

Disciplined Bilevel Programming

Hao Zhu^{1,2} and Joschka Boedecker^{1,2}

¹IMBIT//BrainLinks-BrainTools

²Department of Computer Science, University of Freiburg

September 24, 2026

Abstract

Bilevel optimization provides a natural modeling language for hierarchical decision problems. However, applying existing numerical solvers usually requires substantial manual analysis and reformulation. In this paper, we introduce *disciplined bilevel programming* (DBLP), a symbolic framework that allows users to specify and solve optimistic bilevel problems in a high-level, human-readable way that is close to the mathematical formulation. For problems with a disciplined nonlinear upper problem and a convex lower problem satisfying the disciplined parameterized programming rules, DBLP automatically canonicalizes the lower problem into conic form and constructs an equivalent single-level reformulation using the conic Karush-Kuhn-Tucker conditions. We relax the resulting complementarity constraint and use a gap continuation procedure to approximately solve a sequence of smooth nonlinear problems. We implement DBLP in the open-source Python package BLVPY, an extension of CVXPY for bilevel programming. We demonstrate the modeling and solution capabilities of BLVPY on a range of bilevel optimization problems from several application domains. The proposed framework and implementation allow users to specify and solve bilevel optimization problems within a few lines of code, without prior expertise in bilevel modeling and numerical optimization.

Contents

1	Introduction	3
1.1	Previous and related work	3
1.2	Contributions	5
1.3	Outline	6
2	Disciplined optimization	6
2.1	Disciplined convex programming	6
2.2	Problems involving parameters	7
2.3	Nonlinear optimization problems	8
3	Disciplined bilevel programming	9
3.1	Lower problem canonicalization	9
3.2	Standing assumptions	9
3.3	Conic optimality	10
3.4	Single-level reformulation	11
3.5	A simple example	12
4	Approximate optimality and continuation	14
4.1	Relaxed complementarity	14
4.2	Gap continuation	15
5	Implementation	15
5.1	The BLVPY pipeline	16
5.2	A simple code snippet	18
6	Examples	19
6.1	Ridge regression auto-tuning	19
6.2	Stackelberg security game	20
6.3	Renewable capacity planning	21
6.4	Traffic tolling	23
6.5	Planar truss design	27
6.6	Differentiable model predictive control	28
7	Discussion	32
7.1	Conclusion	32
7.2	Solution quality and guarantees	32
7.3	Lower-suboptimality and polishing	32
A	Proofs	34
A.1	Proof of proposition 4.1	34
A.2	Proof of corollary 4.2	34
B	Consistency of globally solved relaxations	35

1 Introduction

We consider *bilevel optimization problems* of the form

$$\begin{aligned} & \text{minimize} && F_0(x, y) \\ & \text{subject to} && F_i(x, y) \leq 0, \quad i = 1, \dots, m \\ & && y \in S(x), \end{aligned} \tag{1.1}$$

where for a fixed $x \in \mathbf{R}^n$, the constraint set $S(x)$ is defined as the solution set of the following *lower problem* (or *inner problem*):

$$\begin{aligned} S(x) = \operatorname{argmin}_z && f_0(x, z) \\ \text{subject to} && f_i(x, z) \leq 0, \quad i = 1, \dots, p. \end{aligned} \tag{1.2}$$

In this context, the problem (1.1) is referred to as the *upper problem* (or *outer problem*), with $x \in \mathbf{R}^n$ and $y \in \mathbf{R}^k$ denoting the *upper variable* and *lower variable*, respectively. Accordingly, the lower problem (1.2) is parameterized by the upper variable x . The functions $F_0: \mathbf{R}^n \times \mathbf{R}^k \rightarrow \mathbf{R}$ and $f_0: \mathbf{R}^n \times \mathbf{R}^k \rightarrow \mathbf{R}$ are the upper- and lower-level objective functions, respectively, while $F_i, i = 1, \dots, m$, and $f_i, i = 1, \dots, p$, are their corresponding constraint functions.

Following a common convention in bilevel optimization [SMD18], the generic formulations (1.1) and (1.2) display only inequality constraints; equality constraints are also permitted in practice, but are omitted here for notational brevity. Additionally, as shown in Shen *et al.* [SDGB16], an equality constraint $F(x, y) = 0$ may instead be expressed indirectly as the pair of inequality constraints

$$F(x, y) \leq 0 \quad \text{and} \quad -F(x, y) \leq 0.$$

We assume that the problem (1.1) has *optimistic* semantics, meaning that if the lower problem (1.2) has multiple optimal points (*i.e.*, the set $S(x)$ is nonsingleton), then the upper problem (1.1) chooses $y \in S(x)$ that is most favorable to the upper objective $F_0(x, y)$. In the most general case, the bilevel problem (1.1) is a nonconvex optimization problem that can be very difficult to solve, even if all involved functions F_i and f_i are convex.

1.1 Previous and related work

1.1.1 Hierarchical optimization

Hierarchical optimization originates in Stackelberg games [vS51] and was formulated as a mathematical program with an optimization problem in its constraints by Bracken and McGill [BM73]. Bilevel models arise naturally when one decision maker must anticipate another’s optimal response, with applications spanning economics, engineering, machine learning, and operations research. For example, in transportation network design and toll setting, an authority chooses capacities or prices at the upper level, while travelers select routes at the lower level, often according to a traffic equilibrium; see, *e.g.*, Bard [Bar98, chapter 10] and Colson *et al.* [CMS07]. In energy and other regulated markets, a planner, regulator, or producer makes investment and pricing decisions while anticipating market clearing or the responses of consumers and competing firms [DZ20, chapter 5]. In machine learning, the upper problem selects hyperparameters, regularization weights, or model

architecture to minimize validation loss, whereas the lower problem trains the model on the training data [Ped16, FFS⁺18, LVD20, BB21]; see also Dempe and Zemkoho [DZ20, chapter 6]. Just to mention a few.

The distinction between optimistic and *pessimistic* formulations of bilevel problems is essential when the lower solution mapping is set-valued: The former permits an upper-favorable lower optimizer, whereas the latter protects against an upper-adverse one. For more discussion about pessimistic bilevel problems, see Loridan and Morgan [LM96] and Dempe [Dem02].

There are many excellent books and surveys on bilevel optimization. Bard [Bar98] gives a practical treatment of continuous, discrete, convex, and general bilevel programs, together with algorithms and applications in transportation, production planning, and pricing. Dempe [Dem02] develops the theoretical foundations through parametric optimization, optimality conditions, stability and complexity analysis, and a detailed discussion of nonunique lower-level solutions. Colson *et al.* [CMS07] provide a concise survey of applications and solution methods and explain the connection between bilevel programs and mathematical programs with equilibrium constraints. Sinha *et al.* [SMD18] broaden the algorithmic review from classical methods to evolutionary approaches and discuss representative applications. The edited volume of Dempe and Zemkoho [DZ20] collects more recent developments in local, global, and heuristic algorithms, as well as extensions and applications including optimal control, energy markets, data analytics, and security.

1.1.2 Solution methods

In the most general case, finding an exact solution to a bilevel optimization problem of the form (1.1) is NP-hard [HJS92, SMD18], so practitioners usually rely on reformulations, approximations, or heuristic methods that do not guarantee global optimality to find an acceptable approximate solution. We discuss two widely used approaches in the following paragraphs. Descriptions of other methods, such as evolutionary algorithms and branch-and-bound methods for discrete bilevel programming, can be found in the literature listed above.

KKT single-level reformulations. When the lower problem (1.2) is convex and an appropriate constraint qualification holds, its Karush-Kuhn-Tucker (KKT) conditions replace lower optimality by primal feasibility, stationarity, dual feasibility, and complementarity. The resulting single-level problem is a mathematical program with *equilibrium* (or specifically, *complementarity*) constraints, and is exactly equivalent to the original bilevel problem (1.1). This classical connection and its stationarity theory are developed in [YZ95, LPR96, SS00, DD06, DKK06, DD12] (just to list a few). However, exact complementarity causes standard nonlinear programming constraint qualifications to fail, even when the original lower problem is regular. Regularization and smoothing methods therefore solve nearby nonlinear programs and drive a relaxation parameter to zero [FJQ99, Sch01] (see also §4 for more details).

Differentiable optimization. Modern machine learning often treats bilevel problems by differentiating the upper objective through the procedure used to solve the lower problem. One approach fixes a finite sequence of lower-level iterations and applies the chain rule through their update maps [MDA15, FFS⁺18]. Another assumes that the lower solution is locally unique and differentiates its first-order optimality equations using the implicit function

theorem [Ped16, LVD20]. Closely related work derives and implements derivatives of solutions to parameterized optimization problems [GFC⁺16, AK17, AAB⁺19, ABB⁺19, HNB26]. In both approaches, the derivative depends on a particular choice of lower-level solution, *i.e.*, either the result of a fixed number of algorithmic steps or a locally unique solution that varies smoothly with the upper variables. Moreover, most differentiable optimization methods mentioned above assume an unconstrained upper problem or only simple constraints on the upper variables, while general constraints involving the lower solution, as in the problem (1.1), are usually not supported.

1.1.3 Limitations

Despite the widely recognized applications and solution methods for bilevel optimization, a significant gap remains between natural problem formulations (*e.g.*, of the form (1.1)) and the *canonical* forms required by existing solution methods. For example, a KKT reformulation first requires verifying lower-level convexity and a suitable constraint qualification. It then introduces a dual variable and a complementarity condition for each lower-level inequality, which must be encoded in a form accepted by the chosen solver [DBS23, §2 and §4]. Carrying out these steps correctly requires substantial expertise and is often tedious and error-prone, even for experienced practitioners.

This gap has been partially addressed by different *domain specific languages* (DSLs) for bilevel optimization in different programming languages. The early Python package PAO [HC19] extends Pyomo [HWW11, BHH⁺21] to support the symbolic modeling and solving of multilevel optimization problems, but is no longer actively updated or maintained. The more recent Julia package BilevelJuMP.jl [DBS23] extends JuMP [DHL17, LDD⁺23] to support bilevel optimization with conic constraints and quadratic objectives in the lower problem, but it does not support nonlinear expressions at that level. As a result, users must still manually reformulate their problems into at least the canonical conic forms required by BilevelJuMP.jl, which is not ideal for users not well-versed in convex optimization.

1.2 Contributions

In this paper, we introduce *disciplined bilevel programming* (DBLP), a symbolic modeling framework for bilevel optimization problems that allows users to specify and solve bilevel problems in a natural, human-readable way that is very close to the original mathematical formulation (1.1). Given a compatible symbolic bilevel problem specification, DBLP allows automatic canonicalization of the problem into a standard single-level optimization problem that can be solved by existing numerical solvers for general nonlinear optimization. The associated DSL is implemented in the Python package BLVPY, which is fully open-source and available at

<https://github.com/dxogrp/blvpy>.

DBLP is based on the ideas of *disciplined optimization*, whose principles consist in allowing users to construct optimization problems from a set of basic functions and composition rules, while guaranteeing that the resulting problem is compatible with a specific class of solvers (see §2 for more discussion and references). We adapt the KKT reformulation of bilevel problems to the disciplined optimization framework, and propose a rewriting procedure that transforms a DBLP-compliant bilevel problem into a single-level optimization problem compatible with general nonlinear solvers. As a result, users can specify and solve

bilevel optimization problems in short scripts without the expert knowledge of mathematical analysis or optimization. As far as we know, there is no existing work that combines bilevel optimization with the disciplined optimization philosophy.

We should also note that the purpose of this work is *not* to propose a new solution method or theoretical results for bilevel optimization, but to fill the gap between the original mathematical problem formulation and the canonical forms required by a numerical solver, based on many existing results in the literature.

1.3 Outline

The rest of this paper is organized as follows. In §2, we provide a short review of disciplined optimization, focusing on disciplined convex, parameterized, and nonlinear programming. In §3, we introduce the DBLP framework and develop the lower problem canonicalization and exact single-level reformulation via the KKT conditions. In §4, we present the relaxed complementarity condition, its approximate lower-level optimality guarantees, and the gap continuation procedure for (approximately) solving the reformulated problem. The BLVPY implementation of the proposed DBLP framework and its basic usage are described in §5. Finally, in §6, we present numerical examples of applications appearing in different domains, followed by some discussion and comments in §7.

2 Disciplined optimization

In this section, we provide a short introduction to disciplined optimization and present some of its key concepts and results.

2.1 Disciplined convex programming

The original idea of disciplined optimization was proposed by Grant *et al.* [Gra05, GBY06] in the context of convex optimization, and is referred to as *disciplined convex programming* (DCP). The fundamental philosophy of DCP is that, rather than constructing objective and constraint functions without regard for convexity, users draw from a library of *atomic functions* with known convexity properties and combine them in ways which the convexity preserving operations insure convex outcomes.

Specifically, a DCP problem has the form

$$\begin{aligned} & \text{minimize/maximize} && f(x) \\ & \text{subject to} && p_i(x) \sim q_i(x), \quad i = 1, \dots, m, \end{aligned} \tag{2.1}$$

where $x \in \mathbf{R}^n$ is the variable, the function $f: \mathbf{R}^n \rightarrow \mathbf{R}$ is the objective, and $p_i, q_i: \mathbf{R}^n \rightarrow \mathbf{R}$ are the left-hand side and right-hand side functions of the i th constraint. The relational operator $\sim$ represents one of the relations $\leq$, $\geq$, or $=$. In DCP this problem must be convex, which imposes the following additional rules on the curvature of the involved functions in the problem (2.1) listed below.

- For a minimization problem, the objective function f must be convex; for a maximization problem, f must be concave.
- For a constraint of the form $p_i(x) \leq q_i(x)$, the function p_i must be convex, and q_i must be concave.

- For a constraint of the form $p_i(x) \geq q_i(x)$, the function p_i must be concave, and q_i must be convex.
- For a constraint of the form $p_i(x) = q_i(x)$, both p_i and q_i must be affine.

To verify the convexity of a DCP problem in the form (2.1), each function (or *expression*) in the problem is parsed into a *composition tree*, whose nodes represent functions with known convexity or convexity preserving operations (see, *e.g.*, [BV04, §3.2] and [ZB26, §2.4]). We (or a computer) can therefore verify the convexity of the objective and constraints constructively by traversing their composition trees from the leaves to the root and checking whether the resulting functions satisfy the curvature requirements listed above (see [ZB26, §A.4.3] for a specific example).

The DCP framework was originally implemented in the MATLAB package CVX [GB14]. It was later implemented by several other DSLs in different programming languages, including Convex.jl [UMZ⁺14] for Julia, CVXPY [DB16, AVDB18, ZNJ⁺26] for Python, CVXR [FNB20] for R, and CVXRust [Dia25] for Rust. Given a DCP-compliant convex program specified in any of these DSLs, one of the most important practical implications is that the problem can be automatically canonicalized. That is, a user only needs to specify the target problem in a high-level expression close to the mathematical formulation, and the DSL modeling system will automatically convert it into a standard form accepted by general convex conic solvers [AVDB18]. Finally, a compatible numerical solver, such as SCS [OCPB16], Clarabel [GC24], or MOSEK [MOS26], is called to solve the canonicalized problem.

Our work here extends the modeling language CVXPY to support bilevel programming. Although CVXPY has been extended over the years to support many types of nonconvex optimization problems, such as *stochastic* [AKDB15], *convex-concave* [SDGB16], *multi-convex* [SDU⁺17], *geometric* [ADB19], *quasiconvex* [AB20], and *saddle* [SLB24] optimization problems, to the best of our knowledge, disciplined modeling of bilevel optimization has not been implemented in any of these DSLs.

2.2 Problems involving parameters

One of the key extensions of the original DCP framework from Grant *et al.* [GBY06] and Diamond *et al.* [DB16] that is closely related to bilevel optimization is the concept of *disciplined parameterized programming* (DPP) [AAB⁺19]. DPP deals with convex optimization problems that involve *parameters*, which are variables whose values are fixed when the problem is solved, but can be changed between solves. For example, the lower problem (1.2) is a parameterized optimization problem with parameter $x \in \mathbf{R}^n$.

DPP imposes additional restrictions on the original DCP ruleset. Thus, every DPP-compliant problem is also DCP-compliant, but the converse does not hold. Specifically, DPP introduces two restrictions on the involved functions, or expressions, of a parameterized convex problem:

1. In DCP, all parameters themselves are classified as constants, while in DPP, parameters are classified as affine expressions.
2. In DCP, the product expression $\phi(u, v) = uv$ is affine if either u or v is constant (*i.e.*, does not depend on the variables), while in DPP, the product expression $\phi(u, v) = uv$ is affine if

- either subexpression u or v is constant (*i.e.*, depends on neither the parameters nor the variables), or
- one of the expressions u and v is *parameter-affine* (*i.e.*, does not involve variables and is affine in its parameters) and the other is *parameter-free* (*i.e.*, does not involve parameters).

The rules above describe the basic DPP restrictions, but they can be (and actually, have been) extended to cover additional combinations of expressions and parameters [ZNJ⁺26].

As DCP generates programs that are guaranteed to be convex, DPP generates convex programs that are additionally guaranteed to be reducible to *affine-solve-affine* (ASA) compatible canonical forms [AAB⁺19]. This means that, for a DPP-compliant problem, both the map from the original problem parameters to the canonical problem data and the map from the canonical solution back to the original solution are affine. Furthermore, the coefficients of these affine maps are constant, independent of the specific values of the problem parameters.

2.3 Nonlinear optimization problems

Disciplined nonlinear programming (DNLP) extends the atom-and-composition approach of DCP to nonlinear optimization problems that need not be convex [CZNB26]. It allows arbitrary smooth expressions and classifies certain nonsmooth expressions as *linearizable convex* or *linearizable concave* using composition rules analogous to those of DCP. Specifically, a DNLP-compliant problem satisfies the following rules:

- For a minimization problem, the objective must be linearizable convex; for a maximization problem, it must be linearizable concave.
- For a constraint of the form $p_i(x) \leq q_i(x)$, the function p_i must be linearizable convex, and q_i must be linearizable concave.
- For a constraint of the form $p_i(x) \geq q_i(x)$, the function p_i must be linearizable concave, and q_i must be linearizable convex.
- For a constraint of the form $p_i(x) = q_i(x)$, both p_i and q_i must be smooth.

A DNLP-compliant problem can be canonicalized into an equivalent smooth nonlinear program. During this process, admissible nonsmooth convex or concave subexpressions are replaced by auxiliary variables and appropriate epigraph or hypograph constraints, which are then expressed using smooth functions. For example, the epigraph constraint $t \geq |u|$ can be written as the pair of smooth inequalities $-t \leq u \leq t$. Finally, a canonicalized smooth nonlinear program is obtained, which can be solved by standard nonlinear solvers such as IPOPT [WB06], KNITRO [BNW06], COPT [GHW⁺22], and Uno [VL26, VM26].

This DCP-style canonicalization implemented in DNLP has several practical advantages. It allows nonsmooth modeling constructs to be handled by standard smooth nonlinear solvers, simplifies the construction of a valid initial point, and avoids reformulations that unnecessarily violate regularity conditions such as the linear independence constraint qualification. These features generally improve the robustness of the nonlinear optimization solution process and the quality of the resulting solution [CZNB26]. Unlike DCP, however, DNLP does not certify convexity or global optimality.

3 Disciplined bilevel programming

We now present the main ideas of DBLP, based on the concepts of disciplined optimization introduced in the previous section. We define a *disciplined bilevel program* as a bilevel optimization problem of the form (1.1) that satisfies the following rules:

- The objective and constraint functions $F_i: \mathbf{R}^n \times \mathbf{R}^k \rightarrow \mathbf{R}$ for $i = 0, 1, \dots, m$ of the upper problem (1.1) are DNLP-compliant with variables $x \in \mathbf{R}^n$ and $y \in \mathbf{R}^k$.
- The objective and constraint functions $f_i: \mathbf{R}^n \times \mathbf{R}^k \rightarrow \mathbf{R}$ for $i = 0, 1, \dots, p$ of the lower problem (1.2) are DPP-compliant with variable $z \in \mathbf{R}^k$, so that the lower problem is a disciplined convex program, parameterized by $x \in \mathbf{R}^n$.

These two rules allow us to conduct the following canonicalization of the bilevel problem (1.1) into a single-level optimization problem.

3.1 Lower problem canonicalization

Suppose the lower problem (1.2) is a DPP-compliant convex program with parameter $x \in \mathbf{R}^n$. Then this problem can be canonicalized into a standard conic form

$$\begin{aligned} & \text{minimize} && c(x)^T u + d(x) \\ & \text{subject to} && A(x)u + s = b(x) \\ & && s \in \mathcal{K}, \end{aligned} \tag{3.1}$$

where the vectors u and s are the variables, and $c(x)$, $d(x)$, $A(x)$, and $b(x)$ are the problem data. The cone $\mathcal{K}$ is a Cartesian product of proper convex cones. Additionally, by Agrawal *et al.* [AAB⁺19], the map from the original problem parameter x to the group of canonical problem data, given by

$$x \mapsto (c(x), d(x), A(x), b(x)),$$

is affine, with coefficients that are independent of the specific value of x . Note that the offset $d(x)$ in the objective of the canonical form (3.1) is a constant term which does not affect the optimal point of the lower problem (1.2). We include it here for notational simplicity in the following derivations.

3.2 Standing assumptions

Before proceeding with the canonicalization of the bilevel problem (1.1) via the conic formulation (3.1), we state the following convenient sufficient assumptions, which apply throughout the remainder of this paper:

- The admissible domain of the lower-level parameter x is compact. The functions F_i and f_i in (1.1), and c , d , A , and b in (3.1) are continuous. The set defined by the upper constraints of (1.1) is closed.
- The canonicalization is *lossless*, *i.e.*, it preserves both the optimal value and the complete solution set through a fixed affine recovery map ρ . In other words, let $p^*(x)$

and $q^*(x)$ be the optimal values of the original lower problem (1.2) and its canonical form (3.1), respectively. Then for every feasible x , we have

$$p^*(x) = q^*(x) \quad \text{and} \quad S(x) = \left\{ \rho(u) \left| \begin{array}{l} c(x)^T u + d(x) = q^*(x) \\ A(x)u + s = b(x) \\ s \in \mathcal{K} \end{array} \right. \right\}.$$

Moreover, every feasible (u, s) yields a feasible lower point $\rho(u)$ satisfying

$$f_0(x, \rho(u)) \leq c(x)^T u + d(x).$$

- For every feasible x , the lower problem has a finite attained optimum.
- For every feasible x , the canonical problem satisfies Slater's condition, *i.e.*, there exist $\tilde{u}$ and $\tilde{s}$ such that

$$A(x)\tilde{u} + \tilde{s} = b(x), \quad \tilde{s} \in \mathbf{relint} \mathcal{K},$$

where $\mathbf{relint} C$ denotes the relative interior of the set C .

Since $\mathcal{K}$ is a Cartesian product of closed convex cones, the last two assumptions imply strong duality and dual attainment for the canonical problem (3.1) [BV04, §5.9].

The aforementioned technical conditions are standard in parametric and bilevel optimization and are expected to hold for the intended applications [Dem02, chapter 4]. Our implementation (see §5) verifies the structural conditions that can be checked symbolically, including DPP and DNLP compliance and the use of a fixed, audited reduction path for lossless canonicalization [AVDB18, AAB⁺19, CZNB26, ZNJ⁺26]; the remaining analytic conditions, such as attainment and Slater's condition, are treated as user-asserted preconditions (which are, nevertheless, satisfied in most practical applications involving convex lower problems).

3.3 Conic optimality

We define the *dual cone* of $\mathcal{K}$ in the (primal) problem (3.1) as

$$\mathcal{K}^* = \{\lambda \mid \lambda^T s \geq 0 \text{ for all } s \in \mathcal{K}\}.$$

Then the dual problem of the canonical conic lower problem (3.1) is

$$\begin{aligned} & \text{maximize} && -b(x)^T \lambda + d(x) \\ & \text{subject to} && A(x)^T \lambda + c(x) = 0 \\ & && \lambda \in \mathcal{K}^*, \end{aligned}$$

where λ is the (dual) variable and x is the parameter (see, *e.g.*, [BV04, page 266]). According to the assumptions stated in §3.2, we have the following strong duality result: For every feasible x , a point y solves the lower problem if and only if there exist u , s , and λ satisfying

$$\begin{aligned} y &= \rho(u) \\ A(x)u + s &= b(x) \\ A(x)^T \lambda + c(x) &= 0 \\ s &\in \mathcal{K} \\ \lambda &\in \mathcal{K}^* \\ \lambda^T s &= 0. \end{aligned} \tag{3.2}$$

The first equation in (3.2) recovers the original lower variable y from the canonical variable u , while the remaining equations are the primal-feasibility, dual-feasibility, and complementarity conditions for the canonical lower problem (3.1) [OCPB16, §2.1]. In other words, the KKT conditions (3.2) are necessary and sufficient for optimality of the lower problem (1.2), and therefore, the set of lower optimal points $S(x)$ can be equivalently expressed as

$$S(x) = \left\{ \rho(u) \left| \begin{array}{l} A(x)u + s = b(x) \\ A(x)^T \lambda + c(x) = 0 \\ s \in \mathcal{K}, \quad \lambda \in \mathcal{K}^* \\ \lambda^T s = 0 \end{array} \right. \right\}.$$

3.4 Single-level reformulation

A direct implication of the KKT conditions (3.2) is that the bilevel problem (1.1) can be equivalently expressed as a single-level optimization problem with variables x , u , s , and λ . Specifically, the resulting problem is a nonlinear program with complementarity constraints [LPR96, SS00], given by

$$\begin{aligned} & \text{minimize} && F_0(x, \rho(u)) \\ & \text{subject to} && F_i(x, \rho(u)) \leq 0, \quad i = 1, \dots, m \\ & && A(x)u + s = b(x) \\ & && A(x)^T \lambda + c(x) = 0 \\ & && s \in \mathcal{K}, \quad \lambda \in \mathcal{K}^* \\ & && \lambda^T s = 0, \end{aligned} \tag{3.3}$$

where the problem variables are the original upper variable x , the canonical lower variable u , the slack variable s , and the dual variable λ . We use the standard term *lift* to describe an extended representation of a given optimization problem obtained by introducing auxiliary variables [GPT13]. Accordingly, a feasible point (x, u, s, λ) of the single-level reformulation (3.3) lifts the corresponding point (x, y) of the original bilevel problem (1.1), with $y = \rho(u)$. In this context, the single-level reformulation (3.3) is sometimes called a *lifted problem* of (1.1).

Recall that $c(x)$, $A(x)$, and $b(x)$ are affine in x , and that the solution recovery map ρ is affine. Consequently, the compositions $F_i(x, \rho(u))$ preserve the DNLP compliance (as well as the convexity) of the upper-level expressions, while the equality constraints introduced by the KKT reformulation are smooth. The cone-membership constraints remain convex in s and λ , while the complementarity equation $\lambda^T s = 0$ is bilinear and nonconvex. Thus, even when all F_i are convex, the single-level reformulation is generally nonconvex because of complementarity and the parameter-variable products in the primal- and dual-feasibility equations.

Nevertheless, according to the discussion above, if the original bilevel problem (1.1) is DBLP-compliant (according to the rules defined at the beginning of §3), the single-level reformulation (3.3) is DNLP-compliant, and hence can be canonicalized into a smooth nonlinear program that can be solved by standard nonlinear solvers.

3.5 A simple example

As a minimal example, consider the bilevel problem

$$\begin{aligned} & \text{minimize} && \|x - g\|_2^2 + \gamma \|y - h\|_2^2 \\ & \text{subject to} && -r\mathbf{1} \preceq x \preceq r\mathbf{1} \\ & && y \in S(x) \end{aligned} \tag{3.4}$$

with variables $x, y \in \mathbf{R}^n$, where $g, h \in \mathbf{R}^n$, $\gamma > 0$, and $r > 0$ are problem data, and

$$\begin{aligned} S(x) = \operatorname{argmin}_z & \|z - x\|_1 \\ \text{subject to} & z \succeq 0. \end{aligned} \tag{3.5}$$

All vector inequalities in this example are componentwise.

To rewrite the bilevel problem (3.4) into the single-level form (3.3), we first canonicalize the lower problem (3.5) into the conic form (3.1). Introducing an epigraph variable $t \in \mathbf{R}^n$, the lower problem can be written as

$$\begin{aligned} & \text{minimize} && \mathbf{1}^T t \\ & \text{subject to} && -t \preceq x - z \preceq t \\ & && z \succeq 0. \end{aligned}$$

Define the canonical variable u and slack variable s as

$$u = \begin{bmatrix} z \\ t \end{bmatrix} \in \mathbf{R}^{2n} \quad \text{and} \quad s = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \in \mathbf{R}^{3n}.$$

The problem then has the canonical conic form (3.1) with

$$c = \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}, \quad d(x) = 0, \quad A = \begin{bmatrix} I & -I \\ -I & -I \\ -I & 0 \end{bmatrix}, \quad b(x) = \begin{bmatrix} x \\ -x \\ 0 \end{bmatrix}, \quad \mathcal{K} = \mathbf{R}_+^{3n},$$

and the solution recovery map is the coordinate projection

$$\rho(u) = \begin{bmatrix} I & 0 \end{bmatrix} u = z.$$

Note that the canonical problem data and the recovery map are indeed affine in the parameter x and canonical variable u , respectively. Moreover, canonical feasibility is equivalent to $z \succeq 0$ and $t \succeq |z - x|$, and therefore $\mathbf{1}^T t \geq \|z - x\|_1$. Conversely, every lower-feasible z admits a canonical feasible lift with $t = |z - x|$, for which equality holds. Thus, the canonicalization preserves the optimal value and complete solution set under the recovery map $\rho(u) = z$, and is lossless in the sense of the second assumption in §3.2.

Since the nonnegative orthant is self-dual, partition the dual variable as

$$\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \in \mathbf{R}_+^{3n},$$

conformably with s . The dual of the canonical lower problem associated with (3.4) is then given by

$$\begin{aligned} & \text{maximize} && x^T(\lambda_2 - \lambda_1) \\ & \text{subject to} && \lambda_1 - \lambda_2 - \lambda_3 = 0 \\ & && -\lambda_1 - \lambda_2 + \mathbf{1} = 0 \\ & && \lambda_1, \lambda_2, \lambda_3 \succeq 0. \end{aligned}$$

The canonical lower problem is strictly feasible for every x : One may choose any $z \succ 0$ and then choose t sufficiently large. It also has a finite attained optimum. Consequently, strong duality holds and the lower-level condition $y \in S(x)$ is equivalent to the existence of $z, t, s_1, s_2, s_3, \lambda_1, \lambda_2, \lambda_3$ satisfying

$$\begin{aligned} y &= z \\ z - t + s_1 &= x \\ -z - t + s_2 &= -x \\ -z + s_3 &= 0 \\ \lambda_1 - \lambda_2 - \lambda_3 &= 0 \\ -\lambda_1 - \lambda_2 + \mathbf{1} &= 0 \\ s_1, s_2, s_3 &\succeq 0 \\ \lambda_1, \lambda_2, \lambda_3 &\succeq 0 \\ \lambda_1^T s_1 + \lambda_2^T s_2 + \lambda_3^T s_3 &= 0. \end{aligned} \tag{3.6}$$

The first equality in (3.6) identifies the recovered lower variable y with z . The next three equalities, together with $s_1, s_2, s_3 \succeq 0$, impose primal feasibility; the following two equalities, together with $\lambda_1, \lambda_2, \lambda_3 \succeq 0$, impose dual feasibility; and the final equality imposes complementarity. Because every term in the final sum is nonnegative, this scalar condition is equivalent to componentwise complementarity between each s_i and λ_i .

Finally, substituting $y = z$ and the remaining conditions in (3.6) for the lower-level constraint in (3.4) gives the single-level problem

$$\begin{aligned} & \text{minimize} && \|x - g\|_2^2 + \gamma \|z - h\|_2^2 \\ & \text{subject to} && -r\mathbf{1} \preceq x \preceq r\mathbf{1} \\ & && z - t + s_1 = x \\ & && -z - t + s_2 = -x \\ & && -z + s_3 = 0 \\ & && \lambda_1 - \lambda_2 - \lambda_3 = 0 \\ & && -\lambda_1 - \lambda_2 + \mathbf{1} = 0 \\ & && s \succeq 0, \quad \lambda \succeq 0 \\ & && \lambda^T s = 0, \end{aligned} \tag{3.7}$$

with variables $x, z, t \in \mathbf{R}^n$ and $s, \lambda \in \mathbf{R}^{3n}$, which is equivalent to the original bilevel problem (3.4). The single-level reformulation (3.7) of the original bilevel problem (3.4) is easily seen to be DNLP-compliant. From this example, we can also see that performing the canonicalization and KKT reformulation by hand is tedious and error-prone, even for a simple bilevel optimization problem like the one presented.

4 Approximate optimality and continuation

Although the single-level reformulation (3.3) is DNLP-compliant, its exact complementarity constraint gives rise to a degenerate feasible geometry for which standard nonlinear programming constraint qualifications generally fail. Consequently, directly applying off-the-shelf nonlinear solvers may lead to numerical difficulties and weakened convergence guarantees [SS00, Sch01]. As a result, we propose a continuation method that solves a sequence of approximate problems with relaxed complementarity constraints and gradually drives the relaxation parameter to zero.

4.1 Relaxed complementarity

We consider the following relaxation of the single-level reformulation (3.3) with a relaxation parameter $\epsilon > 0$:

$$\begin{aligned} & \text{minimize} && F_0(x, \rho(u)) \\ & \text{subject to} && F_i(x, \rho(u)) \leq 0, \quad i = 1, \dots, m \\ & && A(x)u + s = b(x) \\ & && A(x)^T \lambda + c(x) = 0 \\ & && s \in \mathcal{K}, \quad \lambda \in \mathcal{K}^* \\ & && \lambda^T s \leq \epsilon, \end{aligned} \tag{4.1}$$

where the problem variables are the same as in (3.3).

The relaxation parameter ϵ in the problem (4.1) has the following direct lower-level optimality interpretation.

Proposition 4.1 *Lower-level objective certificate.* Under the assumptions in §3.2, every feasible point of (4.1) satisfies

$$0 \leq f_0(x, \rho(u)) - p^*(x) \leq \lambda^T s \leq \epsilon, \tag{4.2}$$

where $f_0(x, \rho(u))$ is the lower-level objective evaluated at the recovered lower point $\rho(u)$, and $p^*(x)$ is the optimal value of the lower problem (1.2) for the given parameter x .

The proof of this proposition is provided in §A.1 of the appendix. This result shows that the actual complementarity gap $\lambda^T s$ is an a posteriori certificate of lower-level suboptimality, which may be sharper than the prescribed tolerance ϵ . Thus, every feasible point of (4.1) recovers a lower-feasible point that is at most ϵ -suboptimal, and driving ϵ to zero enforces exact lower-level optimality in objective value. Note that although proposition 4.1 controls lower-level objective suboptimality, it does not by itself guarantee either proximity of $\rho(u)$ to $S(x)$ or optimality with respect to the upper problem; strong convexity provides the former guarantee.

Corollary 4.2 *Strongly convex lower problem.* Fix a feasible x and suppose that $f_0(x, \cdot)$ is μ -strongly convex on the feasible set of the lower problem, where $\mu > 0$. Then $S(x) = \{y^*(x)\}$ is a singleton set, and every feasible point of (4.1) satisfies

$$\|\rho(u) - y^*(x)\|_2 \leq \sqrt{\frac{2\epsilon}{\mu}}, \tag{4.3}$$

where $y^*(x)$ is the unique lower minimizer for the given parameter x .

The proof of this corollary is provided in §A.2 of the appendix. Under the strong-convexity condition of corollary 4.2, objective accuracy also controls solution accuracy, with distance decreasing at the rate $O(\sqrt{\epsilon})$. In particular, to guarantee $\|\rho(u) - y^*(x)\|_2 \leq \delta$, it suffices to choose $\epsilon \leq \mu\delta^2/2$. These observations motivate the gap continuation procedure developed below.

4.2 Gap continuation

We consider the following algorithm for solving the bilevel problem (1.1) via a sequence of approximate problems (4.1) with decreasing relaxation parameters $\epsilon \rightarrow 0$.

Algorithm 1 GAP CONTINUATION.

given initial point $(x^{(0)}, u^{(0)}, s^{(0)}, \lambda^{(0)})$, initial relaxation parameter $\epsilon^{(0)} > 0$, and reduction factor $\beta \in (0, 1)$.

$k := 0$.

repeat

1. *Solution.* Approximately solve the relaxation (4.1) with $\epsilon = \epsilon^{(k)}$ using $(x^{(k)}, u^{(k)}, s^{(k)}, \lambda^{(k)})$ as the initial point, and denote the solution by $(x^{(k+1)}, u^{(k+1)}, s^{(k+1)}, \lambda^{(k+1)})$.
2. *Gap reduction.* $\epsilon^{(k+1)} := \beta\epsilon^{(k)}$.
3. $k := k + 1$.

until $\epsilon^{(k)}$ is sufficiently small.

Note that the algorithm uses the computed solution of each relaxation to initialize the next one. Because consecutive relaxations differ only in the complementarity tolerance, gradually reducing ϵ often provides an effective warm start in practice. Moreover, the initial point need not be feasible unless required by the selected nonlinear solver, although a feasible and well-scaled initial guess may improve numerical robustness. Implementation details of algorithm 1 regarding the initialization and other numerical aspects are discussed in §5.

For completeness, §B of the appendix establishes an idealized consistency result for globally solved relaxations. This result does not constitute a convergence guarantee for algorithm 1, whose nonlinear subproblems are generally solved only locally and approximately. More refined convergence analyses of complementarity regularization methods under additional constraint qualifications and second-order conditions can be found in, *e.g.*, Scholtes [Sch01] and Ralph and Wright [RW04].

5 Implementation

We implement the proposed DBLP framework in the Python package BLVPY serving as a CVXPY [DB16, AVDB18, ZNJ⁺26] extension for bilevel optimization, which is fully open-source and available at

<https://github.com/dxogrp/blvpy>.

5.1 The BLVPY pipeline

The solution pipeline of BLVPY can be roughly divided into four stages: audited canonicalization of the lower problem, construction of an initial lifted point, gap continuation through a DNLP solver, and independent verification of the returned numerical point. The current release supports DBLP-compliant optimistic bilevel problems whose lower problems admit lossless canonical conic representations over zero, nonnegative, second-order, exponential, and 3-dimensional power cones.

Audited atoms and canonicalization. BLVPY replaces each linked upper CVXPY variable by a CVXPY parameter in a copy of the lower problem and verifies that the resulting problem satisfies the DCP and DPP rules. DCP and DPP compliance alone, however, does not guarantee that CVXPY canonicalizes every modeled function without approximation or uses only cone blocks and reduction steps supported by BLVPY. It therefore audits both the source expression tree and the CVXPY reduction chain before extracting the affine maps from the upper variables to $A(x)$, $b(x)$, $c(x)$, and $d(x)$, as well as the affine recovery map ρ . Table 1 lists the lower-level atoms audited by the current BLVPY release.

Initialization. BLVPY preserves a user-specified value for each upper variable. Otherwise, letting L and U denote the possibly infinite lower and upper bounds on, say, the scalar variable x , respectively, it sets

$$x^{(0)} = \begin{cases} (L + U)/2, & L, U \in \mathbf{R} \\ L + 1, & L \in \mathbf{R}, \quad U = \infty \\ U - 1, & L = -\infty, \quad U \in \mathbf{R} \\ 0, & L = -\infty, \quad U = \infty. \end{cases}$$

Given this initialization, BLVPY first enforces the CVXPY attributes of the upper variables. If the upper constraints, together with any domain restrictions inherited from the parameterized lower problem, form a DCP set, BLVPY then attempts a least distance projection onto this set. After fixing the upper point, BLVPY solves the canonical lower conic problem to initialize u , s , and λ , and applies the solution recovery map ρ to initialize the original lower variables. If the resulting point fails the independent residual check, an optional restoration problem minimizes a common nonnegative radius that relaxes the lifted constraints and the complementarity gap constraint. The point is used for continuation only if the recomputed violations satisfy the prescribed feasibility tolerance. This initialization strategy turns out to work quite well in practice, but it is not guaranteed to succeed for all problems. When this happens, the user may need to provide a feasible initial point based on problem specific knowledge.

Continuation and local search. BLVPY assembles the lifted DNLP problem once and updates only the parameter ϵ , using every accepted point to warm-start the next nonlinear solve. The default schedule starts at $\epsilon^{(0)} = 10^{-1}$, contracts it by 0.1, and terminates at 10^{-6} ; the default feasibility tolerance is 10^{-7} . When a scheduled reduction from the last accepted value ϵ to a target value $\tilde{\epsilon}$ fails, BLVPY inserts the intermediate value

$$\epsilon_{\text{mid}} = \sqrt{\epsilon \tilde{\epsilon}}$$

Table 1 Lower-level atoms audited by the current BLVPY release. The last column summarizes the representation produced during canonicalization. LP denotes a linear program representation (*i.e.*, involving only zero and nonnegative cones). SOC, EXP, and P3D denote second-order, exponential, and 3-dimensional power cones, respectively. We denote the i th-largest entry of a vector z by $z_{[i]}$.

Atom	Definition	Representation
<i>Affine expressions</i>		
affine expression	$\phi(z) = Mz + q$	Affine
<i>Elementwise atoms</i>		
abs(z)	$\phi(z)_i = z_i $	LP
entr(z)	$\phi(z)_i = -z_i \log z_i$	EXP
exp(z)	$\phi(z)_i = e^{z_i}$	EXP
huber(z,M)	$\phi(z)_i = \begin{cases} z_i^2, & z_i \leq M \\ 2M z_i - M^2, & z_i > M \end{cases}$	SOC
kl_div(z,v)	$\phi(z, v)_i = z_i \log(z_i/v_i) - z_i + v_i$	EXP
log(z)	$\phi(z)_i = \log z_i$	EXP
log1p(z)	$\phi(z)_i = \log(1 + z_i)$	EXP
logistic(z)	$\phi(z)_i = \log(1 + e^{z_i})$	EXP
maximum(z,v)	$\phi(z, v)_i = \max\{z_i, v_i\}$	LP
minimum(z,v)	$\phi(z, v)_i = \min\{z_i, v_i\}$	LP
power(z,p)	$\phi(z)_i = z_i^p$	Affine/SOC/P3D
rel_entr(z,v)	$\phi(z, v)_i = z_i \log(z_i/v_i)$	EXP
xexp(z)	$\phi(z)_i = z_i e^{z_i}$	EXP/SOC
<i>Vector atoms</i>		
cummax(z)	$\phi(z)_i = \max\{z_1, \dots, z_i\}$	LP
dotsort(z,W)	$\phi(z, W) = \sum_i z_{[i]} W_{[i]}$	LP
geo_mean(z,p)	$\phi(z) = \prod_i z_i^{w_i}, w_i = p_i / \sum_j p_j$	SOC
log_sum_exp(z)	$\phi(z) = \log \sum_i e^{z_i}$	EXP
max(z)	$\phi(z) = \max_i z_i$	LP
min(z)	$\phi(z) = \min_i z_i$	LP
norm1(z)	$\phi(z) = \sum_i z_i $	LP
norm_inf(z)	$\phi(z) = \max_i z_i $	LP
pnorm(z,p)	$\phi(z) = \ z\ _p$	LP/SOC/P3D
sum_largest(z,k)	$\phi(z) = \sum_{i=1}^k z_{[i]}$	LP
<i>Quadratic atoms</i>		
quad_form(z,P)	$\phi(z) = z^T P z$	SOC
quad_over_lin(z,t)	$\phi(z) = \ z\ _2^2 / t$	SOC

and retries, with at most eight insertions between successive scheduled targets by default. BLVPY also allows a multistart procedure as a heuristic for finding better local solutions. In this setup, BLVPY runs N independent continuation paths from eligible randomized upper initializations and returns, among the points accepted at the prescribed final tolerance, the one with the smallest upper objective.

Residual verification and output. BLVPY does not accept a point solely on the basis of the status reported by the nonlinear solver. After every nonlinear solve, it independently recomputes the primal- and dual-equality residuals, recovery-map residual, upper-constraint violation, distances to the primal and dual cones, complementarity product, and relaxed-gap violation. An attempted point is accepted only if the solver reports a solution and the recomputed violations satisfy the requested tolerance. The returned result contains immutable snapshots of the source and canonical variables, the accepted and attempted ϵ histories, objectives, solver statuses, residuals, and all multistart runs. An optional diagnostic solves the lower-level conic problem once more with the upper variable fixed at its returned value and reports the difference between the returned lower objective and the resulting optimal value. IPOPT [WB06] is the tested default DNLP backend, while Clarabel [GC24] is the default conic backend for initialization, projection, and diagnostics.

5.2 A simple code snippet

As a basic coding example, the following snippet shows how to specify and solve the bilevel problem defined in (3.4) and (3.5) using BLVPY. More complex practical examples are presented later in §6.

```

1 import cvxpy as cp
2 import blvpy as bp # import the BLVPY package
3
4 # define problem variables
5 x = cp.Variable(n)
6 y = cp.Variable(n)
7
8 # specify the lower problem
9 lower = bp.LowerProblem(
10     cp.Minimize(cp.norm1(y - x)), # lower objective
11     [y >= 0], # lower constraints
12     parameters=[x], # lower parameters
13 )
14
15 # specify the upper problem
16 obj = cp.Minimize(
17     cp.sum_squares(x - g)
18     + gamma * cp.sum_squares(y - h)
19 ) # upper objective
20 constr = [-r <= x, x <= r] # upper constraints
21 prob = bp.BilevelProblem(obj, lower, constr)
22
23 # solve the bilevel problem
24 result = prob.solve()

```

Note that the `BilevelProblem` and `LowerProblem` classes share the same CVXPY variables $\mathbf{x}$ and $\mathbf{y}$, while the argument `parameters=[x]` tells BLVPY to treat the upper variable x as fixed data in the lower problem. The call to `prob.solve()` performs validation, canonicalization, and gap continuation. The default solve uses a single deterministic initialization. To request N independent continuation runs from randomized upper-level initializations, the user can set, for example, `x.sample_bounds = (-r, r)` and pass `best_of=N` to the `solve()` method.

6 Examples

In this section, we present several examples of bilevel optimization problems from different application domains. These examples are simplified and are by no means exhaustive. Our goal here is to illustrate the basic modeling capabilities of the proposed DBLP framework and the BLVPY package. Interested readers may refer to

<https://github.com/dxogrp/blvpy/tree/main/examples>

for more examples not covered in this paper. All presented example bilevel problems were solved with the target complementarity gap $\epsilon = 10^{-9}$, and all returned final solutions passed the validation checks and diagnostics described in §5; see also the example repository for full code scripts, synthetic dataset setups, and detailed numerical results.

6.1 Ridge regression auto-tuning

We consider the problem of automatically tuning the regularization parameter in *ridge regression* [HK70, BB21]. Let $X_{\text{tr}} \in \mathbf{R}^{m_{\text{tr}} \times n}$ and $y_{\text{tr}} \in \mathbf{R}^{m_{\text{tr}}}$ denote the training data, and let $X_{\text{val}} \in \mathbf{R}^{m_{\text{val}} \times n}$ and $y_{\text{val}} \in \mathbf{R}^{m_{\text{val}}}$ denote the validation data. The hyperparameter tuning problem of ridge regression can be formulated as

$$\begin{aligned} & \text{minimize} && (1/m_{\text{val}}) \|X_{\text{val}} w - y_{\text{val}}\|_2^2 \\ & \text{subject to} && 10^{-4} \leq \lambda \leq 10 \\ & && w \in S(\lambda), \end{aligned} \tag{6.1}$$

where

$$S(\lambda) = \operatorname{argmin}_w (1/m_{\text{tr}}) \|X_{\text{tr}} w - y_{\text{tr}}\|_2^2 + \lambda \|w\|_2^2. \tag{6.2}$$

The bilevel problem (6.1) has upper variable $\lambda \in \mathbf{R}_+$ and lower variable $w \in \mathbf{R}^n$, which correspond to the regularization parameter and the regression coefficients, respectively. We can interpret the lower problem (6.2) as a ridge regression problem with a fixed regularization parameter λ , and the upper problem (6.1) as a (constrained) least squares problem that tunes λ to minimize the validation error.

To specify the bilevel problem (6.1) in BLVPY, we can use the following code snippet:

```
1 # problem variables
2 lbd = cp.Variable(nonneg=True)
3 w = cp.Variable(n)
4
5 # lower problem
6 tr_loss = cp.sum_squares(X_tr @ w - y_tr) / m_tr
7 lower = bp.LowerProblem(
8     cp.Minimize(tr_loss + lbd * cp.sum_squares(w)),
```

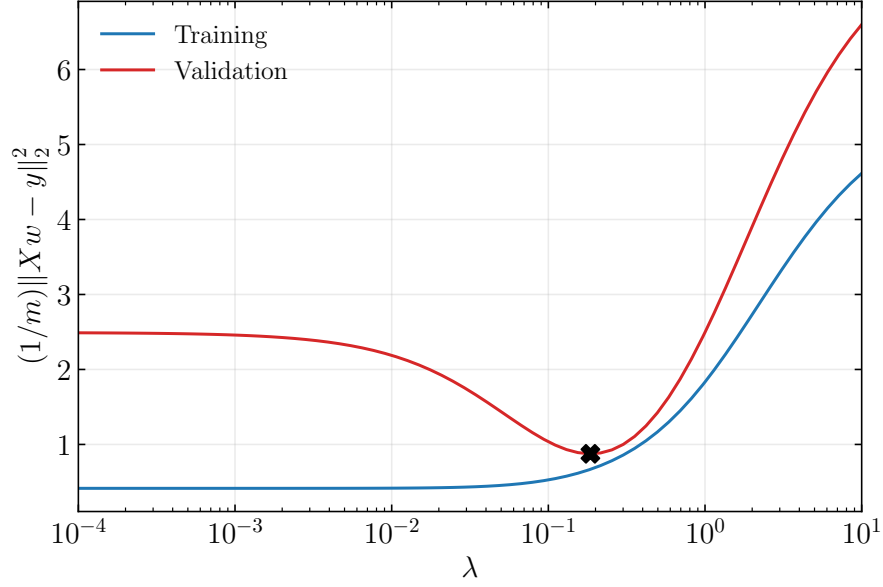


Figure 1 *Ridge regression auto-tuning.* Training and validation losses as functions of the regularization parameter λ . The black cross indicates the final solution of the problem (6.1) returned by BLVPY. Note that the subscripts are removed from the axes labels for simplicity.

```

9     parameters=[lbd],
10 )
11
12 # upper problem
13 val_loss = cp.sum_squares(X_val @ w - y_val) / m_val
14 prob = bp.BilevelProblem(
15     cp.Minimize(val_loss),
16     lower,
17     [lbd >= 1e-4, lbd <= 10],
18 )

```

We solve the problem (6.1) with BLVPY on a randomly generated dataset. Figure 1 shows the training and validation losses as functions of the regularization parameter λ . The figure shows that the final solution returned by BLVPY (shown as a black cross) corresponds to a regularization parameter that approximately achieves the minimum validation loss.

6.2 Stackelberg security game

We consider a Stackelberg security game in which a defender allocates limited security resources to protect a set of targets against an attacker [SAY⁺12, GGF⁺19]. Specifically, suppose there are n targets, and let $p \in \mathbf{R}^n$ with $0 \preceq p \preceq \mathbf{1}$ denote the probability that the defender protects each target, and let $q \in \mathbf{R}^n$ with $0 \preceq q \preceq \mathbf{1}$ denote the probability that the attacker attacks each target. For a fixed defender allocation p , the attacker chooses a

strategy q to solve the lower problem

$$\begin{aligned} S(p) = \operatorname{argmin}_q \quad & -\sum_{i=1}^n (r_i - c_i p_i) q_i \\ \text{subject to} \quad & q \succeq 0, \quad \mathbf{1}^T q = 1, \end{aligned} \quad (6.3)$$

where $r, c \in \mathbf{R}^n$ are given attacker reward and defender coverage effectiveness vectors, respectively, so that $r_i - c_i p_i$ is the attacker's payoff for the i th target after observing defender coverage. The defender then adapts its protection strategy to solve the upper problem

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^n L_i (1 - p_i) q_i + \gamma \|p\|_2^2 \\ \text{subject to} \quad & 0 \preceq p \preceq \mathbf{1}, \quad \mathbf{1}^T p \leq B \\ & q \in S(p), \end{aligned} \quad (6.4)$$

where p, q are the upper and lower variables, respectively, $L \in \mathbf{R}^n$ is a given (uncovered) defender loss vector under attack, $\gamma > 0$ is a given penalty parameter, and $B > 0$ is a given budget for the defender. Note that the lower program (6.3) can have multiple optimal attack targets and the optimistic bilevel semantics considered here is consistent with the strong-Stackelberg convention, *i.e.*, if $S(p)$ contains multiple attacker best responses, the defender may select the one that minimizes its upper-level objective in (6.4).

To specify the problem (6.4) in BLVPY, we can use the following code snippet:

```

1 # problem variables
2 p = cp.Variable(n, nonneg=True)
3 q = cp.Variable(n, nonneg=True)
4
5 # lower problem
6 utility = r - cp.multiply(c, p)
7 lower = bp.LowerProblem(
8     cp.Minimize(-utility @ q),
9     [cp.sum(q) == 1],
10    parameters=[p],
11 )
12
13 # upper problem
14 loss = cp.multiply(L, 1.0 - p) @ q
15 prob = bp.BilevelProblem(
16     cp.Minimize(loss + gamma * cp.sum_squares(p)),
17     lower,
18     [p <= 1, cp.sum(p) <= B],
19 )

```

Figure 2 shows the optimized defender allocation and the corresponding attacker response returned by BLVPY for a randomly generated instance of the problem (6.4). For reference, we also plot the allocations with no and uniform defender coverage and the corresponding attacker responses. The optimized defender allocation returned by BLVPY reduces the expected defender loss by approximately 80% compared with uniform coverage and by approximately 90% compared with no coverage.

6.3 Renewable capacity planning

We consider the problem of renewable capacity planning in a power system [ZZFC17], where a generation planner must choose renewable capacity before an electricity dispatcher sees the

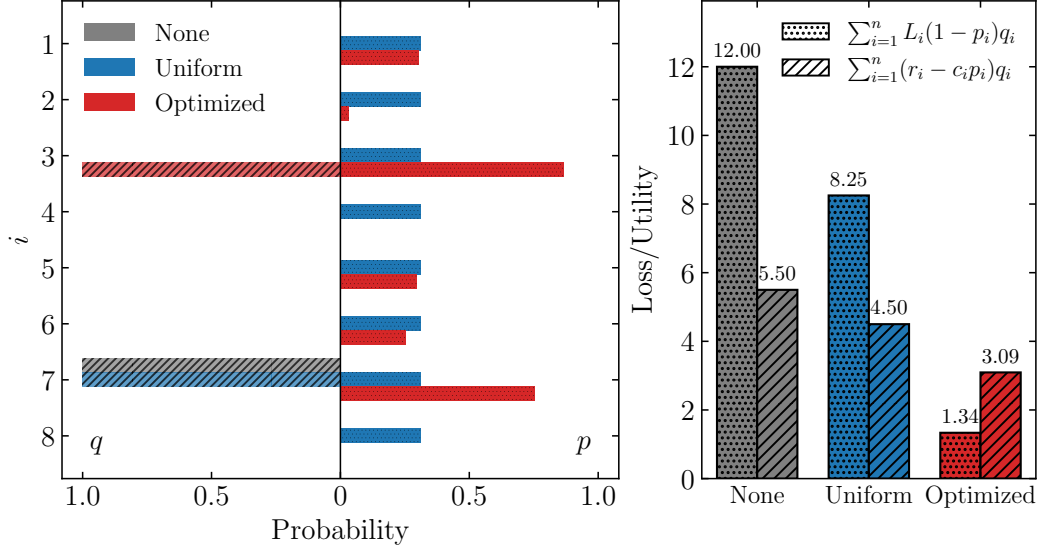


Figure 2 *Stackelberg security game.* *Left.* Defender protection p and attacker response q under none, uniform, and optimized protection. *Right.* The corresponding expected defender loss and attacker utility. All numerical quantities are reported in normalized units.

resulting operating limits and optimizes the dispatch accordingly. Let $\alpha \in \mathbf{R}_+$ denote the renewable capacity, and $L \in \mathbf{R}_+^m$ denote the renewable generation availability at m different time periods, so the vector $\alpha L \in \mathbf{R}_+^m$ represents the renewable generation limits at all time periods. Let $D \in \mathbf{R}_+^m$ denote the electricity demand at all time periods, and $r, g \in \mathbf{R}_+^m$ denote the renewable and thermal generation dispatch at all time periods. A generation planner may formulate the renewable capacity planning problem as

$$\begin{aligned}
& \text{minimize} && p_r \alpha^2 + p_g \mathbf{1}^T g \\
& \text{subject to} && 0 \leq \alpha \leq \alpha_{\max} \\
& && (r, g) \in S(\alpha),
\end{aligned} \tag{6.5}$$

where a grid dispatcher solves the lower problem

$$\begin{aligned}
S(\alpha) = \operatorname{argmin}_{r, g} && c_r \mathbf{1}^T r + c_g \mathbf{1}^T g + \delta(\|r\|_2^2 + \|g\|_2^2) \\
&& \text{subject to} && 0 \preceq r \preceq \alpha L, \quad g \succeq 0 \\
&& && r + g \succeq D.
\end{aligned}$$

The problem (6.5) has upper variable α and lower variables r, g . The planning and dispatching cost coefficients p_r, p_g, c_r, c_g , the maximum renewable capacity $\alpha_{\max}$, the regularization parameter $\delta > 0$, as well as the vectors L, D are assumed to be given.

To specify the bilevel problem (6.5) in BLVPY, we can use the following code snippet:

```

1 # problem variables
2 alpha = cp.Variable()
3 r = cp.Variable(m)

```

```

4 g = cp.Variable(m)
5
6 # lower problem
7 lower = bp.LowerProblem(
8     cp.Minimize(
9         c_r * cp.sum(r) + c_g * cp.sum(g)
10        + delta * (cp.sum_squares(r) + cp.sum_squares(g))),
11    [
12        r >= 0.0,
13        r <= cp.multiply(L, alpha),
14        g >= 0.0,
15        r + g >= D,
16    ],
17    parameters=[alpha],
18 )
19
20 # upper problem
21 prob = bp.BilevelProblem(
22     cp.Minimize(p_r * cp.square(alpha) + p_g * cp.sum(g)),
23     lower,
24     [alpha >= 0.0, alpha <= alpha_max],
25 )

```

We solve the problem (6.5) with BLVPY on a synthetic dataset, and the final solution returned by BLVPY is shown in figure 3. The lower plot indicates that the BLVPY solution (shown as a red dashed line) approximately minimizes the total renewable and thermal planning cost, while the upper plot shows the corresponding optimal dispatch under the given demand and renewable limit.

6.4 Traffic tolling

We consider second-best congestion pricing on a Braess network [BNW05, Ver02]. Specifically, consider a network with four nodes O, A, B, D , in which travelers move from the origin O to the destination D . As illustrated in figure 4, the ordered edge and path lists are

$$E = (OA, OB, AD, BD, AB) \quad \text{and} \quad P = (OAD, OBD, OABD),$$

respectively. The authority may impose a nonnegative toll $\tau \in \mathbf{R}_+$ on the central edge AB , which is used only by the third path. Let $x \in \mathbf{R}^3$ collect the path flows in the order specified by P , and let $f = (f_{OA}, f_{OB}, f_{AD}, f_{BD}, f_{AB}) \in \mathbf{R}^5$ collect the edge flows in the order specified by E . The path-edge incidence relation is given by

$$f = Hx, \quad H = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$

where $H_{ei} = 1$ if edge e belongs to path i , and $H_{ei} = 0$ otherwise. In particular, the last row gives $f_{AB} = x_3$. For a total travel demand $q \in \mathbf{R}_+$, feasible path flows satisfy $x \succeq 0$ and $\mathbf{1}^T x = q$, which also implies $f = Hx \succeq 0$.

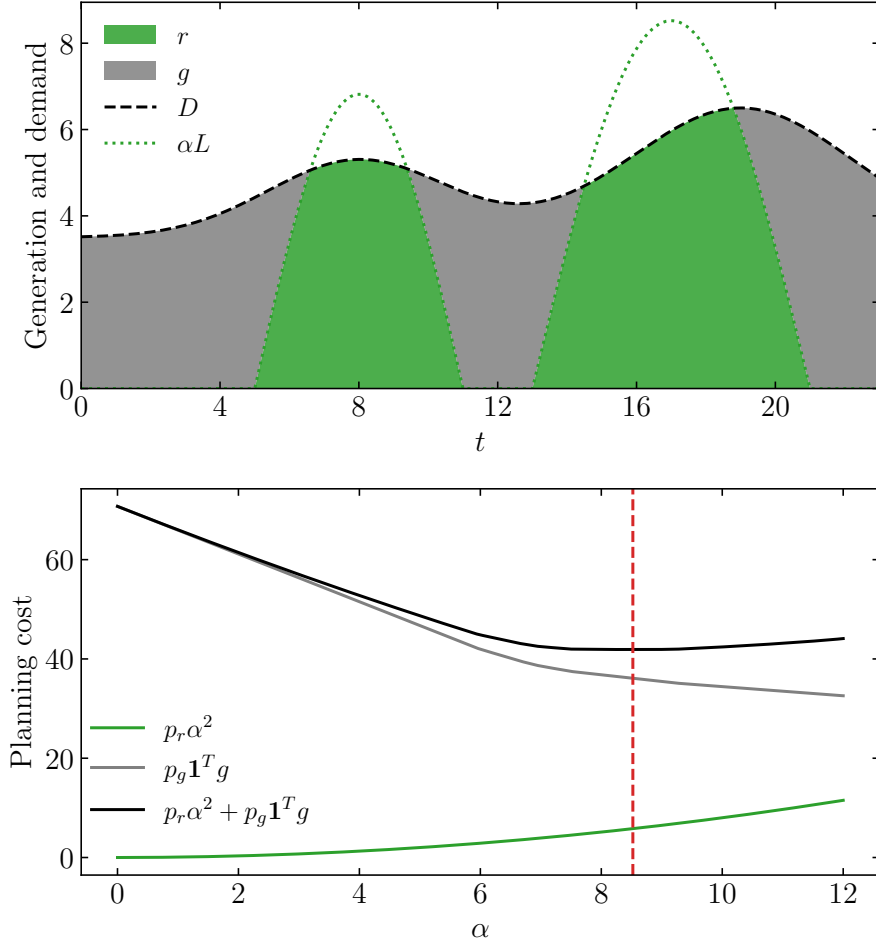


Figure 3 *Renewable capacity planning.* *Top.* Optimal renewable r and thermal g dispatch under demand D and the optimal renewable limit αL obtained from the BLVPY solution. *Bottom.* Investment and thermal generation costs versus capacity, with the dashed line marking the final solution of the problem (6.5) returned by BLVPY.

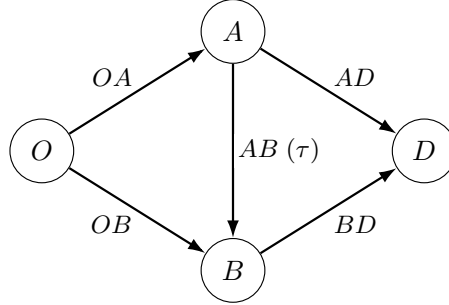


Figure 4 Braess network used in the traffic-tolling example. The arrows indicate the permitted travel directions, and τ denotes the toll imposed on the central edge AB .

Let $e \in E$ denote an edge in the network, and let f_e denote the flow on that edge. We assume that the travel time on edge e is an affine function of the flow f_e , given by

$$T_e(f_e) = b_e + a_e f_e,$$

where $a_e, b_e \geq 0$ are given edge-specific parameters. For a toll τ imposed on the central edge AB , we assume that travelers choose their paths according to the Wardrop equilibrium [War52], which can be formulated as the following lower problem:

$$\begin{aligned} S(\tau) = \operatorname{argmin}_x \quad & \sum_{e \in E} (b_e f_e + (1/2) a_e f_e^2) + \tau f_{AB} \\ \text{subject to} \quad & x \succeq 0, \quad \mathbf{1}^T x = q \\ & f = Hx, \end{aligned}$$

which is sometimes called the Beckmann potential formulation [BMW56, §3.1.2] of the Wardrop equilibrium. The authority's objective is to minimize the total travel time of all travelers plus a penalty term for the toll, which can be expressed as the upper problem

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} f_e T_e(f_e) + \rho \tau^2 \\ \text{subject to} \quad & 0 \leq \tau \leq \tau_{\max} \\ & x \in S(\tau), \end{aligned} \tag{6.6}$$

where x is the lower variable, τ is the upper variable, and $\rho > 0$ is a given penalty parameter.

To specify the problem (6.6) in BLVPY, we can use the following code snippet:

```

1 # problem variables
2 tau = cp.Variable(1)
3 x = cp.Variable(3)
4 f = H @ x
5
6 # lower problem
7 lower = bp.LowerProblem(
8     cp.Minimize(0.5 * a @ cp.square(f) + b @ f + tau[0] * x[2]),
9     [x >= 0, cp.sum(x) == q],
10    parameters=[tau],
11 )

```

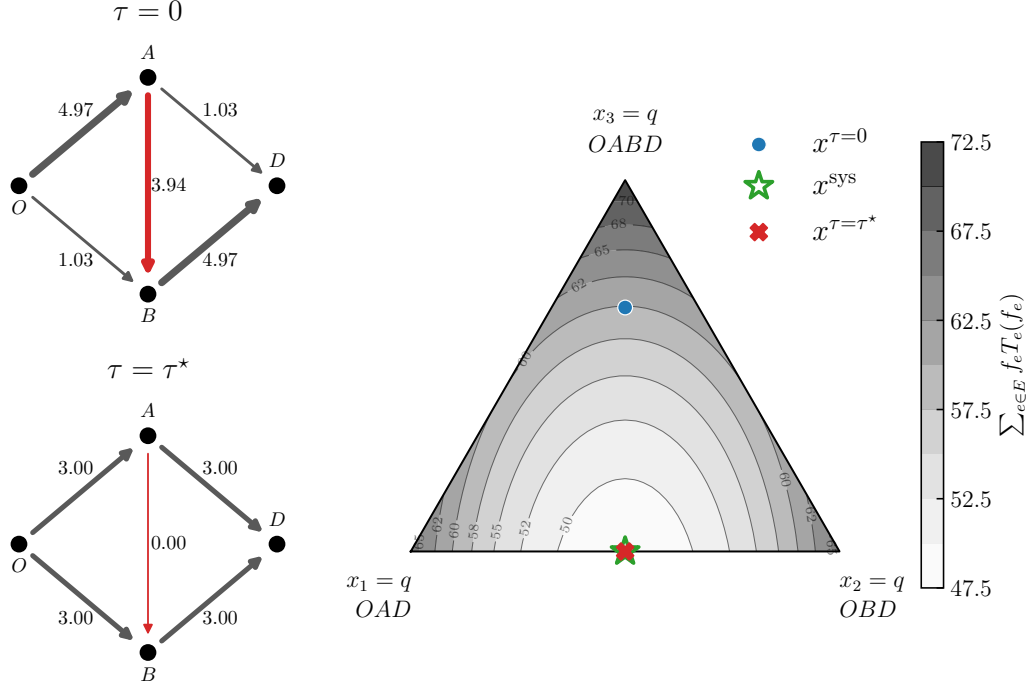


Figure 5 Traffic tolling on a Braess network. *Left.* Edge flows in untolled ($\tau = 0$) and optimally tolled ($\tau = \tau^*$) Braess networks returned by BLVPY. *Right.* Total travel time over feasible path flow allocations, where x^{sys} denotes the system optimal flow obtained by directly minimizing the total travel time assuming a central controller could assign routes to travelers. All numerical quantities are reported in normalized units.

```

12
13 # upper problem
14 time = b @ f + a @ cp.square(f)
15 prob = bp.BilevelProblem(
16     cp.Minimize(time + rho * cp.sum_squares(tau)),
17     lower,
18     [tau >= 0, tau <= tau_max],
19 )

```

We solve the problem (6.6) with BLVPY on a synthetic dataset. The two network diagrams on the left of figure 5 compare the edge flows under the optimal toll $\tau^* = 2.03$ returned by BLVPY for the problem (6.6), with those under no toll (*i.e.*, $\tau = 0$). With $\tau = 0$, approximately 3.94 flow units use the central path $OABD$, which therefore raises network travel time to 59.89 units. The selected toll $\tau^* = 2.03$ removes almost all flow from AB and divides demand between OAD and OBD . As a result, network travel time falls by approximately 20% to 48.18 units. This is essentially the system optimal flow, which is achieved by directly minimizing the total travel time (*i.e.*, the objective of the upper problem (6.6)), as if a central controller could directly assign routes to travelers. The right panel in

figure 5 shows total network travel time over every feasible path flow allocation satisfying $x_1 + x_2 + x_3 = q$. For the untolled flow $x^{\tau=0}$ (shown as a blue dot), the Wardrop equilibrium is not system optimal, while the optimal toll flow $x^{\tau=\tau^*}$ returned by BLVPY (shown as a red cross) is very close to the system optimal flow x^{sys} (shown as a green star).

6.5 Planar truss design

We consider the problem of designing a planar truss structure (see, *e.g.*, [CK09, chapter 5] and [TAA17, chapters 2 and 11]). Let $a \in \mathbf{R}_+^n$ and $L \in \mathbf{R}_+^n$ denote the cross-sectional areas and lengths of the n truss members, respectively, and let $E \in \mathbf{R}$ denote the Young's modulus of the truss material. Let $u \in \mathbf{R}^m$ denote the nodal displacements for a given set of applied nodal forces $f \in \mathbf{R}^m$, where m represents the number of displacement degrees of freedom. The designer's upper problem is to minimize the compliance $f^T u$ of the truss structure, subject to a budget constraint on the total material volume, *i.e.*,

$$\begin{aligned} & \text{minimize} && f^T u \\ & \text{subject to} && a_{\min} \mathbf{1} \preceq a \preceq a_{\max} \mathbf{1} \\ & && L^T a \leq V_{\max} \\ & && u \in S(a), \end{aligned} \tag{6.7}$$

where $a_{\min}, a_{\max} > 0$ are given lower and upper bounds on the cross-sectional areas, and $V_{\max} > 0$ is a given maximum material volume. We assume that the length vector $L > 0$ is given and fixed, so the problem (6.7) has upper variable a and lower variable u . The lower problem is the linear elasticity problem that computes the nodal displacements u for a given cross-sectional area allocation a , which can be expressed as

$$\begin{aligned} S(a) = \operatorname{argmin}_u & \quad (1/2)u^T K(a)u - f^T u \\ \text{subject to} & \quad u_i = 0, \quad i \in \mathcal{F}, \end{aligned}$$

where $\mathcal{F}$ is the set of fixed displacement indices, and

$$K(a) = B^T \operatorname{diag} \left(\frac{Ea_1}{L_1}, \dots, \frac{Ea_n}{L_n} \right) B$$

is the stiffness matrix of the truss structure, where $B \in \mathbf{R}^{n \times m}$ is the compatibility matrix that maps nodal displacements to member elongations.

Suppose a structural designer needs to allocate material among the members of a two-bay cantilever truss, as shown in figure 6. The truss has 6 nodes and $n = 10$ members, and the number of displacement degrees of freedom is $m = 12$ (since each node has 2 displacement degrees of freedom for the horizontal and vertical directions). Both degrees of freedom at each of the two leftmost nodes are fixed, and a unit downward load is applied at the lower-right node and a smaller downward load at the upper-right node (shown by the arrows). The following code snippet shows how to specify the corresponding bilevel problem (6.7) in BLVPY for this example.

```

1 # problem variables
2 a = cp.Variable(n, nonneg=True)
3 u = cp.Variable(m)
4
```

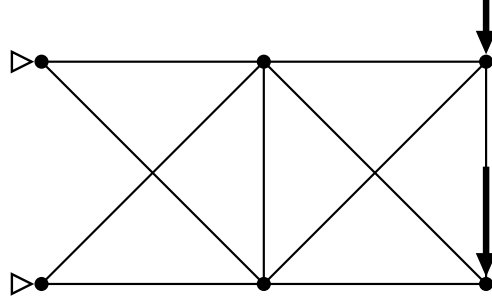


Figure 6 *Two-bay cantilever truss*. The circles and solid lines represent the nodes and members of the truss, respectively. The arrows indicate the applied nodal forces, and the leftmost triangles indicate the supported nodes with fixed displacements.

```

5 # lower problem
6 energy = 0.5 * cp.sum(cp.multiply(
7     cp.multiply(E / L, a), cp.square(B @ u)
8 ))
9 lower = bp.LowerProblem(
10     cp.Minimize(energy - f @ u),
11     [u[F] == 0],
12     parameters=[a],
13 )
14
15 # upper problem
16 problem = bp.BilevelProblem(
17     cp.Minimize(f @ u),
18     lower,
19     [L @ a <= V_max, a_min <= a, a <= a_max],
20 )

```

Figure 7 shows the optimized cross-sectional areas returned by BLVPY for the problem (6.7) with synthetic numerical problem data. A thicker line indicates a larger cross-sectional area, and the color of each member indicates the magnitude of the axial stress in that member under the optimized design. We also show the uniform design with the same total material volume for comparison. The figure shows that the optimized design returned by BLVPY allocates more material to members under higher stress. The compliances of the uniform and optimized designs are 1.90 and 1.35 (normalized units), respectively, so the optimized design reduces compliance by approximately 30% without using additional material.

6.6 Differentiable model predictive control

We consider the problem of learning model predictive control (MPC) regularization weights from expert demonstrations for a direct-current motor speed control task [AJS⁺18, RMD26]. Specifically, the controller problem is a finite-horizon MPC problem that computes the optimal control actions for given regularization weights, and the learner problem is an imitation learning problem that tunes the regularization weights to minimize the discrepancy between the MPC output and expert demonstrations. Conventionally, such a hierarchical

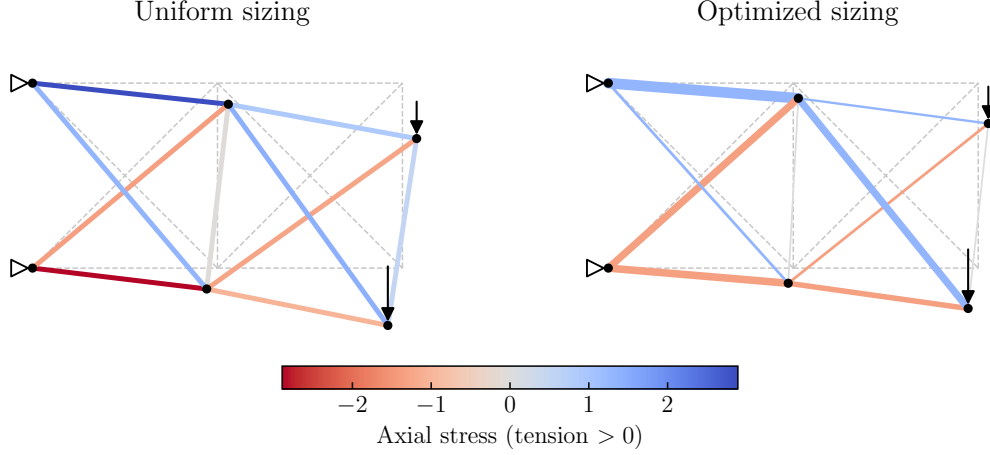


Figure 7 *Planar truss design.* Uniform baseline (shown left) and optimized (shown right) cross-sectional area designs of the two-bay cantilever truss in figure 6. The dashed lines indicate the undeformed geometry of the truss, while the solid lines indicate the deformed geometry under the applied loads. The color of each member indicates the magnitude of the axial stress in that member under equilibrium.

MPC problem is solved by differentiating through the MPC controller problem (see, *e.g.*, [AJS⁺18] and [ABB21]). Here we show that it can also be formulated as a bilevel problem and solved with BLVPY.

Let $x(t) = (\omega(t), i(t)) \in \mathbf{R}^2$ denote the state of the motor at time $t = 0, 1, \dots, N$, where $\omega(t)$ and $i(t)$ are the angular speed and current, respectively, and let $u(t) \in \mathbf{R}$ denote the control input (voltage) at time $t = 0, 1, \dots, N - 1$. We collect the complete state and control trajectories as $x = \begin{bmatrix} x(0) & \dots & x(N) \end{bmatrix} \in \mathbf{R}^{2 \times (N+1)}$ and $u = (u(0), \dots, u(N - 1)) \in \mathbf{R}^N$, respectively, with the previous input $u(-1) = 0$ as fixed data. The motor dynamics can be expressed as a linear discrete-time system

$$x(t + 1) = Ax(t) + Bu(t),$$

where $A \in \mathbf{R}^{2 \times 2}$ and $B \in \mathbf{R}^{2 \times 1}$ are given system matrices. We consider an MPC controller that solves the following lower problem for given regularization weights $\delta, \eta \in \mathbf{R}_+$:

$$\begin{aligned} S(\delta, \eta) = \operatorname{argmin}_{x, u} \quad & \sum_{t=1}^N (\omega(t) - 1)^2 + 5(\omega(N) - 1)^2 + 0.04 \sum_{t=1}^N (i(t) - 0.1)^2 \\ & + \delta \sum_{t=0}^{N-1} (u(t) - 0.2)^2 + \eta \sum_{t=0}^{N-1} (u(t) - u(t-1))^2 \\ \text{subject to} \quad & x(0) = (0, 0), \quad x(t+1) = Ax(t) + Bu(t), \quad t = 0, \dots, N-1 \\ & |u(t)| \leq 2, \quad t = 0, \dots, N-1 \\ & |i(t)| \leq 0.5, \quad -0.05 \leq \omega(t) \leq 1.25, \quad t = 0, \dots, N. \end{aligned}$$

In this example, we learn the weights δ for the control magnitude and η for the control rate of change from expert state and control trajectories $\hat{x}(t)$, $t = 0, \dots, N$ and $\hat{u}(t)$, $t = 0, \dots, N - 1$

over the same horizon, while holding all other weights fixed. The corresponding upper problem is given by

$$\begin{aligned}
& \text{minimize} && \sum_{t=0}^N \|x(t) - \hat{x}(t)\|_2^2 + 0.1 \sum_{t=0}^{N-1} \|u(t) - \hat{u}(t)\|_2^2 \\
& \text{subject to} && 0.01 \leq \delta \leq 1, \quad 0.01 \leq \eta \leq 1 \\
& && (x, u) \in S(\delta, \eta),
\end{aligned} \tag{6.8}$$

where δ, η are the upper variables and x, u are the lower variables. The following code snippet shows how to specify the bilevel problem (6.8) in BLVPY.

```

1 # problem variables
2 delta = cp.Variable(nonneg=True)
3 eta = cp.Variable(nonneg=True)
4 x = cp.Variable((2, 25))
5 u = cp.Variable(24)
6
7 # lower problem
8 u_diff = cp.diff(cp.hstack([0, u]))
9 mpc_obj = (
10     cp.sum_squares(x[0, 1:] - 1)
11     + 5 * cp.square(x[0, -1] - 1)
12     + 0.04 * cp.sum_squares(x[1, 1:] - 0.1)
13     + delta * cp.sum_squares(u - 0.2)
14     + eta * cp.sum_squares(u_diff)
15 )
16 x_next = A @ x[:, :-1] + cp.outer(B, u)
17 lower = bp.LowerProblem(
18     cp.Minimize(mpc_obj),
19     [
20         x[:, 0] == np.zeros(2),
21         x[:, 1:] == x_next,
22         cp.abs(u) <= 2,
23         cp.abs(x[1, :]) <= 0.5,
24         x[0, :] >= -0.05,
25         x[0, :] <= 1.25,
26     ],
27     parameters=[delta, eta],
28 )
29
30 # upper problem
31 loss = (
32     cp.sum_squares(x - x_hat)
33     + 0.1 * cp.sum_squares(u - u_hat)
34 )
35 problem = bp.BilevelProblem(
36     cp.Minimize(loss),
37     lower,
38     [
39         delta >= 0.01, delta <= 1.0,
40         eta >= 0.01, eta <= 1.0,
41     ]
42 )

```

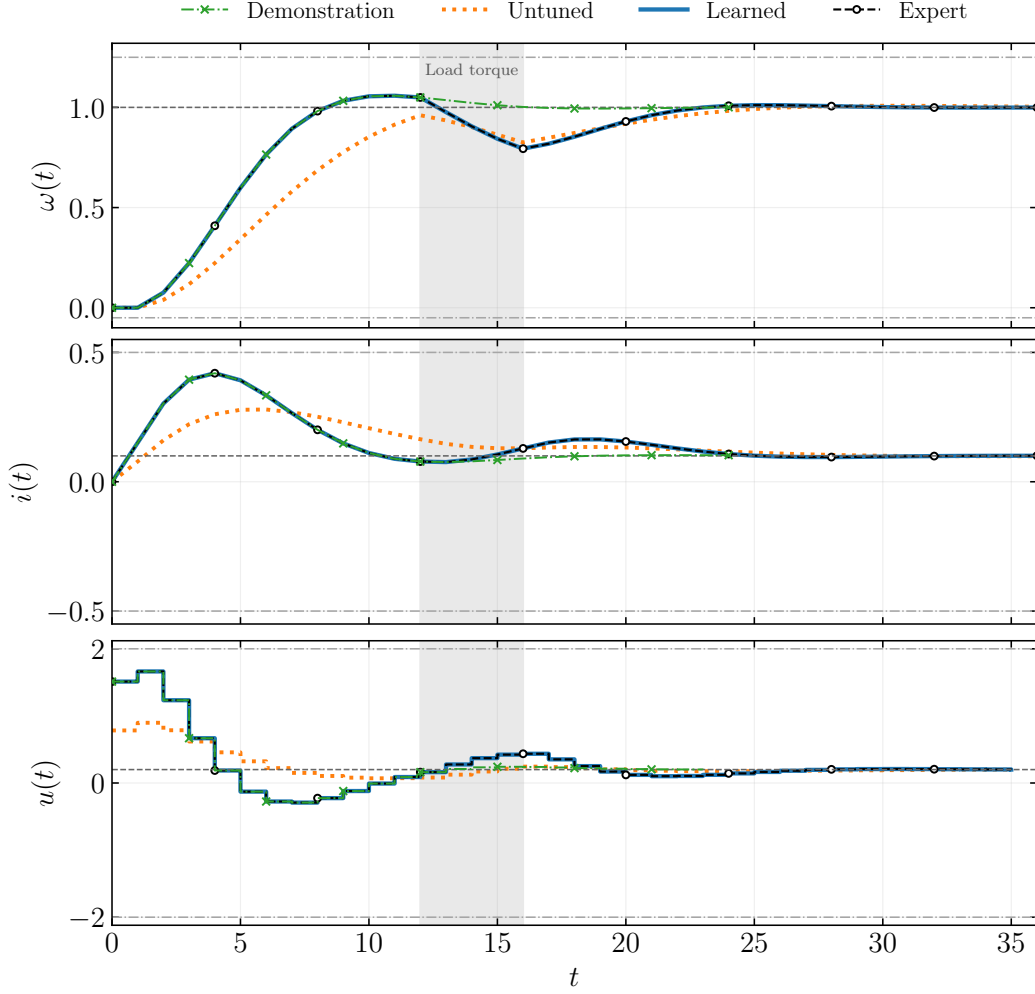


Figure 8 *Differentiable model predictive control.* Validation trajectories for angular speed $\omega(t)$ (top), current $i(t)$ (middle), and control input $u(t)$ (bottom) under the untuned baseline controller with uniform weights $\delta = \eta = 0.5$ (shown in orange) and the learned controller with weights returned by BLVPY (shown in blue). The expert demonstration used for imitation learning and the expert validation trajectory are shown in green and black, respectively. The shaded region indicates the duration of an unannounced load torque pulse. The horizontal dashed and dashdotted lines plot the target values and corresponding safety limits, respectively.

We solve the problem (6.8) with BLVPY using synthetic expert demonstrations, then validate the learned weights in a 36-step receding horizon rollout with an unannounced load torque pulse. We also compare the learned weights with an untuned baseline controller that uses uniform weights $\delta = \eta = 0.5$. The resulting closed-loop trajectories are shown in figure 8. The learned weights returned by BLVPY reproduce the expert behavior both in the nominal case and under the unannounced load torque pulse. In particular, on the nominal demonstration trajectory, the imitation loss (*i.e.*, the upper objective in problem (6.8)) decreases from 0.90 for the untuned baseline to essentially zero for the learned controller.

7 Discussion

7.1 Conclusion

In this paper, we introduce DBLP, a disciplined framework for modeling and solving bilevel optimization problems. If a bilevel problem has a DNLP-compliant upper problem and a DPP-compliant convex lower problem, DBLP can then automatically constructs a single-level reformulation of the original problem via the conic KKT conditions and solves it using a gap continuation procedure. Based on these ideas, we implement the open-source BLVPY package as an extension to CVXPY, which allows users to specify bilevel problems in a high-level, human-readable way, while automatically handling the low-level transformations and solver interfacing.

To illustrate the expressive scope of DBLP, we consider examples from several application domains. In each case, the bilevel model can be specified compactly in a few lines of human-readable code close to the original mathematical formulation, while DBLP automatically constructs the single-level reformulation and interfaces it with the numerical solver. By separating high-level model specification from these low-level transformations, DBLP is intended to make bilevel programming more accessible in practice. We hope that this will facilitate its use in new application areas.

7.2 Solution quality and guarantees

We should emphasize that the DBLP framework and gap continuation procedure do *not* guarantee global optimality of the computed solution, even when the upper-level objective and constraints are convex. This is because the single-level reformulation (3.3) and its relaxation (4.1) are generally nonconvex and are therefore solved only approximately by nonlinear solvers. In the most general case, little can be said about the convergence properties of generic nonlinear numerical solvers; interested readers may refer to Cederberg *et al.* [CZNB26, §2.3] for a detailed discussion.

What DBLP *does* provide is a lower-level certificate: If the final point satisfies the constraints of (4.1) for a tolerance ϵ , proposition 4.1 ensures that the recovered lower point is feasible and at most ϵ -suboptimal for the lower problem, *i.e.*, roughly speaking, the returned point is approximately bilevel-feasible for the original problem (1.1). Our examples show that such a point is often a good (or at least acceptable) solution in practice.

7.3 Lower-suboptimality and polishing

It can happen in practice that for a given bilevel problem of the form (1.1), the continuation procedure described in §4.2 cannot reach a requested tolerance ϵ . If this point was accepted

at a larger tolerance, say, $\tilde{\epsilon} > \epsilon$, proposition 4.1 still certifies lower-level feasibility and at most $\tilde{\epsilon}$ lower-level suboptimality, and the point may therefore be reported as an approximate solution of (1.1) together with its actual complementarity gap and residuals.

On the other hand, as a post-processing step, one may fix the upper variable at its returned value and solve the associated convex lower problem (1.2) again to obtain a more accurate lower-level response. Computationally, this polishing step is typically inexpensive and numerically reliable since it requires only a single convex solve. It tends to work particularly well when the lower solution is unique and stable with respect to the upper variable and the upper constraints have sufficient slack, so that the change in the lower variable does not destroy upper-level feasibility. We should note, however, that this replacement is not an automatic repair, since it can change the upper objective, violate upper constraints that depend on the lower variable, and, when the lower problem has multiple solutions, return a response that is not the one selected by the optimistic bilevel problem. The polished upper-lower solution pair should therefore be reevaluated against all upper constraints and objectives; unless it passes these checks, the resolve should be treated as a diagnostic rather than as a certified solution of the original bilevel problem. If the polished solution is indeed feasible and improves the upper objective, it may be accepted as a better solution of the original problem.

To support this post-solve polishing procedure, BLVPY provides a `polish` method on the `BilevelProblem` class that can be applied directly to any solution returned by `solve`, so that a user can perform this step directly in native BLVPY workflow without manually reconstructing and initializing the lower convex problem again in CVXPY.

A Proofs

In this appendix, we provide the detailed proofs of the results stated in the main text.

A.1 Proof of proposition 4.1

Proof. Fix a feasible point (x, u, s, λ) of (4.1), and denote its canonical primal and dual objective values by

$$P = c(x)^T u + d(x) \quad \text{and} \quad D = -b(x)^T \lambda + d(x),$$

respectively. Primal feasibility gives $b(x) = A(x)u + s$, while dual feasibility gives $A(x)^T \lambda = -c(x)$. Substituting these identities into the primal-dual gap yields

$$\begin{aligned} P - D &= c(x)^T u + b(x)^T \lambda \\ &= c(x)^T u + (A(x)u + s)^T \lambda \\ &= c(x)^T u + u^T A(x)^T \lambda + s^T \lambda \\ &= s^T \lambda. \end{aligned}$$

Because (u, s) is primal feasible and λ is dual feasible, weak duality gives

$$D \leq q^*(x) \leq P,$$

where $q^*(x)$ is the optimal value of the canonical lower problem (3.1) for the given parameter x . It then follows that

$$0 \leq P - q^*(x) \leq P - D = s^T \lambda.$$

According to the losslessness assumption in §3.2, we have

$$p^*(x) = q^*(x).$$

Moreover, the feasible point recovery property ensures that $\rho(u)$ is feasible for the original lower problem and that

$$f_0(x, \rho(u)) \leq P.$$

Lower-level feasibility also gives

$$f_0(x, \rho(u)) \geq p^*(x).$$

Combining these inequalities, we have

$$0 \leq f_0(x, \rho(u)) - p^*(x) \leq P - p^*(x) \leq s^T \lambda.$$

Finally, feasibility of (4.1) imposes $s^T \lambda \leq \epsilon$, which proves (4.2). $\square$

A.2 Proof of corollary 4.2

Proof. The feasible set of the lower problem is convex because the lower problem is DPP-compliant. Strong convexity on this set implies that the lower minimizer is unique: If two distinct feasible points were both optimal, their midpoint would be feasible and, by strong

convexity, would have a strictly smaller objective value, yielding a contradiction. We may therefore write

$$S(x) = \{y^*(x)\},$$

which is a singleton set containing the unique lower minimizer $y^*(x)$.

For the fixed x , let (u, s, λ) be any feasible lift in (4.1) and set $y = \rho(u)$. By the feasible-point recovery property in §3.2, y is feasible for the original lower problem. For any $\theta \in (0, 1)$, convexity of the feasible set implies that

$$y_\theta = (1 - \theta)y^*(x) + \theta y$$

is lower feasible. By optimality of $y^*(x)$ and μ -strong convexity,

$$\begin{aligned} f_0(x, y^*(x)) &\leq f_0(x, y_\theta) \\ &\leq (1 - \theta)f_0(x, y^*(x)) + \theta f_0(x, y) - \frac{\mu}{2}\theta(1 - \theta)\|y - y^*(x)\|_2^2. \end{aligned}$$

After rearranging and dividing by $\theta > 0$, we obtain

$$f_0(x, y) - p^*(x) \geq \frac{\mu}{2}(1 - \theta)\|y - y^*(x)\|_2^2.$$

Taking $\theta \rightarrow 0$ yields

$$f_0(x, \rho(u)) - p^*(x) \geq \frac{\mu}{2}\|\rho(u) - y^*(x)\|_2^2.$$

Recall that proposition 4.1 bounds the left-hand side by ϵ . Rearranging yields (4.3). $\square$

B Consistency of globally solved relaxations

For $\epsilon \geq 0$, let $\mathcal{W}_\epsilon$ denote the feasible set of the relaxed problem (4.1) in the full lifted space of $w = (x, u, s, \lambda)$. In particular, $\mathcal{W}_0$ is the feasible set of the exact single-level reformulation (3.3) (i.e., when $\epsilon = 0$), since $s \in \mathcal{K}$ and $\lambda \in \mathcal{K}^*$ imply $\lambda^T s \geq 0$. For the result below, we additionally assume that the exact reformulation (3.3) is feasible and that $\mathcal{W}_{\tilde{\epsilon}}$ is compact for some $\tilde{\epsilon} > 0$. The compactness of $\mathcal{W}_{\tilde{\epsilon}}$ is only a sufficient condition and may be replaced by any application-specific condition ensuring that the sequence of lifted solutions introduced below has an accumulation point.

Proposition B.1 *Consistency of globally solved relaxations.* Suppose that the assumptions in §3.2 and the lifted-compactness condition above hold. Let the sequence $(\epsilon^{(k)})$ satisfy $0 < \epsilon^{(k)} \leq \tilde{\epsilon}$ and decrease to zero for $k = 0, 1, 2, \dots$, and let

$$w^{*(k)} = (x^{*(k)}, u^{*(k)}, s^{*(k)}, \lambda^{*(k)})$$

be a global solution of (4.1) with $\epsilon = \epsilon^{(k)}$. Then every accumulation point of the sequence $(w^{*(k)})$ is a global solution of the exact reformulation (3.3), and the optimal values of the relaxed problems converge to the optimal value of the exact reformulation.

Proof. Because $0 < \epsilon^{(k)} \leq \tilde{\epsilon}$, the nesting of the relaxed feasible sets gives

$$w^{*(k)} \in \mathcal{W}_{\epsilon^{(k)}} \subseteq \mathcal{W}_{\tilde{\epsilon}}.$$

The lifted-compactness condition therefore ensures that the sequence $(w^{*(k)})$ has an accumulation point. Let $\bar{w} = (\bar{x}, \bar{u}, \bar{s}, \bar{\lambda})$ be an arbitrary accumulation point, and choose a subsequence such that

$$w^{*(k_j)} \rightarrow \bar{w}.$$

The upper constraints are closed by continuity of F_i for $i = 1, \dots, m$ and ρ , the primal- and dual-feasibility equations are continuous, and the cones $\mathcal{K}$ and $\mathcal{K}^*$ are closed. Hence, all constraints of (3.3) other than exact complementarity are preserved in the limit. Moreover, feasibility of each relaxed problem gives

$$0 \leq \lambda^{*(k_j)T} s^{*(k_j)} \leq \epsilon^{(k_j)} \rightarrow 0.$$

Continuity of the inner product then yields $\bar{\lambda}^T \bar{s} = 0$, so $\bar{w} \in \mathcal{W}_0$.

Let $w \in \mathcal{W}_0$ be arbitrary. Because $\mathcal{W}_0 \subseteq \mathcal{W}_{\epsilon^{(k_j)}}$, global optimality of $w^{*(k_j)}$ for its relaxed problem gives

$$F_0(x^{*(k_j)}, \rho(u^{*(k_j)})) \leq F_0(x, \rho(u)).$$

Passing to the limit and using continuity of F_0 and ρ gives

$$F_0(\bar{x}, \rho(\bar{u})) \leq F_0(x, \rho(u)).$$

Thus, $\bar{w}$ is globally optimal over $\mathcal{W}_0$. Because the accumulation point was arbitrary, every accumulation point of $(w^{*(k)})$ is a global solution of (3.3).

Finally, let $v^{*(k)}$ denote the optimal value of the relaxed problem with tolerance $\epsilon^{(k)}$, and let v_0^* denote the optimal value of the exact reformulation when $\epsilon = 0$. Since the feasible sets shrink as $\epsilon^{(k)}$ decreases and $\mathcal{W}_0 \subseteq \mathcal{W}_{\epsilon^{(k)}}$, we have

$$v^{*(k)} \leq v^{*(k+1)} \leq v_0^*.$$

Along the convergent subsequence above, continuity of the objective and global optimality of $\bar{w}$ give $v^{*(k_j)} \rightarrow v_0^*$. The monotonicity of $(v^{*(k)})$ then implies that the full sequence converges to v_0^* . $\square$

Under the assumptions in §3.2, the conic KKT conditions (3.2) are necessary and sufficient for $\rho(u)$ to solve the lower problem for a given x . Thus, the recovery map $(x, u, s, \lambda) \mapsto (x, \rho(u))$ sends every feasible point of the exact reformulation (3.3) to a feasible point of the original bilevel problem (1.1), while every bilevel-feasible point admits such a lift; the upper objective is unchanged under this correspondence. Consequently, every accumulation point identified by proposition B.1 recovers a globally optimal solution of the original bilevel problem.

Note that without global solves (*e.g.*, in algorithm 1 and practical applications), any accumulation point of a sequence of exactly feasible relaxed points with $\epsilon^{(k)} \rightarrow 0$ is still feasible for the exact reformulation, but it need not be optimal. For locally or approximately solved relaxations, the lower-level certificate in proposition 4.1 remains valid whenever primal feasibility, dual feasibility, cone membership, and the relaxed complementarity constraint are satisfied, but convergence to a globally optimal bilevel solution is not guaranteed.

Acknowledgments

This work has been funded as part of BrainLinks-BrainTools, which is funded by the Federal Ministry of Economics, Science and Arts of Baden-Württemberg within the sustainability program for projects of the Excellence Initiative II, and CRC/TRR 384 “IN-CODE”.

References

- [AAB⁺19] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [AB20] A. Agrawal and S. Boyd. Disciplined quasiconvex programming. *Optimization Letters*, 14:1643–1657, 2020.
- [ABB⁺19] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 1(2):107–115, 2019.
- [ABB21] A. Agrawal, S. Barratt, and S. Boyd. Learning convex optimization models. *IEEE/CAA Journal of Automatica Sinica*, 8(8):1355–1364, 2021.
- [ADB19] A. Agrawal, S. Diamond, and S. Boyd. Disciplined geometric programming. *Optimization Letters*, 13:961–976, 2019.
- [AJS⁺18] B. Amos, I. D. Jimenez Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for end-to-end planning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- [AK17] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 2017.
- [AKDB15] A. Ali, J. Z. Kolter, S. Diamond, and S. Boyd. Disciplined convex stochastic programming: A new framework for stochastic optimization. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, pages 62–71. AUAI Press, 2015.
- [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.
- [Bar98] J. F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*, volume 30 of *Nonconvex Optimization and Its Applications*. Springer, 1998.
- [BB21] S. T. Barratt and S. P. Boyd. Least squares auto-tuning. *Engineering Optimization*, 53(5):789–810, 2021.
- [BHH⁺21] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Sirola, J.-P. Watson, and D. L. Woodruff. *Pyomo—Optimization Modeling in Python*, volume 67 of *Springer Optimization and Its Applications*. Springer, 3rd edition, 2021.
- [BM73] J. Bracken and J. T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973.

- [BMW56] M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation*. Yale University Press, 1956. With an Introduction by T. C. Koopmans. Published for Cowles Commission for Research in Economics.
- [BNW05] D. Braess, A. Nagurney, and T. Wakolbinger. On a paradox of traffic planning. *Transportation Science*, 39(4):446–450, 2005. Translated from the original German article “Über ein Paradoxon aus der Verkehrsplanung” by D. Braess, *Unternehmensforschung*, 12:258–268, 1968.
- [BNW06] R. H. Byrd, J. Nocedal, and R. A. Waltz. KNITRO: An integrated package for nonlinear optimization. In G. Di Pillo and M. Roma, editors, *Large-scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 35–59. Springer, 2006.
- [BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [CK09] P. W. Christensen and A. Klarbring. *An Introduction to Structural Optimization*, volume 153 of *Solid Mechanics and Its Applications*. Springer, 2009.
- [CMS07] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153:235–256, 2007.
- [CZNB26] D. Cederberg, W. Zhang, P. Nobel, and S. Boyd. Disciplined nonlinear programming. *arXiv*, 2606.02896, 2026.
- [DB16] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
- [DBS23] J. Dias Garcia, G. Bodin, and A. Street. BilevelJuMP.jl: Modeling and solving bilevel optimization problems in Julia. *INFORMS Journal on Computing*, 36(2):327–335, 2023.
- [DD06] J. Dutta and S. Dempe. Bilevel programming with convex lower level problems. In S. Dempe and V. Kalashnikov, editors, *Optimization with Multivalued Mappings: Theory, Applications, and Algorithms*, volume 2 of *Springer Series in Optimization and Its Applications*, pages 51–71. Springer, 2006.
- [DD12] S. Dempe and J. Dutta. Is bilevel programming a special case of a mathematical program with complementarity constraints? *Mathematical Programming*, 131:37–48, 2012.
- [Dem02] S. Dempe. *Foundations of Bilevel Programming*, volume 61 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 2002.
- [DHL17] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.
- [Dia25] S. Diamond. *Crate cvxrust Documentation*, 2025. Version 0.1.0. Available at <https://docs.rs/cvxrust/0.1.0/cvxrust/>.

- [DKK06] S. Dempe, V. V. Kalashnikov, and N. Kalashnykova. Optimality conditions for bilevel programming problems. In S. Dempe and V. Kalashnikov, editors, *Optimization with Multivalued Mappings: Theory, Applications, and Algorithms*, volume 2 of *Springer Series in Optimization and Its Applications*, pages 3–28. Springer, 2006.
- [DZ20] S. Dempe and A. Zemkoho, editors. *Bilevel Optimization: Advances and Next Challenges*, volume 161 of *Springer Optimization and Its Applications*. Springer, 2020.
- [FFS⁺18] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1568–1577. PMLR, 2018.
- [FJQ99] F. Facchinei, H. Jiang, and L. Qi. A smoothing method for mathematical programs with equilibrium constraints. *Mathematical Programming*, 85:107–134, 1999.
- [FNB20] A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 94:1–34, 2020.
- [GB14] M. Grant and S. Boyd. CVX: MATLAB software for disciplined convex programming, version 2.1, 2014.
- [GBY06] M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In *Global Optimization: From Theory to Implementation*, pages 155–210. Springer, 2006.
- [GC24] P. J. Goulart and Y. Chen. Clarabel: An interior-point solver for conic programs with quadratic objectives. *arXiv*, 2405.12762, 2024.
- [GFC⁺16] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. Santa Cruz, and E. Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv*, 1607.05447, 2016.
- [GGF⁺19] Q. Guo, J. Gan, F. Fang, L. Tran-Thanh, M. Tambe, and B. An. On the inducibility of Stackelberg equilibrium for security games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2020–2028. AAAI Press, 2019.
- [GHW⁺22] D. Ge, Q. Huangfu, Z. Wang, J. Wu, and Y. Ye. Cardinal Optimizer (COPT) user guide. *arXiv*, 2208.14314, 2022.
- [GPT13] J. Gouveia, P. A. Parrilo, and R. R. Thomas. Lifts of convex sets and cone factorizations. *Mathematics of Operations Research*, 38(2):248–264, 2013.
- [Gra05] M. Grant. *Disciplined Convex Programming*. PhD thesis, Stanford University, 2005.
- [HC19] W. Hart and A. Castillo. Pao, 2019. Computer software, available at <https://doi.org/10.11578/dc.20201105.1>.

- [HJS92] P. Hansen, B. Jaumard, and G. Savard. New branch-and-bound rules for linear bilevel programming. *SIAM Journal on Scientific and Statistical Computing*, 13(5):1194–1217, 1992.
- [HK70] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- [HNB26] Q. Healey, P. Nobel, and S. Boyd. Differentiating through a quadratic cone program. *Optimization Letters*, 2026.
- [HWW11] W. E. Hart, J.-P. Watson, and D. L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3:219–260, 2011.
- [LDD⁺23] M. Lubin, O. Dowson, J. Dias Garcia, J. Huchette, B. Legat, and J. P. Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 15:581–589, 2023.
- [LM96] P. Loridan and J. Morgan. Weak via strong Stackelberg problem: New results. *Journal of Global Optimization*, 8:263–287, 1996.
- [LPR96] Z.-Q. Luo, J.-S. Pang, and D. Ralph. *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press, 1996.
- [LVD20] J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In S. Chiappa and R. Calandra, editors, *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1540–1552. PMLR, 2020.
- [MDA15] D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2113–2122. PMLR, 2015.
- [MOS26] MOSEK ApS. *MOSEK Optimization Suite*, 2026. Version 11.1.3.
- [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.
- [Ped16] F. Pedregosa. Hyperparameter optimization with approximate gradient. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 737–746. PMLR, 2016.
- [RMD26] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2nd edition, 2026.
- [RW04] D. Ralph and S. J. Wright. Some properties of regularization and penalization schemes for MPECs. *Optimization Methods and Software*, 19(5):527–556, 2004.

- [SAY⁺12] E. Shieh, B. An, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. PROTECT: An application of computational game theory for the security of the ports of the United States. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 2173–2179. AAAI Press, 2012.
- [Sch01] S. Scholtes. Convergence properties of a regularization scheme for mathematical programs with complementarity constraints. *SIAM Journal on Optimization*, 11(4):918–936, 2001.
- [SDGB16] X. Shen, S. Diamond, Y. Gu, and S. Boyd. Disciplined convex-concave programming. In *IEEE 55th Conference on Decision and Control*, pages 1009–1014. IEEE, 2016.
- [SDU⁺17] X. Shen, S. Diamond, M. Udell, Y. Gu, and S. Boyd. Disciplined multi-convex programming. In *29th Chinese Control and Decision Conference*, pages 895–900. IEEE, 2017.
- [SLB24] P. Schiele, E. Luxenberg, and S. Boyd. Disciplined saddle programming. *Transactions on Machine Learning Research*, 2024.
- [SMD18] A. Sinha, P. Malo, and K. Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2018.
- [SS00] H. Scheel and S. Scholtes. Mathematical programs with complementarity constraints: Stationarity, optimality, and sensitivity. *Mathematics of Operations Research*, 25(1):1–22, 2000.
- [TAA17] T. Terlaky, M. F. Anjos, and S. Ahmed, editors. *Advances and Trends in Optimization with Engineering Applications*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2017.
- [UMZ⁺14] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. In *Proceedings of the Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28, 2014.
- [Ver02] E. T. Verhoef. Second-best congestion pricing in general networks. Heuristic algorithms for finding second-best optimal toll levels and toll points. *Transportation Research Part B: Methodological*, 36(8):707–729, 2002.
- [VL26] C. Vanaret and S. Leyffer. Implementing a unified solver for nonlinearly constrained optimization. *Mathematical Programming Computation*, 2026.
- [VM26] C. Vanaret and A. Montoison. Uno: a composable framework for nonlinearly constrained optimization. *Journal of Open Source Software*, 11(123):10229, 2026.
- [vS51] H. von Stackelberg. *Grundlagen der theoretischen Volkswirtschaftslehre*. Omnitypie-Gesellschaft, 2nd edition, 1951. Translated into English as *The Theory of the Market Economy* by A. T. Peacock, Oxford University Press, 1952.
- [War52] J. G. Wardrop. Road paper. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers*, 1(3):325–362, 1952.

- [WB06] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006. Prepring on *Optimization Online*: <https://optimization-online.org/?p=9506>. IBM Research Report, RC 23149, IBM T.J. Watson Research Center Yorktown Heights, NY. USA, 2004.
- [YZ95] J. J. Ye and D. L. Zhu. Optimality conditions for bilevel programming problems. *Optimization*, 33(1):9–27, 1995.
- [ZB26] H. Zhu and J. Boedecker. *Disciplined Machine Learning*, 2026. Working draft (dated June 28, 2026), available at <https://github.com/dxogrp/dmlbook>.
- [ZNJ⁺26] W. Zhang, P. Nobel, A. Jeendgar, R. Murray, P. Schiele, and S. Diamond. CVXPY 1.9: Recent advances in optimization modeling software. *arXiv*, 2606.14891, 2026.
- [ZZFC17] Q. Zeng, B. Zhang, J. Fang, and Z. Chen. A bi-level programming for multistage co-expansion planning of the integrated gas and electricity system. *Applied Energy*, 200:192–203, 2017.