Differentiating Through a Cone Program

Akshay Agrawal Shane Barratt Stephen Boyd Enzo Busseti
Walaa M. Moursi*

May 21, 2020

Abstract

We consider the problem of efficiently computing the derivative of the solution map of a convex cone program, when it exists. We do this by implicitly differentiating the residual map for its homogeneous self-dual embedding, and solving the linear systems of equations required using an iterative method. This allows us to efficiently compute the derivative operator, and its adjoint, evaluated at a vector. These correspond to computing an approximate new solution, given a perturbation to the cone program coefficients (*i.e.*, perturbation analysis), and to computing the gradient of a function of the solution with respect to the coefficients. Our method scales to large problems, with numbers of coefficients in the millions. We present an open-source Python implementation of our method that solves a cone program and returns the derivative and its adjoint as abstract linear maps; our implementation can be easily integrated into software systems for automatic differentiation.

1 Introduction

A *cone program* is an optimization problem in which the objective is to minimize a linear function over the intersection of a subspace and a convex cone. Cone programs include linear programs, second-order cone programs, and semidefinite programs. Indeed, every convex optimization problem can be expressed as a cone program [Nem07].

Cone programs can be efficiently solved by a number of methods, including interior-point methods [NN94] and operator splitting methods such as the alternating directions method of multipliers [OCPB16]. Cone programs have found applications in many areas, including control [BEFB94], machine learning [HTF09, BPC+11], finance [Mar52, BBD+17], supply chain management [BT04, BTGNV05], and energy management [MBBW19], to name just a few. Cone programs are widely used in conjunction with several popular software systems for convex optimization, which reformulate a convex optimization problem expressed in a domain specific language into an equivalent cone program. These include YALMIP [Löf04], CVX [GB14], CVXPY [DB16a, AVDB18], CVXR [FNB19], and Convex.jl [UMZ+14].

Solution maps and implicit functions. An optimization problem can be viewed as a (possibly multi-valued) function mapping the problem data, *i.e.*, the numerical data defining the problem, to the (primal and dual) solution. This *solution map* is in general set-valued. In neighborhoods

*Authors listed in alphabetical order.

where the solution map is single-valued, it is an implicit function of the problem data. In these neighborhoods it becomes meaningful and interesting to discuss how perturbations in the problem data affect the solution. The point of this paper is to calculate the effects of such perturbations for cone programs, in an efficient way.

The use of implicit differentiation to study the sensitivity of solution mappings of optimization problems dates back several decades, with the works of Fiacco [FM68] and Robinson [Rob80] marking significant milestones. The book [BS00] provides a thorough treatment of the subject, and [DR09] is a good reference for implicit functions more generally. In recent applications, the framework of implicit functions has been used to differentiate through quadratic programs [AK17], stochastic optimization [DAK17], physics simulators [dSA+18], control algorithms [AJS+18], and games [LFK18].

Automatic differentiation (AD). Contemporary interest in applications of implicit differentiation is partly due to the availability of high-quality, modern, open-source AD software. AD is a family of techniques for algorithmically computing exact derivatives of compositions of differentiable functions. Techniques for AD have been known since at least the 1950s [BKSF59]; see also [Wen64, Gri89, GW08]. In fact, AD is essentially an efficient way of computing the chain rule, which can be traced back to a centuries-old manuscript by Leibniz [Lei76] (see [RF10] for a detailed history).

There are two main variants of AD. Reverse-mode AD computes the derivative of a composition of atomic differentiable functions by computing the sensitivity of an output with respect to the intermediate variables (without materializing the matrices for the intermediate derivatives). In this way, reverse-mode can efficiently compute the derivatives of scalar-valued functions. Forward-mode AD computes the derivative by calculating the sensitivity of the intermediate variables with respect to an input variable [GW08, §2].

Reverse-mode AD was implemented as early as the 1980s [Spe80]. Its rediscovery by the machine learning community (where it is known as backpropagation [RHW88]) and the modern popularity of deep neural networks have led to the development of many software libraries for reverse-mode AD. Examples include TensorFlow [ABC+16, AMP+19], PyTorch [PGC+17], autograd [MDA15], and Zygote [Inn19]. The library JAX [FJL18] supports both reverse-mode and forward-mode AD. This requires representing the derivative of each atomic function as an *abstract linear map*, *i.e.*, as methods that apply the derivative and its adjoint [DB16b]; implementing just reverse-mode AD only requires the adjoint. Many of the atomic functions included in these libraries are not differentiable at all points in their domains. At non-differentiable points, libraries compute heuristic quantities instead of derivatives. For a discussion on non-differentiability as it relates to AD, see [GW08, §14].

This paper. In this paper, we give conditions that guarantee the existence of the derivative of the solution map for a cone program, and we provide an expression for the derivative at points where these conditions are satisfied. As in [BMB18], our formulation involves expressing the cone program as the problem of finding a zero of a particular function, specifically the residual map for a homogeneous self-dual embedding of the program [YTM94, OCPB16]. We also show how to efficiently compute the derivative and its adjoint, which involves computing projections onto convex cones, solving a linear system, and exploiting sparsity. In §3, we present an open-source Python package that furnishes the derivative of a cone program as an abstract linear map.

2 The solution map and its derivative

We consider a (convex) conic optimization problem in its primal (P) and dual (D) forms (see, *e.g.*, [BV04, §4.6.1] or [BTN01, §1.4]):

$$\begin{array}{llll} \text{(P)} & \text{minimize} & c^T x & \\ & \text{subject to} & Ax + s = b & \\ & & s \in \mathcal{K}, & \end{array} \qquad \begin{array}{llll} \text{(D)} & \text{minimize} & b^T y & \\ & \text{subject to} & A^T y + c = 0 & (1) \\ & & y \in \mathcal{K}^*. & \end{array}$$

Here $x \in \mathbf{R}^n$ is the *primal* variable, $y \in \mathbf{R}^m$ is the *dual* variable, and $s \in \mathbf{R}^m$ is the primal *slack* variable. The set $\mathcal{K} \subseteq \mathbf{R}^m$ is a nonempty, closed, convex cone with *dual cone* $\mathcal{K}^* \subseteq \mathbf{R}^m$. The *problem data* are $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$, $c \in \mathbf{R}^n$, and the cone $\mathcal{K}$. (In the sequel, however, we will consider the cone as fixed.) In the following we let $N = m + n + 1$, and use Π to denote the projection onto $\mathbf{R}^n \times \mathcal{K}^* \times \mathbf{R}_+$. Finally, we define

$$\mathcal{Q} = \left\{ Q = \begin{bmatrix} 0 & A^T & c \\ -A & 0 & b \\ -c^T & -b^T & 0 \end{bmatrix} \in \mathbf{R}^{N\times N} \;\middle|\; (A, b, c) \in \mathbf{R}^{m\times n} \times \mathbf{R}^m \times \mathbf{R}^n \right\}.$$

Evidently $\mathcal{Q}$ is a proper subspace of the space of $N \times N$ skew symmetric matrices.

The solution map. We call (x, y, s) a solution of the primal-dual conic program (1) if

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

For given problem data, the corresponding primal-dual conic program (1) may have no solution, a unique solution, or multiple solutions. We focus on the case when it has a unique solution. We define the *solution map* $\mathcal{S} : \mathbf{R}^{m\times n} \times \mathbf{R}^m \times \mathbf{R}^n \to \mathbf{R}^{n+2m}$ as the function mapping (A, b, c) to vectors (x, y, s) that satisfy (2). We express $\mathcal{S}$ as the composition $\phi \circ s \circ Q$, where

- $Q : \mathbf{R}^{m\times n} \times \mathbf{R}^m \times \mathbf{R}^n \to \mathcal{Q}$ maps the problem data to the corresponding skew-symmetric matrix in $\mathcal{Q}$,
- $s : \mathcal{Q} \to \mathbf{R}^N$ furnishes a solution of the homogeneous self-dual embedding, and
- $\phi : \mathbf{R}^N \to \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}^m$ maps a solution of the homogeneous self-dual embedding to the solution of the primal-dual pair (1).

At a point (A, b, c) where $\mathcal{S}$ is differentiable, the derivative of the solution map is

$$\mathsf{D}\mathcal{S}(A, b, c) = \mathsf{D}\phi(z)\mathsf{D}s(Q)\mathsf{D}Q(A, b, c),$$

by the chain rule. In the remainder of this section, we describe the functions Q, s, and ϕ and their derivatives, along with sufficient conditions for their differentiability.

Skew-symmetric mapping. Define

$$Q = Q(A, b, c) = \begin{bmatrix} 0 & A^T & c \\ -A & 0 & b \\ -c^T & -b^T & 0 \end{bmatrix} \in \mathcal{Q}.$$

Homogeneous self-dual embedding. The homogeneous self-dual embedding of (1) uses the variable $z \in \mathbf{R}^N$. We partition z as $(u, v, w) \in \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}$. The *normalized residual map* introduced in [BMB18] is the function $\mathcal{N} : \mathbf{R}^N \times \mathcal{Q} \to \mathbf{R}^N$, defined by

$$\mathcal{N}(z, Q) = \big((Q - I)\Pi + I\big)(z/|w|).$$

For a given Q, z can be used to construct the solution of the primal-dual pair (1) if and only if $\mathcal{N}(z, Q) = 0$ and $w > 0$ [BMB18].

The normalized residual map is an affine function of Q, hence its derivative $\mathsf{D}_Q\mathcal{N}$ always exists, and is given by

$$\mathsf{D}_Q\mathcal{N}(z, Q)(U) = U\Pi(z/|w|), \quad \mathsf{D}_Q\mathcal{N}(z, Q)^T(y) = y(\Pi(z/|w|))^T.$$

We now turn to $\mathsf{D}_z\mathcal{N}(z, Q)$. $\mathcal{N}$ is differentiable at z, with $w \neq 0$, whenever Π is differentiable at z, in which case one can directly verify that

$$\mathsf{D}_z\mathcal{N}(z, Q) = ((Q - I)\mathsf{D}\Pi(z) + I)/w - \operatorname{sign}(w)((Q - I)\Pi + I)(z/w^2)e^T, \tag{3}$$

where $e = (0, 0, \ldots, 1) \in \mathbf{R}^N$. In particular, when z is a solution of the primal-dual pair (1), the second term on the right hand side of (3) vanishes and we have

$$\mathsf{D}_z\mathcal{N}(z, Q) = ((Q - I)\mathsf{D}\Pi(z) + I)/w.$$

Implicit function theorem. Suppose that z is a solution of the primal-dual pair (1) for a given Q, and that Π is differentiable at z. Then $\mathcal{N}$ is differentiable at z, $\mathcal{N}(z, Q) = 0$ and $w > 0$. Now, suppose that $\mathsf{D}_z\mathcal{N}(z, Q)$ is invertible. It follows from the implicit function theorem (see, *e.g.*, [Din07] and [DR09]) that there exists a neighborhood $V \subseteq \mathcal{Q}$ of Q on which the solution $z = s(Q)$ of $\mathcal{N}(z, Q) = 0$ is unique. Furthermore, s is differentiable on V, $\mathcal{N}(s(Q), Q) = 0$ for all $Q \in V$, and

$$\mathsf{D}s(Q) = -(\mathsf{D}_z\mathcal{N}(s(Q), Q))^{-1}\mathsf{D}_Q\mathcal{N}(s(Q), Q).$$

Solution construction. The function $\phi : \mathbf{R}^N \to \mathbf{R}^{n+2m}$, given by

$$\phi(z) = (u, \Pi_{\mathcal{K}^*}(v), \Pi_{\mathcal{K}^*}(v) - v)/w,$$

constructs a solution (x, y, s) of the primal-dual pair (1) from a solution $z = (u, v, w)$ of the homogeneous self-dual embedding ($\Pi_{\mathcal{K}^*}$ denotes the projection onto $\mathcal{K}^*$). If $\Pi_{\mathcal{K}^\star}$ is differentiable with derivative $\mathsf{D}\Pi_{\mathcal{K}^\star}(v) \in \mathbf{R}^{m\times m}$, then ϕ is also differentiable, with derivative

$$\mathsf{D}\phi(z) = \begin{bmatrix} I & 0 & -x \\ 0 & \mathsf{D}\Pi_{\mathcal{K}^*}(v) & -y \\ 0 & \mathsf{D}\Pi_{\mathcal{K}^*}(v) - I & -s, \end{bmatrix}.$$

Note that a solution of the homogeneous self-dual embedding can also be constructed from a solution of the primal-dual pair [BMB18].

3 Implementation

In this section we detail how to form the derivative of a cone program as an abstract linear map. We also describe our Python package that implements these methods, and as an example use it to differentiate a semidefinite program.

Sparsity. The matrix A is often stored and manipulated as a sparse matrix. We assume that the sparsity pattern of A is fixed, meaning we only have to consider the nonzero entries of A when computing the derivative and its adjoint; this can provide significant speed-ups if A is very sparse. Of course, one can still furnish the derivative with respect to every entry of A by making A dense.

Computing the derivative. Applying the derivative $\mathsf{D}\mathcal{S}(A,b,c)$ to a perturbation $(\mathsf{d}A,\mathsf{d}b,\mathsf{d}c)$ corresponds to evaluating

$$(\mathsf{d}x,\mathsf{d}y,\mathsf{d}s) = \mathsf{D}\mathcal{S}(A,b,c)(\mathsf{d}A,\mathsf{d}b,\mathsf{d}c) = \mathsf{D}\phi(z)\mathsf{D}s(Q)\mathsf{D}Q(A,b,c)(\mathsf{d}A,\mathsf{d}b,\mathsf{d}c),$$

where $\mathsf{d}A$ has the same sparsity pattern as A. We work from right to left. The first step is to form

$$\mathsf{d}Q = \begin{bmatrix} 0 & \mathsf{d}A^T & \mathsf{d}c \\ -\mathsf{d}A & 0 & \mathsf{d}b \\ -\mathsf{d}c^T & -\mathsf{d}b^T & 0 \end{bmatrix}.$$

The next step is to compute

$$g = \mathsf{D}_Q\mathcal{N}(s(Q),Q)(\mathsf{d}Q) = \mathsf{d}Q\Pi(z/|w|),$$

and then

$$\mathsf{d}z = -M^{-1}g,$$

where $M = ((Q-I)\mathsf{D}\Pi(z)+I)/w$. One option is to form M as a dense matrix, factorize it, and then back-solve. However, when M is large, as is the case in many applications, this can be impractical. Instead, we suggest using LSQR [PS82] to solve

$$\underset{\mathsf{d}z}{\text{minimize}} \quad \|M\mathsf{d}z + g\|_2^2,$$

which only requires multiplication by M and M^T, and is of particular interest for semidefinite cone programs. Next we partition $\mathsf{d}z$ as $\mathsf{d}z = (\mathsf{d}u,\mathsf{d}v,\mathsf{d}w)$. The final step is to compute

$$\begin{bmatrix} \mathsf{d}x \\ \mathsf{d}y \\ \mathsf{d}s \end{bmatrix} = \begin{bmatrix} \mathsf{d}u - (\mathsf{d}w)x \\ \mathsf{D}\Pi_{\mathcal{K}^*}(v)\mathsf{d}v - (\mathsf{d}w)y \\ \mathsf{D}\Pi_{\mathcal{K}^*}(v)\mathsf{d}v - \mathsf{d}v - (\mathsf{d}w)s \end{bmatrix}.$$

Computing the adjoint of the derivative. Applying the adjoint of the derivative to a perturbation $(\mathsf{d}x,\mathsf{d}y,\mathsf{d}s)$ corresponds to evaluating

$$(\mathsf{d}A,\mathsf{d}b,\mathsf{d}c) = \mathsf{D}\mathcal{S}(A,b,c)^T(\mathsf{d}x,\mathsf{d}y,\mathsf{d}s) = \mathsf{D}Q(A,b,c)^T\mathsf{D}s(Q)^T\mathsf{D}\phi(z)^T(\mathsf{d}x,\mathsf{d}y,\mathsf{d}s).$$

We again work right to left. The first step is to form

$$\mathsf{d}z = \mathsf{D}\phi(z)^T(\mathsf{d}x,\mathsf{d}y,\mathsf{d}s) = \begin{bmatrix} \mathsf{d}x \\ \mathsf{D}\Pi^T_{\mathcal{K}^*}(v)(\mathsf{d}y+\mathsf{d}s) - \mathsf{d}s \\ -x^T\mathsf{d}x - y^T\mathsf{d}y - s^T\mathsf{d}s \end{bmatrix}.$$

Next we form $g = -M^{-T}\mathsf{d}z$, again using LSQR. Then $\mathsf{d}Q$ is given by

$$\mathsf{d}Q = g(\Pi(z/|w|))^T.$$

Instead of explicitly forming $\mathrm{d}Q$, we only obtain its nonzero entries. Let its nonzero entries be indexed by Ω; we compute

$$(\mathrm{d}Q)_{ij} = \begin{cases} g_i \Pi(z/|w|)_j & (i,j) \in \Omega \\ 0 & \text{otherwise.} \end{cases}$$

Partitioning $\mathrm{d}Q$ as

$$\mathrm{d}Q = \begin{bmatrix} \mathrm{d}Q_{11} & \mathrm{d}Q_{12} & \mathrm{d}Q_{13} \\ \mathrm{d}Q_{21} & \mathrm{d}Q_{22} & \mathrm{d}Q_{23} \\ \mathrm{d}Q_{31} & \mathrm{d}Q_{32} & \mathrm{d}Q_{33} \end{bmatrix},$$

the final expressions are given by

$$\begin{aligned} \mathrm{d}A &= \mathrm{d}Q_{12}^T - \mathrm{d}Q_{21} \\ \mathrm{d}b &= \mathrm{d}Q_{23} - \mathrm{d}Q_{32}^T \\ \mathrm{d}c &= \mathrm{d}Q_{13} - \mathrm{d}Q_{31}^T. \end{aligned}$$

Integration into AD. The calculations that we have described can be immediately be integrated into the forward and reverse-mode AD software described in §1. In a forward-mode AD system, one calculates the sensitivity of the intermediate variables with respect to perturbations; this operation corresponds to applying the derivative to those perturbations. In a reverse-mode AD system, one computes the sensitivity of a scalar function with respect to intermediate variables; this operation is given by applying the adjoint of the derivative to the derivative of the scalar function with respect to x, y, and s.

3.1 Python implementation

We provide a Python implementation of the ideas described in the paper, available at

`https://www.github.com/cvxgrp/diffcp`.

We use the libraries NumPy for dense linear algebra [VDWCV11] and SciPy for sparse linear algebra and its LSQR implementation [JOP+01]. To solve the cone program, we use the numerical solver SCS [OCPB16]. Our implementation supports any cone that can be expressed as the Cartesian product of the zero cone, positive orthant, second-order cone, positive semidefinite cone, exponential cone, and dual exponential cone. Expressions for the derivative of the projection onto each of these cones are given in [AWK17] and [BMB18]; most of these have analytical expressions.

The Python package exposes one function,

`solve_and_derivative(A, b, c, cone_dict)`,

where `A` is a SciPy sparse matrix, `b` and `c` are NumPy arrays, and `cone_dict` is a dictionary representing the cone $\mathcal{K}$. This function returns a solution (x, y, s) of the primal-dual pair and two functions, `derivative(dA, db, dc)` and `adjoint_derivative(dx, dy, ds)`, which respectively apply the derivative and its adjoint (at `(A, b, c)`) to their inputs and return the result.

3.2 Example

We consider differentiating a semidefinite program

$$\begin{array}{ll} \text{minimize} & \mathbf{tr}(C^T X) \\ \text{subject to} & X \succeq 0, \\ & \mathbf{tr}(A_i X) = b_i, \quad i = i, \ldots, p, \end{array} \tag{4}$$

with optimization variable $X \in \mathbf{S}^n_+$ and problem data $C \in \mathbf{S}^n$, $A_1, \ldots, A_p \in \mathbf{S}^n$, and $b \in \mathbf{R}^p$ ($\mathbf{S}^n$ denotes the set of symmetric matrices in $\mathbf{R}^{n \times n}$, and $\mathbf{S}^n_+$ denotes the set of symmetric positive semidefinite matrices in $\mathbf{R}^{n \times n}$). This problem can be readily cast as a standard cone program (1), where the cone is the Cartesian product of a zero cone and a semidefinite cone.

We generated a feasible, bounded random instance of (4) with $p = 100$ and $n = 300$. We used our Python package to solve this instance and retrieve the derivative and its adjoint. The solve, which calls into SCS, took about 195.7 seconds on a machine with a six-core Intel i7-8700K. Next, we computed the derivative of the optimal value of (4) with respect to each of the A_i by applying the adjoint of the derivative to the gradient of the objective. Applying the adjoint of the derivative took about 191.7 seconds, which is roughly equal to the time it took to solve the problem. Note that we calculated the derivative of the objective with respect to 4,515,000 elements (each of the A_i), which required the solution of a 90,401 $\times$ 90,401 linear system. This evidently would not have been practical had we not treated the linear mapping as an abstract operator.

Acknowledgements

We thank Brandon Amos and Zico Kolter, who concurrently and independently derived calculations similar to the ones in this paper, for many useful discussions. Shane Barratt is supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1656518. Walaa Moursi is partially supported by the Natural Science and Engineering Research Council of Canada Postdoctoral Fellowship.

References

[ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *Proc. USENIX Conference on Operating Systems Design and Implementation*, pages 265–283, 2016.

[AJS+18] B. Amos, I. Jimenez, J. Sacks, B. Boots, and Z. Kolter. Differentiable MPC for end-to-end planning and control. In *Proc. Advances in Neural Information Processing Systems*, pages 8299–8310, 2018.

[AK17] B. Amos and Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. Intl. Conf. on Machine Learning*, volume 70, pages 136–145, 2017.

[AMP+19] A. Agrawal, A. Modi, A. Passos, A. Lavoie, Ashish Agarwal, A. Shankar, I. Ganichev, J. Levenberg, M. Hong, R. Monga, and S. Cai. TensorFlow Eager: A multi-stage,

Python-embedded DSL for machine learning. In *Proc. Systems for Machine Learning*, 2019.

[AVDB18] 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.

[AWK17] A. Ali, E. Wong, and Z. Kolter. A semismooth Newton method for fast, generic convex programming. In *Proc. Intl. Conf. on Machine Learning*, pages 70–79, 2017.

[BBD+17] S. Boyd, E. Busseti, S. Diamond, R. Kahn, K. Koh, P. Nystrup, and J. Speth. Multi-period trading via convex optimization. *Foundations and Trends in Optimization*, 3(1):1–76, 2017.

[BEFB94] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994.

[BKSF59] L. M. Beda, L. N. Korolev, N. V. Sukkikh, and T. S. Frolova. Programs for automatic differentiation for the machine BESM. Technical Report, Institute for Precise Mechanics and Computation Techniques, Academy of Science, 1959.

[BMB18] E. Busseti, W. Moursi, and S. Boyd. Solution refinement at regular points of conic problems. *arXiv preprint arXiv:1811.02157*, 2018.

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[BS00] J. Bonnans and A. Shapiro. *Perturbation Analysis of Optimization Problems*. Springer Series in Operations Research. Springer-Verlag, New York, 2000.

[BT04] D. Bertsimas and A. Thiele. A robust optimization approach to supply chain management. In *Proc. Intl. Conf. on Integer Programming and Combinatorial Optimization*, pages 86–100. Springer, 2004.

[BTGNV05] A. Ben-Tal, B. Golany, A. Nemirovski, and J.-P. Vial. Retailer-supplier flexible commitments contracts: A robust optimization approach. *Manufacturing & Service Operations Management*, 7(3):248–271, 2005.

[BTN01] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization*. SIAM, 2001.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[DAK17] P. Donti, B. Amos, and Z. Kolter. Task-based end-to-end model learning in stochastic optimization. In *Proc. Advances in Neural Information Processing Systems*, pages 5484–5494, 2017.

[DB16a] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DB16b] S. Diamond and S. Boyd. Matrix-free convex optimization modeling. In *Optimization and its applications in control and data sciences*, volume 115 of *Springer Optim. Appl.*, pages 221–264. Springer, Cham, 2016.

[Din07] U. Dini. *Lezioni di Analisi Infinitesimale*, volume 1. Università di Pisa, 1907.

[DR09] A. Dontchev and R. Rockafellar. *Implicit Functions and Solution Mappings*. Springer, 2009.

[dSA+18] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and Z. Kolter. End-to-end differentiable physics for learning and control. In *Proc. Advances in Neural Information Processing Systems*, pages 7178–7189, 2018.

[FJL18] R. Frostig, M. Johnson, and C. Leary. Compiling machine learning programs via high-level tracing. In *Systems for Machine Learning*, 2018.

[FM68] A. Fiacco and G. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley and Sons, Inc., New York-London-Sydney, 1968.

[FNB19] A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 2019.

[GB14] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, 2014.

[Gri89] A. Griewank. On automatic differentiation. In M. Iri and K. Tanabe, editors, *Mathematical Programming*, pages 83–108. Kluwer Academic Publishers, 1989.

[GW08] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2008.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[Inn19] M. Innes. Don't unroll adjoint: Differentiating SSA-form programs. In *Proc. Advances in Neural Information Processing Systems, Workshop on Systems for ML and Open Source Software*, 2019.

[JOP+01] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

[Lei76] G.W. Leibniz. Calculus tangentium differentialis. In *The Early Mathematical Manuscripts of Leibniz*, chapter V, pages 124–127. Open Court, 1920/1676. Original manuscript dated 1676, translation by J. M. Child published in 1920.

[LFK18] C. Ling, F. Fang, and Z. Kolter. What game are we playing? End-to-end learning in normal and extensive form games. In *Proc. Intl. Joint Conf. on Artificial Intelligence*, pages 396–402, 2018.

[Löf04] J. Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proc. Computer-Aided Control System Design Conference*, 2004.

[Mar52] H. Markowitz. Portfolio selection. *Journal of Finance*, 7(1):77–91, 1952.

[MBBW19] N. Moehle, E. Busseti, S. Boyd, and M. Wytock. Dynamic energy management. *arXiv preprint arXiv:1903.06230*, 2019.

[MDA15] D. Maclaurin, D. Duvenaud, and R. Adams. Autograd: Effortless gradients in NumPy. In *Proc. Intl. Conf. on Machine Learning, AutoML Workshop*, 2015.

[Nem07] A. Nemirovski. Advances in convex optimization: Conic programming. In *International Congress of Mathematicians*, volume 1, pages 413–444. Eur. Math. Soc., Zürich, 2007.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior-point Polynomial Algorithms in Convex Programming*, volume 13 of *SIAM Studies in Applied Mathematics*. SIAM, 1994.

[OCPB16] 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, 2016.

[PGC+17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *Proc. Advances in Neural Information Processing Systems, Workshop on Automatic Differentiation*, 2017.

[PS82] C. Paige and M. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, 1982.

[RF10] O. Rodríguez and J. Fernandez. A semiotic reflection on the didactics of the chain rule. *The Mathematics Enthusiast*, 7(2):321–332, 2010.

[RHW88] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[Rob80] S. Robinson. Strongly regular generalized equations. *Mathematics of Operations Research*, 5(1):43–62, 1980.

[Spe80] B. Speelpenning. *Compiling fast partial derivatives of functions given by algorithms*. PhD thesis, University of Illinois, Urbana, Dept. of Computer Science, 1980.

[UMZ+14] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. *Super Computing Workshop on High Performance Technical Computing in Dynamic Languages*, 2014.

[VDWCV11] S. Van Der Walt, C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[Wen64] R. Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, 1964.

[YTM94] Y. Ye, M. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19(1):53–67, 1994.