times \mathbf{R}_+,$$

the variables,

$$u = (x, y, \tau) \in \mathbf{R}^N, \quad v = (0, s, \kappa) \in \mathbf{R}^N$$

and the functions $Q_1 : \mathbf{R}^N \to \mathbf{R}^n$, $Q_2 : \mathbf{R}^N \to \mathbf{R}^m$, and $Q_3 : \mathbf{R}^N \to \mathbf{R}$ defined as

$$Q_1(u) = Px + A^T y + \tau q, \quad Q_2(u) = -Ax + \tau b, \quad Q_3(u) = -(1/\tau) x^T P x - q^T x - b^T y,$$

we simplify the sequel by writing (4) as

$$Q(u) = v, \quad u \in K, \quad v \in K^*, \quad u_N + v_N > 0. \tag{5}$$

Here $Q : \mathbf{R}^N \to \mathbf{R}^N$ is defined as $Q(u) = (Q_1(u), Q_2(u), Q_3(u))$. Lastly, we define a solution to (5) as a complementary solution if $u_N v_N = 0$.

2.2 Solution map

For given problem data, the corresponding QCP (1) may have no solution, a unique solution, or multiple solutions. For the remainder of this paper, we assume it has a unique solution. We define the solution map $S : \Theta \to \mathbf{R}^{n+2m}$ of a family of parameterized optimization problems as the function mapping θ to vectors (x, y, s) that satisfy (2). Similar to [4], we express this function as composition of functions. Unlike in [4], we only have two functions in our composition: $S = \phi \circ s$, where

- $s : \Theta \to \mathbf{R}^N$ maps the problem data to a complementary solution of the homogeneous embedding and
- $\phi : \mathbf{R}^N \to \mathbf{R}^{n+2m}$ maps a complementary solution to a solution of the primal-dual pair.

At a point θ where S is differentiable, the derivative of the solution map is

$$DS(\theta) = D\phi\left(s(\theta)\right) Ds(\theta),$$

by the chain rule. In the remainder of this section we develop an expression for $DS(\theta)$ by following the approach taken in [4]:

- We pose the problem of finding a (complementary) solution to the homogeneous embedding (5) as finding a root of a (differentiable) map, a function of both an input to the embedding and the primal-dual pair's problem data θ.
- We consider the differentiability of this map and collect its derivatives with respect to both the embedding input and problem data.
- Using the implicit function theorem, we find $Ds(\theta)$ in terms of these derivatives. While $Ds(\theta)$ will require the evaluation of s at a point θ and we never find an expression for s directly, in practice we can supply such a point, $s(\theta)$, by using a (convex) quadratic conic optimization numerical solver.

2.3 Other machinery

This subsection closely follows [11].

The conic complementarity set. The *conic complementarity set* is defined as

$$\mathcal{C} = \left\{(u, v) \in K \times K^* \;\middle|\; u^T v = 0\right\}.$$

Let Π and Π° be the projections onto the cone K and its polar cone $K^\circ = -K^*$, respectively. Note the functional equality (a form of the Moreau decomposition) $\Pi^\circ = I - \Pi$, where I is the identity operator.

Minty's parameterization of the complementarity set. Let $M : \mathbf{R}^N \to \mathcal{C}$ be the Minty parameterization of $\mathcal{C}$, defined as

$$M(z) = (\Pi z, -\Pi^\circ z),$$

with inverse $M^{-1} : \mathcal{C} \to \mathbf{R}^N$ given by

$$M^{-1}(u, v) = u - v.$$

Unlike in [11],

$$-\Pi^\circ z = Q(\Pi z), \quad z_N \neq 0 \tag{6}$$

are not equivalent to the homogeneous embedded conditions (5). While z satisfying (6) implies that $u, v = M(z)$ satisfy (5), there exists a non-complementary solution (u, v) to (5) such that $z = M^{-1}(u, v)$ does not satisfy (6). However, (6) is equivalent to the KKT conditions (2).

Residual map. The residual map $\mathcal{R} : \mathbf{R}^N \to \mathbf{R}^N$ is defined as

$$\mathcal{R}(z) = Q\left(\Pi z\right) + \Pi^\circ z = Q\left(\Pi z\right) - \Pi z + z. \tag{7}$$

The map $\mathcal{R}$ is positive homogeneous and differentiable almost everywhere.

Normalized residual map. The *normalized residual map* $\mathcal{N} : \left\{ z \in \mathbf{R}^N \;\middle|\; z_N \neq 0 \right\} \to \mathbf{R}^N$ is defined as

$$\mathcal{N}(z) = \mathcal{R}(z/\left|z_N\right|) = \mathcal{R}(z)/\left|z_N\right|. \tag{8}$$

By (6), if $z \in \mathbf{R}^N$ is a solution to the conic pair (1) then $\mathcal{N}(z) = 0$. Conversely, if $\mathcal{N}(z) = 0$, then z is a solution to (1).

Data dependence. Throughout this note we have fixed the data defining the primal-dual pair (1), and consequently have not made explicit the dependence of $\mathcal{R}, \mathcal{N}, Q$ on θ. As we consider the derivative of these functions with respect to θ, we will update our notation writing

$$Q(u, \theta), \qquad \mathcal{R}(z, \theta), \quad \text{and} \quad \mathcal{N}(z, \theta).$$

2.4 Derivatives

Normalized residual map with respect to the data. The normalized residual map is an affine function of the problem data, θ. Therefore, it is differentiable with

$$D_\theta \mathcal{N}(z,\theta)[\widetilde{\theta}] = \frac{1}{|z_N|} D_\theta Q(\Pi z, \theta)[\widetilde{\theta}] \quad \text{and} \quad D_\theta \mathcal{N}(z,\theta)^T[w] = \frac{1}{|z_N|} D_\theta Q(\Pi z, \theta)^T[w],$$

where

$$D_\theta Q(u,\theta)[\widetilde{\theta}] = \begin{bmatrix} \widetilde{P}x + \widetilde{A}^T y + \tau \widetilde{q} \\ -\widetilde{A}x + \tau \widetilde{b} \\ (-1/\tau) x^T \widetilde{P} x - \widetilde{q}^T x - \widetilde{b}^T y \end{bmatrix}$$

and $D_\theta Q(u,\theta)^T[w] = (\widetilde{P}, \widetilde{A}, \widetilde{b}, \widetilde{q}, \widetilde{b})$ where

$$\widetilde{P} = 1/2 \left(w_{1:n} x^T + x w_{1:n}^T \right) - (w_N/\tau) x x^T, \qquad \widetilde{A} = y w_{1:n}^T - w_{n+1:n+m} x^T,$$
$$\widetilde{q} = \tau w_{1:n} - w_N x, \qquad \widetilde{b} = \tau w_{n+1:n+m} - w_N y,$$

for $w \in \mathbf{R}^N$ and $\widetilde{\theta} \in \Theta$.

Normalized residual map with respect to the variables. The normalized residual map is differentiable at z if $z_N \neq 0$ and Π is differentiable at z. When z is a solution of the primal-dual pair (1),

$$D_z \mathcal{N}(z,\theta) = \frac{1}{z_N} \left(D_z Q(\Pi z, \theta) D\Pi(z) - D\Pi(z) + I \right),$$

where the Jacobian of the nonlinear, homogeneous map Q with respect to the embedding input is

$$D_u Q(u,\theta) = \begin{bmatrix} P & A^T & q \\ -A & 0 & b \\ (-2/\tau) x^T P - q^T & -b^T & (1/\tau^2) x^T P x \end{bmatrix}.$$

Implicit function theorem applied to $\mathcal{N}$. If z is a solution of the primal-dual pair (1) and Π is differentiable at z, then $\mathcal{N}$ is differentiable at z, $N(z,\theta) = 0$, and $z_N > 0$. Now suppose that $D_z \mathcal{N}(z,\theta)$ is invertible. The implicit function theorem [33] guarantees that there exists a neighborhood $V \subseteq \Theta$ of θ on which the solution $z = s(\theta)$ of $\mathcal{N}(z,\theta)$ is unique. Furthermore, s is differentiable on V, $\mathcal{N}(s(\theta),\theta) = 0$ for all $\theta \in V$, and

$$Ds(\theta) = -\left(D_z \mathcal{N}(z,\theta) \right)^{-1} D_\theta \mathcal{N}(z,\theta).$$

Solution construction. To construct a solution (x, y, s) of the primal-dual pair (1) from a complementary solution z of the homogeneous embedding, we use the function ϕ given in [4]. With $\phi : \mathbf{R}^N \to \mathbf{R}^{n+2m}$ given by

$$\phi(z) = (z_{1:n}, \Pi_{\mathcal{K}^*}(z_{n+1:n+m}), \Pi_{\mathcal{K}^*}(z_{n+1:n+m}) - z_{n+1:n+m})/z_N.$$

If $\Pi_{\mathcal{K}^*}$ is differentiable at $z_{n+1:n+m}$, then ϕ is also differentiable and

$$D\phi(z) = \begin{bmatrix} I & 0 & -x \\ 0 & D\Pi_{\mathcal{K}^*}(z_{n+1:n+m}) & -y \\ 0 & D\Pi_{\mathcal{K}^*}(z_{n+1:n+m}) - I & -s \end{bmatrix}.$$

3 Implementation

3.1 Computing the Jacobian-vector product

Applying the derivative $D\mathcal{S}(\theta)$ to a perturbation $d\theta = (dP, dA, dq, db) \in \Theta$ corresponds to evaluating

$$\begin{aligned}(dx, dy, ds) = D\mathcal{S}(\theta)[d\theta] &= D\phi(s(\theta))Ds(\theta)[d\theta] \\ &= D\phi(z)\left(-\left(D_z\mathcal{N}(z, \theta)\right)^{-1}\right) D_\theta\mathcal{N}(z, \theta)[d\theta].\end{aligned}$$

Given a solution (x, y, s) to (1), we construct a root of the normalized residual map as $z = s(\theta) = M^{-1}(u, v) = u - v$ where $u = (x, y, 1)$ and $v = (0, s, 0)$.

We now work from right to left. First, we compute Πz and form $d_\theta\mathcal{N} = D_\theta\mathcal{N}(z, \theta)[d\theta]$. Second, we compute

$$dz = -F^{-1}d_\theta\mathcal{N},$$

where

$$F = D_z\mathcal{N}(z, \theta) = 1/z_N\left(D_zQ(\Pi z, \theta)D\Pi(z) - D\Pi(z) + I\right).$$

Since it is impractical to form or factor F as a dense matrix in some applications (*e.g.*, when F is large), we use LSMR [15] to solve

$$\underset{dz}{\text{minimize}} \quad \|F dz + d_\theta\mathcal{N}\|_2^2,$$

which only requires multiplication with F and F^T. Finally, we compute the solution perturbations as

$$\begin{bmatrix} dx \\ dy \\ ds \end{bmatrix} = \begin{bmatrix} dz_{1:n} - (dz_N)x \\ D\Pi_{\mathcal{K}^*}(z_{n+1:n+m})[dz_{n+1:n+m}] - (dz_N)y \\ D\Pi_{\mathcal{K}^*}(z_{n+1:n+m})[dz_{n+1:n+m}] - (dz)_{n+1:n+m} - (dz_N)s \end{bmatrix}.$$

3.2 Computing the vector-Jacobian product

The adjoint of the derivative applied to a perturbation (dx, dy, ds) is

$$\begin{aligned} d\theta = (dP, dA, dq, db) &= D\mathcal{S}(\theta)^T[(dx, dy, ds)] \\ &= Ds(\theta)^T D\phi(s(\theta))^T[(dx, dy, ds)] \\ &= D_\theta\mathcal{N}(z,\theta)^T \left(-\left(D_z\mathcal{N}(z,\theta)^T\right)^{-1}\right) D\phi(z)^T[(dx, dy, ds)], \end{aligned}$$

letting $z = s(\theta)$ as in §3.1. Working right to left, first, we evaluate

$$dz = D\phi(z)^T[(dx, dy, ds)] = \begin{bmatrix} dx \\ D\Pi_{\mathcal{K}^*}(z_{n+1:n+m})[dy + ds] - ds \\ -x^T dx - y^T dy - s^T ds \end{bmatrix}.$$

Second, we evaluate Πz and form $d_\theta\mathcal{N} = -F^{-T} dz$ using LSMR. Finally the problem data perturbation $d\theta = (dP, dA, dq, db)$ is given by

$$\begin{aligned} dP &= \frac{1}{2}\left(d_\theta\mathcal{N}_{1:n}\left(\Pi z\right)_{1:n}^T + \left(\Pi z\right)_{1:n} d_\theta\mathcal{N}_{1:n}^T\right) - \left(d_\theta\mathcal{N}_N / \left(\Pi z\right)_N\right)\left(\Pi z\right)_{1:n}\left(\Pi z\right)_{1:n}^T, \\ dA &= \left(\Pi z\right)_{n+1:n+m} d_\theta\mathcal{N}_{1:n}^T - d_\theta\mathcal{N}_{n+1:n+m}\left(\Pi z\right)_{1:n}^T, \\ dq &= \left(\Pi z\right)_N d_\theta\mathcal{N}_{1:n} - d_\theta\mathcal{N}_N\left(\Pi z\right)_{1:n}, \\ db &= \left(\Pi z\right)_N d_\theta\mathcal{N}_{n+1:n+m} - d_\theta\mathcal{N}_N\left(\Pi z\right)_{n+1:n+m}. \end{aligned}$$

However, we do not form dP and dA exactly as formulated. Instead we only compute their nonzero (or, more precisely, non-explicit-zero) entries as dictated by the sparsity patterns of P and A respectively.

3.3 Hardware accelerated Python implementation

We have developed an open-source JAX [10] library (also making significant use of the packages Equinox [22] and Lineax [30]), `diffqcp`, which implements these algorithms and is available at `https://github.com/cvxgrp/diffqcp`. Our implementation supports any QCP whose cone can be expressed as the Cartesian product of the zero cone, the positive orthant, second-order cones, and positive semidefinite cones. (Support for exponential cones, power cones, and their duals is in development.)

Data movement. Host-to-device transfers have been a long-standing limitation of CVXPYlayers, a Python library for constructing differentiable convex optimization layers in PyTorch [29], JAX, and TensorFlow [1] using CVXPY. Since CVXPYlayers only supports computing the derivative (and its adjoint) of the solution map of a conic program on the CPU, using this library to embed a differentiable convex optimization layer in a neural network requires tranferring any data on the device to the host during the forward or backward pass. Such transfers can be expensive, so having to perform them on both the forward and backward passes during every training iteration can make this embedding prohibitive. Being a JAX library, `diffqcp` can compute JVPs and VJPs on a GPU, allowing our software to be integrated into GPU workflows, such as neural network training, without these significant host-to-device data transfers.

Performance. `diffqcp` relies on JAX to enable its high performance. The JAX library uses Python as a "metaprogramming language" to build performant and just-in-time compiled XLA programs. Moreover we rely on the JAX transformation `vmap`, to simplify writing SIMD computations. `diffqcp` makes extensive use of this transformation to "batch" projections onto a family of cones with the same dimensionality. This batching is especially advantageous when computing JVPs and VJPs on a GPU, as it enables the execution of many independent computations in parallel, thereby maximizing processor occupancy and overall throughput.

3.4 Example

To test our implementation, we applied gradient descent to a loss function of the form

$$\|x - x^\star\|_2^2 + \|r - r^\star\|_2^2 + \|s - s^\star\|_2^2 + \|y - y^\star\|_2^2.$$

where each $(x, r), s, y$ are the optimal primal, slack, and dual solutions of

$$\begin{array}{ll} \text{minimize} & r^T P r + q^T \begin{bmatrix} x \\ r \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} C & D \\ E & 0 \\ f^T & 0^T \end{bmatrix} \begin{bmatrix} x \\ r \end{bmatrix} + s = b \\ & s \in \{0\}^m \times \mathbf{R}_+^n \times \{0\} \end{array} \tag{9}$$

with D, E, P are diagonal, and q, f, b are vectors and where $(x^\star, r^\star), s^\star, y^\star$ are the primal, slack, and dual solutions of

$$\begin{array}{ll} \text{minimize} & r^T I r \\ \text{subject to} & \begin{bmatrix} C^\star & -I \\ -I & 0 \\ \mathbf{1}^T & 0^T \end{bmatrix} \begin{bmatrix} x \\ r \end{bmatrix} + s = \begin{bmatrix} d^\star \\ 0 \\ 1 \end{bmatrix} \\ & s \in \{0\}^m \times \mathbf{R}_+^n \times \{0\} \end{array}$$

for a randomly selected $C^\star, d^\star$.

Results. We take $m = 2000$ and $n = 1000$. CuClarabel and `diffqcp` on an Intel Xeon E5-2670 CPU and an NVIDIA TITAN Xp GPU took 44.20 seconds per iteration. As a control, we canonicalized the objective with SOCs and ran the gradient descent with Clarabel and `diffcp`, which took 96.86 seconds per iteration on the Intel Xeon E5-2670 CPU. The improved modeling capacity and GPU-acceleration enabled a 2.19× speedup.

A Cones, projections, and derivatives

Zero cone. The zero cone $\{0\}$ has dual cone $\mathbf{R}$, projection operation $\Pi_{\{0\}}(z) = 0$, and $D\Pi(z)[dz] = 0$.

Nonnegative cone. The nonnegative cone $\{x \mid x \geq 0\}$ has dual cone $\{x \mid x \geq 0\}$, projection operator

$$\Pi_{\{x|x\geq 0\}}(z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0, \end{cases}$$

and

$$D\Pi_{\{x|x\geq 0\}}(z)[dz] = \begin{cases} dz & z > 0 \\ 0 & z < 0. \end{cases}$$

Second-order cone. The second order cone $\mathcal{K}_{\text{soc}} = \{(t,u) \in \mathbf{R} \times \mathbf{R}^n : \|u\|_2 \leq t\}$ has dual cone $\mathcal{K}_{\text{soc}}$, projection operator

$$\Pi_{\text{soc}}((t,u)) = \begin{cases} 0 & \|u\|_2 \leq -t \\ (t,u) & \|u\|_2 \leq t \\ (1/2)(1 + t/\|u\|_2)(\|u\|_2, u) & \|u\|_2 \geq |t| \end{cases}$$

and

$$D\Pi_{\text{soc}}((t,u))[(dt,du)] = \begin{cases} 0 & \|u\|_2 < -t \\ (dt,du) & \|u\|_2 < t \\ \frac{1}{2\|u\|_2}\begin{bmatrix} \|u\|_2 dt + u^T du \\ u dt + (t + \|u\|_2) du - (t/\|u\|_2^2)(u^T du) u \end{bmatrix} & \|u\|_2 > |t|. \end{cases}$$

Positive semidefinite cone. The positive semidefinite cone $\mathcal{K}_{\text{psd}} = \{A \in \mathbf{S}^n \mid A \succeq 0\}$ has dual cone $\mathcal{K}_{\text{psd}}$, projection operator

$$\Pi_{\text{psd}}(Z) = \sum_{i=1}^{n} \max\{0, \lambda_i\} v_i v_i^T$$

where $\{(\lambda_i, v_i) \mid \lambda_1 \geq \cdots \geq \lambda_i \geq \cdots \geq \lambda_n\}$ is the eigendecomposition of Z, and

$$D\Pi(Z)[dZ] = V\left(B \circ \left(V^T dZ V\right)\right) V^T,$$

where $\circ$ denotes the Hadamard (*i.e.*, element-wise) product, $k = \min\{k \mid \lambda_k < 0\}$. The symmetric B is given by

$$B_{ij} = \begin{cases} 0 & i \geq k, j \geq k \\ \frac{\lambda_i}{\lambda_i - \lambda_j} & i < k, j \geq k \\ \frac{\lambda_j}{\lambda_j - \lambda_i} & i \geq k, j < k \\ 1 & i < k, j < k. \end{cases}$$

See [11] for the derivation and proof of $D\Pi_{\text{psd}}$.

Exponential cone. The exponential cone

$$\mathcal{K}_{\text{exp}} = \left\{(x,y,z) \in \mathbf{R}^3 \,\middle|\, ye^{x/y} \leq z, y > 0\right\} \cup \left\{(x,0,z) \in \mathbf{R}^3 \,\middle|\, x \leq 0, z \geq 0\right\}$$

has dual cone

$$\mathcal{K}^*_{\text{exp}} = \left\{(u,v,w) \in \mathbf{R}^3 \,\middle|\, u < 0, -ue^{v/u} \leq ew\right\} \cup \left\{(0,v,w) \in \mathbf{R}^3 \,\middle|\, v \geq 0, w \geq 0\right\}$$

and polar cone $\mathcal{K}^\circ_{\text{exp}} = -\mathcal{K}^*_{\text{exp}}$. Let $p = (x,y,z)$.

- For $p \in \mathcal{K}_{\text{exp}}$, $\Pi_{\text{exp}}(p) = p$.
- For $p \in \mathcal{K}^\circ_{\text{exp}}$, $\Pi_{\text{exp}}(p) = 0$.
- For $p \notin \mathcal{K}_{\text{exp}} \cup \mathcal{K}^\circ_{\text{exp}}$ with $x < 0$ and $y < 0$, $\Pi_{\text{exp}}(p) = (x, 0, \max\{z, 0\})$.
- For all other p, the projection $\Pi_{\text{exp}}(p)$ must be found via its definition, *i.e.*, as the (unique) solution to

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|\hat{p} - p\| \\ \text{subject to} & \hat{z} = \hat{y}e^{\hat{x}/\hat{y}},\ \hat{y} > 0, \end{array} \tag{10}$$

  where $\hat{p} = (\hat{x}, \hat{y}, \hat{z})$ is the optimization variable. See [17] for a fast and numerically robust univariate root-finding algorithm that can be used to compute the projection.

In the following cases, the projection operator is differentiable at p.

- For $p \in \textbf{int}\,\mathcal{K}_{\text{exp}}$, $D\Pi_{\text{exp}}(p) = D\Pi_{\text{exp}}(p)^T = I$.
- For $p \in \textbf{int}\,\mathcal{K}^\circ_{\text{exp}}$, $D\Pi_{\text{exp}}(p) = D\Pi_{\text{exp}}(p)^T = 0$.

- For $p \notin \mathcal{K}_{\text{exp}} \cup \mathcal{K}^{\circ}_{\text{exp}}$ with $x < 0$, $y < 0$, and $z \neq 0$, $D\Pi_{\text{exp}}(p) = D\Pi_{\text{exp}}(p)^T = \mathbf{diag}(1, 0, \mathbf{1}\{z > 0\})$ where for $\alpha \in \mathbf{R}$, $\mathbf{1}\{\alpha > 0\}$ is 1 if $\alpha > 0$ else is 0.

- For $p \in \mathbf{int}\left(\mathbf{R}^3 \setminus \left(\mathcal{K}_{\text{exp}} \cup \mathcal{K}^*_{\text{exp}} \cup (\mathbf{R}_- \times \mathbf{R}_- \times \mathbf{R})\right)\right)$, $D\Pi_{\text{exp}}(p) = (J^{-1})_{1:3,1:3}$ where

$$J = \begin{bmatrix} 1 + \frac{\mu^\star e^{x^\star/y^\star}}{y^\star} & -\frac{\mu^\star x^\star e^{x^\star/y^\star}}{(y^\star)^2} & 0 & e^{x^\star/y^\star} \\ -\frac{\mu^\star x^\star e^{x^\star/y^\star}}{(y^\star)^2} & 1 + -\frac{\mu^\star (x^\star)^2 e^{x^\star/y^\star}}{(y^\star)^3} & 0 & (1 - x^\star/y^\star)e^{x^\star/y^\star} \\ 0 & 0 & 1 & -1 \\ e^{x^\star/y^\star} & (1 - x^\star/y^\star)e^{x^\star/y^\star} & -1 & 0 \end{bmatrix}.$$

  In this Jacobian, $(x^\star, y^\star, z^\star)$ is the solution to (10) and $\mu^\star \in \mathbf{R}$ is the solution to the dual problem. See [6] for the derivation and proof of J.

Dual exponential cone. The dual exponential cone is given above. Via the Moreau decomposition, its projection operator is $\Pi_{\text{exp}*}(z) = z + \Pi_{\text{exp}}(-z)$ with derivative $D\Pi_{\text{exp}*}(z)[dz] = dz - D\Pi_{\text{exp}}(-z)[dz]$

Power cone. Our power cone and dual power cone results are based on [21]. The (3D) power cone

$$\mathcal{K}_{\text{pow},\alpha} = \left\{(x, y, z) \in \mathbf{R}^3 \;\middle|\; x^\alpha y^{1-\alpha} \geq |z|, x \geq 0, y \geq 0\right\}$$

has dual cone

$$\mathcal{K}^*_{\text{pow},\alpha} = \left\{(u, v, w) \in \mathbf{R}^3 \;\middle|\; \left(\frac{u}{\alpha}\right)^\alpha \left(\frac{v}{1-\alpha}\right)^{1-\alpha} \geq |w|, u \geq 0, v \geq 0\right\}$$

and polar cone $\mathcal{K}^{\circ}_{\text{pow},\alpha} = -\mathcal{K}^*_{\text{pow},\alpha}$. Let $p = (x, y, z)$.

- For $p \in \mathcal{K}_{\text{pow},\alpha}$, $\Pi_{\text{pow},\alpha}(p) = p$.

- For $p \in \mathcal{K}^{\circ}_{\text{pow},\alpha}$, $\Pi_{\text{pow}}(p) = 0$.

- For $p \notin \mathcal{K}_{\text{pow},\alpha} \cup \mathcal{K}^{\circ}_{\text{pow},\alpha}$ and $z = 0$, $\Pi_{\text{pow}}(p) = (\max\{x, 0\}, \max\{y, 0\}, 0)$.

- For $p \notin \mathcal{K}_{\text{pow},\alpha} \cup \mathcal{K}^{\circ}_{\text{pow},\alpha}$ and $z \neq 0$, $\Pi_{\text{pow}}(p) = (f_x, f_y, \mathbf{sign}(z) r)$, where

$$f_x(r) = \frac{1}{2}\left(x + \sqrt{x^2 + 4\alpha r(|z| - r)}\right), \quad f_y(r) = \frac{1}{2}\left(y + \sqrt{y^2 + 4(1-\alpha) r(|z| - r)}\right),$$

and r is the (unique) solution to the (nonconvex) problem

$$\begin{array}{ll} \text{find} & r \\ \text{subject to} & r = f_x(r)^\alpha f_y(r)^{1-\alpha} \\ & 0 < r < |z| \end{array} \tag{11}$$

defined in [21, (5)].

In the following cases, the projection operator is continuously differentiable at p.

- For $p \in \mathbf{int}\, \mathcal{K}_{\text{pow},\alpha}$, $D\Pi_{\text{pow}}(p) = D\Pi_{\text{pow}}(p)^T = I$.
- For $p \in \mathbf{int}\, \mathcal{K}^\circ_{\text{pow},\alpha}$, $D\Pi_{\text{pow}}(p) = D\Pi_{\text{pow}}(p)^T = 0 \in \mathbf{R}^{3\times 3}$.
- [21, Theorem 3.1] For $p \notin \mathcal{K}_{\text{pow},\alpha} \cup \mathcal{K}^\circ_{\text{pow},\alpha}$ and $z \neq 0$,

$$D\Pi_{\text{pow}}(p) = \begin{bmatrix} \frac{1}{2} + \frac{x}{2g_x} + \frac{\alpha^2(|z|-2r)rL}{g_x^2} & \frac{(\alpha-\alpha^2)(|z|-2r)rL}{g_y g_x} & \mathbf{sign}(z)\frac{\alpha r L}{g_x} \\ \frac{(\alpha-\alpha^2)(|z|-2r)rL}{g_x g_y} & \frac{1}{2} + \frac{y}{2g_y} + \frac{(1-\alpha)^2(|z|-2r)rL}{g_y^2} & \mathbf{sign}(z)\frac{(1-\alpha)rL}{g_y} \\ \mathbf{sign}(z)\frac{\alpha r L}{g_x} & \mathbf{sign}(z)\frac{(1-\alpha)rL}{g_y} & \frac{r}{|z|} + \frac{r}{|z|}TL \end{bmatrix},$$

  where $g_x = 2f_x - x, g_y = 2f_y - y$,

$$L = \frac{2(|z|-r)}{|z| + (|z|-2r)(\frac{\alpha x}{g_x} + \frac{(1-\alpha)y}{g_y})}, \quad T = -\left(\frac{\alpha x}{g_x} + \frac{(1-\alpha)y}{g_y}\right),$$

  and r is the solution of (11).

- For $p \notin \mathcal{K}_{\text{pow},\alpha} \cup \mathcal{K}^\circ_{\text{pow},\alpha}$, $z = 0$, and $x, y \neq 0$,

$$D\Pi_{\text{pow}}(p) = \begin{bmatrix} \mathbf{1}\{x > 0\} & 0 & 0 \\ 0 & \mathbf{1}\{y > 0\} & 0 \\ 0 & 0 & d \end{bmatrix}.$$

  The component d is defined as

$$d = \begin{cases} 1 & x > 0, y < 0, \alpha > 1/2, \text{ or } y > 0, x < 0, \alpha < 1/2 \\ 0 & x > 0, y < 0, \alpha < 1/2, \text{ or } y > 0, x < 0, \alpha > 1/2 \\ d_x & x > 0, y < 0, \alpha = 1/2 \\ d_y & x < 0, y > 0, \alpha = 1/2, \end{cases}$$

  where

$$d_x = \frac{x}{2|y| + x} \quad \text{and} \quad d_y = \frac{y}{2|x| + y}.$$

Dual power cone. The dual power cone is given above. Via the Moreau decomposition, its projection operator is $\Pi_{\text{pow}*}(z) = z + \Pi_{\text{pow}}(-z)$ with derivative $D\Pi_{\text{pow}*}(z)[dz] = dz - D\Pi_{\text{pow}}(-z)[dz]$

Acknowledgements

Parth Nobel was supported in part by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1656518. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The work of Stephen Boyd was supported by ACCESS (AI Chip Center for Emerging Smart Systems), sponsored by InnoHK funding, Hong Kong SAR.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

[3] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato. Learning convex optimization control policies. In A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, editors, *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pages 361–373. PMLR, 10–11 Jun 2020.

[4] A. Agrawal, S. T. Barratt, S. P. Boyd, E. Busseti, and W. M. Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 2019.

[5] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[6] A. Ali, E. Wong, and J. Z. Kolter. A semismooth newton method for fast, generic convex programming. In *International Conference on Machine Learning*, 2017.

[7] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. *arXiv preprint arXiv:1703.00443*, 2017.

[8] A. Bambade, F. Schramm, S. E. Kazdadi, S. Caron, A. Taylor, and J. Carpentier. PROXQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond. working paper or preprint, Sept. 2023.

[9] S. T. Barratt and S. P. Boyd. Fitting a kalman smoother to data. *2020 American Control Conference (ACC)*, pages 1526–1531, 2019.

[10] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[11] E. Busseti, W. M. Moursi, and S. P. Boyd. Solution refinement at regular points of conic problems. *Computational Optimization and Applications*, 74:627 – 643, 2018.

[12] Y. Chen, D. Tse, P. Nobel, P. Goulart, and S. Boyd. CuClarabel: GPU acceleration for a conic optimization solver, 2024.

[13] A. Degleris, A. E. Gamal, and R. Rajagopal. Gradient methods for scalable multi-value electricity network expansion planning, 2024.

[14] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *2013 European control conference (ECC)*, pages 3071–3076. IEEE, 2013.

[15] D. C.-L. Fong and M. A. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM J. Sci. Comput.*, 33:2950–2971, 2010.

[16] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[17] H. A. Friberg. Projection onto the exponential cone: a univariate root-finding problem. *Optimization Methods and Software*, 38:457 – 473, 2023.

[18] A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 94(14):1–34, 2020.

[19] P. Goulart and Y. Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives. 2024.

[20] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. http://stanford.edu/~boyd/graph_dcp.html.

[21] L. T. K. Hien. Differential properties of euclidean projection onto power cone. *Mathematical Methods of Operations Research*, 82:265–284, 2015.

[22] P. Kidger and C. Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[23] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[24] P. Nobel, E. Candès, and S. Boyd. Tractable evaluation of stein's unbiased risk estimate with convex regularizers. *IEEE Transactions on Signal Processing*, 71:4330–4341, 2023.

[25] P. Nobel, D. LeJeune, and E. J. Candès. RandALO: Out-of-sample risk estimation in no time flat, 2024.

[26] G. Négiar, M. W. Mahoney, and A. S. Krishnapriyan. Learning differentiable solvers for systems with hard constraints, 2023.

[27] B. O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31(3):1999–2023, 2021.

[28] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, Jun 2016.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[30] J. Rader, T. Lyons, and P. Kidger. Lineax: unified linear solves and linear least-squares in jax and equinox. *AI for science workshop at Neural Information Processing Systems 2023, arXiv:2311.17283*, 2023.

[31] S. M. Robinson. Some continuity properties of polyhedral multifunctions. *Mathematical Programming*, 10(1):128–141, 1981.

[32] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[33] R. T. Rockafellar. Variational analysis. *Grundlehren der Mathematischen Wissenschaften*, 317, 1998.

[34] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.