Disciplined Biconvex Programming

Hao Zhu[1,2] and Joschka Boedecker[1,2]

[1]IMBIT//BrainLinks-BrainTools
[2]Department of Computer Science, University of Freiburg

November 10, 2025

Abstract

We introduce *disciplined biconvex programming* (DBCP), a modeling framework for specifying and solving biconvex optimization problems. Biconvex optimization problems arise in various applications, including machine learning, signal processing, computational science, and control. Solving a biconvex optimization problem in practice usually resolves to heuristic methods based on alternate convex search (ACS), which iteratively optimizes over one block of variables while keeping the other fixed, so that the resulting subproblems are convex and can be efficiently solved. However, designing and implementing an ACS solver for a specific biconvex optimization problem usually requires significant effort from the user, which can be tedious and error-prone. DBCP extends the principles of disciplined convex programming to biconvex problems, allowing users to specify biconvex optimization problems in a natural way based on a small number of syntax rules. The resulting problem can then be automatically split and transformed into convex subproblems, for which a customized ACS solver is then generated and applied. DBCP allows users to quickly experiment with different biconvex problem formulations, without expertise in convex optimization. We implement DBCP into the open source Python package `dbcp`, as an extension to the famous domain specific language `CVXPY` for convex optimization.

Contents

1 Introduction

We consider biconvex optimization problems, which consist in optimizing a biconvex objective function subject to a biconvex set constraint. Informally, by saying that a set (or function) is biconvex, we mean that it is convex in each of two blocks of variables when the other block is fixed. Biconvex optimization problems arise in various applications, including machine learning [PT94, UHZB16, ZYHB25], signal and image processing [LWDF09, CVR14], recommender systems [CP09, Joh14], control [SGL94, VB00, JDMV20], and brain computer interfaces [DCP07, SXB14].

Different from convex optimization problems, biconvex optimization problems are in general nonconvex and can be very hard to solve. In fact, many of them have been shown to be NP-hard (see [WYZ12], [TÖ95], and [ZYHB25] for some examples). Nevertheless, various heuristic methods based on *alternate convex search* (ACS) [DL94] exist for finding good solutions to many biconvex optimization problems in practice. The basic idea of ACS-type methods is to iteratively optimize over one block of variables while keeping the other block fixed, such that each subproblem is convex and can be efficiently solved. Although in the most general case, theoretical convergence of ACS-type methods is only guaranteed to stationary points, these methods turn out to work quite well in practice and hence have become the most popular choice for finding a satisfactory solution to biconvex optimization problems.

Designing and implementing an ACS solver for a specific biconvex optimization problem usually requires significant effort from the user, including properly partitioning the biconvex problem into convex subproblems, implementing robust and efficient solvers for the subproblems, and designing appropriate stopping criteria. Moreover, maintaining numerical stability and ensuring convergence of the ACS procedure typically necessitate integrating appropriate regularizations or modifications into the subproblems to ensure properties such as strong convexity and feasibility. With the help of *domain specific languages* (DSLs) for convex optimization, such as `CVXPY` [DB16, AVDB18], specifying and solving the (modified) convex subproblems is largely simplified and automated, while the other steps still require the user to have expert knowledge about convex analysis and programming, and can be tedious and error-prone.

In this paper, we propose a modeling framework for biconvex optimization problems, named *disciplined biconvex programming* (DBCP), which extends the ideas of *disciplined convex programming* (DCP) [GBY06] to biconvex problems. DBCP for biconvex optimization is analogous to DCP for convex optimization, which allows users to specify biconvex optimization problems in a natural way based on a small number of syntax rules. When an optimization problem description complies with the DBCP syntax rules, it is guaranteed to be a valid biconvex problem, and more importantly, it can be automatically split and transformed into convex (specifically, DCP-compliant) subproblems and augmented, to which a customized ACS solver is then generated and applied. Like DCP does for convex optimization, DBCP makes it easy to specify and solve biconvex problems, allowing users to quickly prototype and experiment with different biconvex problem formulations, without expertise in (or even knowledge of) convex analysis and solution methods for biconvex problems. In fact, most users might be unaware of the variable partition, transformation, augmentation, and solution processes, from which a solution to the biconvex problem is found. We implement DBCP into a Python package, named `dbcp`, as an extension of the DSL `CVXPY`, which is fully open-sourced at

`https://github.com/nrgrp/dbcp`.

1.1 Previous and related work

Biconvex analysis. The first notice of biconvexity structure in the context of mathematical programming can be traced back to Falk *et al.* [FS69] in the 1960s. Only a few papers exist in the literature where biconvex sets are investigated. Most of the corresponding theoretical results can be found in the papers of Aumann and Hart [AH86] and Goh *et al.* [GTS+94]. Biconvex functions appear regularly in practice and hence have been discussed widely in the literature. Properties of biconvex functions that are most relevant to optimization problems can be found in the works of Goh *et al.* [GTS+94] and Gorski *et al.* [GPK07]. In addition, biconvex functions play an important role in martingale theory, and, in particular, in the analysis of Banach and Hilbert spaces [Bur81, Bur86, AH86, Lee93]. Biconvex functions can also be used to derive results on the robust stability of control systems in practical control engineering [GH00a, GH00b]. Thibault [Thi84], Jouak and Thibault [JT85], and Borwein [Bor86] analyzed the continuity and differentiability of measurable biconvex operators in topological vector spaces. Al-Khayyal and Falk [AKF83] published results on the maximization of biconvex functions. In the most general case, very little can be said about the global or even local optimality properties of biconvex optimization problems. Nevertheless, some useful properties of partial optimality conditions (*i.e.*, conditions for stationary points) of biconvex problems are discussed in [WHJ76, GPK07].

Solution methods for biconvex problems. Current solution methods for biconvex problems can be roughly categorized into two classes: heuristic methods and global optimization methods. Heuristic methods for biconvex problems aim at finding stationary points of the biconvex objective function, and are mostly based on the idea of alternately optimizing two convex subproblems, *i.e.*, the ACS-type methods. ACS methods are a special case of *block relaxation methods* [War63, Pow73, DL94] where the variables are divided into disjoint blocks, and in each step, only the variables of an active block are optimized while those of the other blocks are fixed. In particular, for ACS, only two blocks of variables defined by the convex subproblems are activated in cycles. Since the resulting subproblems are convex, efficient convex minimization methods can be used to solve these subproblems. A survey on ACS methods for biconvex optimization problems can be found in [WHJ76]. Gorski *et al.* [GPK07] showed that under weak assumptions all solution points generated by ACS form a compact connected set and that each of these points is a stationary point of the objective function. However, no better convergence results regarding local or global optimality properties can be obtained in general. Regarding the global optimization methods, Floudas and Visweswaran [FV90] adapted the idea of *branch-and-bound* [LW66, BM07] to solve biconvex problems with global optimality. Detailed mathematical background and outline of this algorithm are given in [FV90, FV93, Flo95, Flo00], and some basic convergence properties of this method are discussed in [GPK07].

Biconvex optimization applications. The practical usefulness of biconvex programming in many applications is also increasingly well known. In machine learning, nonnegative matrix factorization [PT94, LS99, UHZB16], dictionary learning [AEB06], and bilinear regression [HKV08, CP09, Joh14] are all well-known biconvex optimization problems, which have been widely applied in representation learning, signal processing, and recommender systems. The very famous k-means clustering problem can also be formulated as a biconvex program [Llo82, SMD22, ZYHB25]. Blind deconvolution problems [LWDF09, CVR14] are another important class of biconvex optimization problems for image processing and

communication. Other studies such as Mishra *et al.* [MDC17] and Fosson [Fos18] explored biconvex structures in sparse learning and reweighted regression. Biconvex problems also appear when dealing with bilinear matrix inequalities for bilinear control system synthesis [SGL94, VB00, JDMV20]. Finally, Dyrholm *et al.* [DCP07] and Shi *et al.* [SXB14] applied bilinear discriminant component analysis techniques to brain computer interfaces. We discuss some of these applications in more detail in §6.

Disciplined convex programming. DCP [GBY06] is a modeling framework for convex optimization problems. If a mathematical optimization problem is written following the DCP syntax ruleset, it is guaranteed to be a valid convex program, and can then be automatically transformed into the standard conic form [AVDB18], which can be handled by generic conic solvers, such as OSQP [SBG+20], ECOS [DCB13], SCS [OCPB16], and Clarabel [GC24]. Many DSLs for convex optimization, such as `YALMIP` [Lof04], `CVX` [GB14], `Convex.jl` [UMZ+14], `CVXPY` [DB16], and `CVXR` [FNB20], are based on the DCP ruleset. Over the years, DCP has been extended to handle stochastic [AKDB15], convex-concave [SDGB16], geometric [ADB19], quasiconvex [AB20], and saddle [SLB24] problems. The most related extension of DCP to our work is *disciplined multi-convex programming* [SDU+17], which provides a DSL for specifying and solving multi-convex optimization problems via block relaxation methods. However, it seems to be no longer actively supported or maintained. In this paper, we work with `CVXPY` and focus on biconvex problems only, since most multi-convex problems in practice can be formulated as biconvex problems by grouping variables appropriately.

1.2 Outline

This paper is *not* about establishing new theoretical results or computational methods for biconvex optimization problems. Instead, we collect well-known ideas and assemble them into a disciplined DSL, for specifying and solving biconvex optimization problems in a natural human-readable way which is close to the mathematical formulation. These ideas are implemented in the open-source Python package `dbcp`, which extends `CVXPY` to support biconvex programming.

The rest of this paper is organized as follows. In §2, we provide formal definitions for biconvex sets, functions, and optimization problems, including some specific examples of these objects, and briefly review their basic properties. In §3, we introduce the ACS-based method for solving biconvex optimization problems, and discuss some practical augmentations when dealing with initialization and numerical stability, which are all integrated into the `dbcp` package. Then in §4, we introduce the DBCP biconvex syntax ruleset for specifying biconvex optimization problems, as an extension of the DCP convex ruleset. The implementation and basic usage of the provided Python package `dbcp`, where the functions and features mentioned above are implemented, is discussed in §5. Finally, in §6, we present some specific numerical examples for specifying and solving biconvex optimization problems that frequently appear in practice using the `dbcp` package, to show the simplicity and effectiveness of our framework. These examples along with the corresponding code snippets also serve as a preliminary tutorial for new users to get started with `dbcp`.

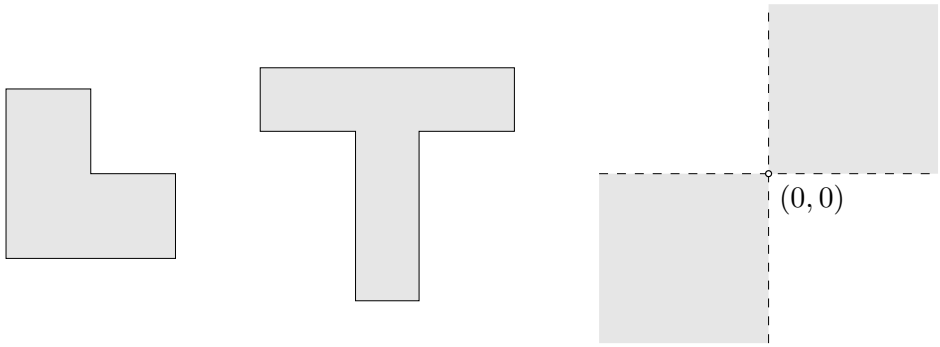

Figure 1 Examples of biconvex sets.

2 Biconvex programming

In this section, we provide formal definitions for biconvex sets, biconvex functions, and biconvex optimization problems, to establish our notation, and briefly review some of the basic properties of these objects. To start with, let $\mathcal{X} \subseteq \mathbf{R}^n$ and $\mathcal{Y} \subseteq \mathbf{R}^k$ be two nonempty closed convex sets. We should also note that throughout this paper, we use the convention that functions are implicitly extended-valued, *i.e.*, by writing $f \colon A \to \mathbf{R}$, we really mean $f \colon A \to \mathbf{R} \cup \{\infty\}$. In other words, a function defined on A can take the value ∞ for some points in A [Roc70, BV04].

2.1 Biconvex sets

A set $B \subseteq \mathcal{X} \times \mathcal{Y} \subseteq \mathbf{R}^{n+k}$ is a *biconvex set*, if for every fixed $y \in \mathcal{Y}$, the set

$$B_y = \{x \in \mathcal{X} \mid (x, y) \in B\} \subseteq \mathbf{R}^n$$

is convex, and for every fixed $x \in \mathcal{X}$, the set

$$B_x = \{y \in \mathcal{Y} \mid (x, y) \in B\} \subseteq \mathbf{R}^k$$

is convex.

Obviously, a biconvex set is not necessarily convex in general. Figure 1 shows some examples of biconvex sets which are not convex. In particular, a biconvex set does not even have to be connected, as shown by the example

$$B = \{(x, y) \in \mathbf{R}^2 \mid x, y < 0\} \cup \{(x, y) \in \mathbf{R}^2 \mid x, y > 0\},$$

whose picture is shown on the right of figure 1.

An important property of biconvex sets is that, similar to convex sets, the intersection of an arbitrary collection of biconvex sets is still a biconvex set [AH86, GPK07].

2.2 Biconvex functions

A function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$ is a *biconvex function* if its effective domain

$$\mathbf{dom}\, f = \{(x, y) \in \mathcal{X} \times \mathcal{Y} \mid f(x, y) < \infty\}$$

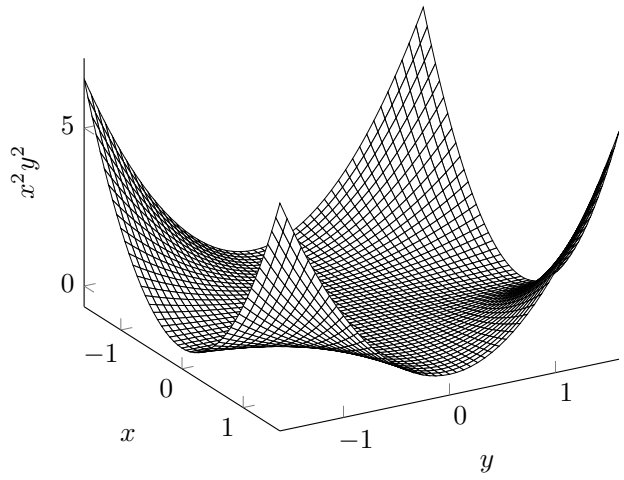

Figure 2 Graph of the biconvex function $f(x, y) = x^2y^2$.

is a biconvex set, and for every fixed $y \in \mathcal{Y}$, the function

$$f_y \colon \mathcal{X} \to \mathbf{R}, \quad x \mapsto f(x, y),$$

is convex in x, and for every fixed $x \in \mathcal{X}$, the function

$$f_x \colon \mathcal{Y} \to \mathbf{R}, \quad y \mapsto f(x, y),$$

is convex in y. In other words, a biconvex function is the one that is convex in each of two blocks of variables when the other block is fixed. We can also define *biconcave*, *biaffine*, and *bilinear* functions similarly, by replacing the property of being convex for f_y and f_x by the property of being concave, affine, or linear, respectively. Figure 2 shows an example of a biconvex function given by $f(x, y) = x^2y^2$.

Now we list some basic properties of biconvex functions. We do not provide formal proofs here; interested readers may refer to [GTS+94] and [GPK07] for the corresponding proofs and discussion of the following statements (which are more or less obvious with the basic convex analysis toolbox [Roc70]). Similar to convex functions, the sublevel sets of a biconvex function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, given by

$$C_\alpha = \{(x, y) \in \mathbf{dom}\, f \mid f(x, y) \leq \alpha\},$$

are biconvex sets, for any $\alpha \in \mathbf{R}$. Also, many arithmetic properties that are valid for convex functions can be transferred to the biconvex case:

- *Nonnegative weighted sums.* Evidently, if f is a biconvex function and $w \geq 0$, then the function wf is biconvex. If f_1 and f_2 are both biconvex functions, then so is their sum $f_1 + f_2$. In the general case, if $f_1, \ldots, f_m$ are biconvex functions, and $w_1, \ldots, w_m \geq 0$, then the function
$$f = w_1 f_1 + \cdots + w_m f_m$$
is also biconvex.

- *Composition with a biaffine mapping.* Suppose $h\colon \mathbf{R}^n \to \mathbf{R}$ is convex, and $g\colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}^n$ is biaffine, then the function $f\colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, given by $f(x,y) = h(g(x,y))$ is biconvex. In particular, if $h\colon \mathbf{R} \to \mathbf{R}$ is convex, and $A \in \mathbf{R}^{m\times n}$, $C \in \mathbf{R}^{m\times k}$, $b, d \in \mathbf{R}^m$, then the function $f\colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, given by

  $$f(x,y) = h((Ax+b)^T(Cy+d))$$

  is biconvex.

- *Pointwise maximum and supremum.* If $f_1, \ldots, f_m$ are biconvex functions, then the function

  $$f(x,y) = \max\{f_1(x,y), \ldots, f_m(x,y)\}$$

  is also biconvex. More generally, if $\{f_i\}_{i\in I}$ is a family of biconvex functions indexed by a set I, then the function

  $$f(x,y) = \sup_{i\in I} f_i(x,y)$$

  is biconvex.

- *Composition.* If $h\colon \mathbf{R} \to \mathbf{R}$ is convex and nondecreasing, and $g\colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$ is biconvex, then the function $f\colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, given by $f(x,y) = h(g(x,y))$ is biconvex. This property can be easily extended to multivariate functions $h\colon \mathbf{R}^m \to \mathbf{R}$ that are convex and nondecreasing in each argument. Interested readers may refer to [BV04, §3.2.4] for more details.

We should note that the property of *composition with a biaffine mapping* above is a special case of the more general *composition* property, but is of more practical importance, since most biconvexity structures in applications arise from composing convex functions with biaffine mappings. (See §6 for some examples.)

2.3 Biconvex optimization problems

A *biconvex optimization problem* is one of the form

$$\begin{array}{lll} \text{minimize} & f_0(x,y) & \\ \text{subject to} & f_i(x,y) \le 0, & i = 1,\ldots,m \\ & h_i(x,y) = 0, & i = 1,\ldots,p, \end{array} \tag{2.1}$$

where $x \in \mathcal{X}$, $y \in \mathcal{Y}$ are the optimization variables. The functions $f_i\colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, $i = 0,\ldots,m$, are biconvex, and $h_i\colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, $i = 1,\ldots,p$, are biaffine. Roughly speaking, the problem (2.1) can be interpreted as minimizing a biconvex objective function over a biconvex feasible set defined by biconvex inequality constraints and biaffine equality constraints, since each of these constraints defines a biconvex set, and the intersection of biconvex sets is still a biconvex set (*cf.* §2.1). The *feasible set* of (2.1) is defined as

$$\mathcal{D} = \left\{ (x,y) \in \mathbf{dom}\, f_0 \;\middle|\; \begin{array}{ll} f_i(x,y) \le 0, & i = 1,\ldots,m \\ h_i(x,y) = 0, & i = 1,\ldots,p \end{array} \right\}.$$

Different from convex optimization problems, in the most general case, very little can be said about the global or even local optimality properties of biconvex optimization problems.

Instead, people usually consider *partial optimality* for biconvex optimization problems, which is even more weaker than local optimality. Suppose $(x^\star, y^\star) \in \mathcal{D}$ is a feasible point of (2.1), then $(x^\star, y^\star)$ is a *partially optimal point* of (2.1) if for all $x \in \mathcal{D}_{y^\star}$, $y \in \mathcal{D}_{x^\star}$, we have

$$f_0(x^\star, y^\star) \leq f_0(x, y^\star) \qquad \text{and} \qquad f_0(x^\star, y^\star) \leq f_0(x^\star, y),$$

where

$$\mathcal{D}_{y^\star} = \left\{ x \in \mathcal{X} \;\middle|\; \begin{array}{ll} f_0(x, y^\star) < \infty & \\ f_i(x, y^\star) \leq 0, & i = 1, \ldots, m \\ h_i(x, y^\star) = 0, & i = 1, \ldots, p \end{array} \right\},$$

and

$$\mathcal{D}_{x^\star} = \left\{ y \in \mathcal{Y} \;\middle|\; \begin{array}{ll} f_0(x^\star, y) < \infty & \\ f_i(x^\star, y) \leq 0, & i = 1, \ldots, m \\ h_i(x^\star, y) = 0, & i = 1, \ldots, p \end{array} \right\}.$$

It can be shown that for a differentiable biconvex optimization problem, every stationary point of f_0 over $\mathcal{D}$ is partially optimal, and vise versa [GPK07]. However, such a point is not necessarily a local optimum, as stationary points can be saddle points of the objective function. Some necessary conditions for a partially optimal point of a biconvex problem being a local optimum are discussed in [GPK07], but in general, no stronger results can be obtained. Albeit a weak notion of optimality, partially optimal points for biconvex problems is still widely used in practice, and turns out to be good enough for many applications.

3 Solving biconvex problems

As a result of the optimality properties regarding biconvex optimization problems discussed in §2.3, 'solving' a biconvex optimization problem in practice usually resolves to finding a stationary point of the objective function over the feasible set. In this section, we introduce the ACS heuristic for this purpose that is implemented in the DBCP framework, where the basic idea is to transform the biconvex problem into two convex subproblems, which can then be handled directly by DCP-type DSLs for convex optimization. We also discuss some practical augmentations to ACS when dealing with the biconvex problem initialization and numerical stability issues.

3.1 Alternate convex search

ACS is a minimization method as a special case of block relaxation methods [DL94], where the variables are divided into two disjoint blocks, and in each step, only the variables of an active block are optimized while those of the other block are fixed. Specifically, for a biconvex optimization problem of the form (2.1), ACS iterates between solving the following two convex subproblems:

$$\begin{array}{ll} \text{minimize} & f_0(x, \tilde{y}) \\ \text{subject to} & f_i(x, \tilde{y}) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x, \tilde{y}) = 0, \quad i = 1, \ldots, p, \end{array} \tag{3.1}$$

where $x \in \mathcal{X}$ is the variable, $\tilde{y} \in \mathcal{Y}$ is the fixed problem data, and

$$\begin{array}{ll} \text{minimize} & f_0(\tilde{x}, y) \\ \text{subject to} & f_i(\tilde{x}, y) \leq 0, \quad i = 1, \ldots, m \\ & h_i(\tilde{x}, y) = 0, \quad i = 1, \ldots, p, \end{array} \tag{3.2}$$

where $y \in \mathcal{Y}$ is the variable, $\tilde{x} \in \mathcal{X}$ is the fixed data. Since the problems (3.1) and (3.2) are convex, efficient convex minimization methods can be used to solve these subproblems. The full ACS procedure is summarized in the following algorithm.

Algorithm 3.1 *Alternate convex search.*

given a starting point $(x^{(0)}, y^{(0)}) \in \mathcal{D}$.
$k := 0$.
repeat

1. *Solve (3.1) with* $\tilde{y} = y^{(k)}$. Try to obtain

$$x^{(k+1)} := \operatorname*{argmin}_{x \in \mathcal{X}} \left\{ f_0(x, y^{(k)}) \;\middle|\; \begin{array}{ll} f_i(x, y^{(k)}) \leq 0, & i = 1, \ldots, m \\ h_i(x, y^{(k)}) = 0, & i = 1, \ldots, p \end{array} \right\}, \tag{3.3}$$

   and quit if infeasible.
2. *Solve (3.2) with* $\tilde{x} = x^{(k+1)}$. Try to obtain

$$y^{(k+1)} := \operatorname*{argmin}_{y \in \mathcal{Y}} \left\{ f_0(x^{(k+1)}, y) \;\middle|\; \begin{array}{ll} f_i(x^{(k+1)}, y) \leq 0, & i = 1, \ldots, m \\ h_i(x^{(k+1)}, y) = 0, & i = 1, \ldots, p \end{array} \right\}, \tag{3.4}$$

   and quit if infeasible.
3. $k := k + 1$.

until stopping criteria is satisfied.

Gorski *et al.* [GPK07] showed that under weak assumptions, the sequence of objective function values $\{f_0(x^{(k)}, y^{(k)})\}_{k=0}^{\infty}$ generated by ACS is monotonically nonincreasing and convergent. Furthermore, if the sequence $\{(x^{(k)}, y^{(k)})\}_{k=0}^{\infty}$ converges to $(x^\star, y^\star)$, then $(x^\star, y^\star)$ is a stationary point of f_0 (and hence, is a partially optimal point of (2.1)). We should note that ACS is sensitive to initialization, *i.e.*, with different initial points $(x^{(0)}, y^{(0)})$, the sequence $\{(x^{(k)}, y^{(k)})\}_{k=0}^{\infty}$ generated through algorithm 3.1 may converge to different stationary points of f_0 with (probably) different function values.

There are several ways to define the stopping criteria for ACS. A simple choice is to stop when the difference of the objective values of (2.1) with the variable values obtained between two consecutive iterations is below a certain threshold $\epsilon > 0$, *i.e.*, quit when

$$|f_0(x^{(k+1)}, y^{(k+1)}) - f_0(x^{(k)}, y^{(k)})| < \epsilon. \tag{3.5}$$

As a small variation, one may also use the difference between the optimal values of the problems (3.1) and (3.2) in one iteration as the criterion, *i.e.*, quit when

$$|f_0(x^{(k+1)}, y^{(k+1)}) - f_0(x^{(k+1)}, y^{(k)})| < \epsilon. \tag{3.6}$$

In practice, there is not much difference between using (3.5) and (3.6) as the termination criteria of the ACS procedure, except that using the latter does not require storing the

objective value of (2.1) from the previous iteration. Other choices include limiting the maximum number of iterations, or stopping when the changes of the optimization variables are below certain thresholds, *e.g.*, quit when

$$\max\{\|x^{(k+1)} - x^{(k)}\|_2,\ \|y^{(k+1)} - y^{(k)}\|_2\} < \epsilon,$$

for some $\epsilon > 0$.

3.2 Proximal regularizations

Directly solving the subproblems (3.1) and (3.2) in the algorithm 3.1 may lead to numerical problems or slow convergence in practice. One possible reason for this is that the biconvex objective function f_0 can be very 'flat' in some region along certain directions when one block of variables is fixed (an example is shown in figure 2, in the region where x and y are close to zero), where the convex solver for subproblems may have difficulty getting sufficient progress in minimizing the objective along the descent direction. A common technique to alleviate these issues is to add proximal regularization terms to the objective functions of the subproblems. Specifically, in the $(k+1)$th iteration of algorithm 3.1, instead of performing (3.3) and (3.4), we try to perform the following updates:

$$\begin{array}{rcll} x^{(k+1)} & := & \operatorname{argmin}_{x\in\mathcal{X}} & f_0(x, y^{(k)}) + \lambda\|x - x^{(k)}\|_2^2 \\ & & \text{subject to} & f_i(x, y^{(k)}) \le 0, \quad i = 1, \ldots, m \\ & & & h_i(x, y^{(k)}) = 0, \quad i = 1, \ldots, p, \end{array} \tag{3.7}$$

and

$$\begin{array}{rcll} y^{(k+1)} & := & \operatorname{argmin}_{y\in\mathcal{Y}} & f_0(x^{(k+1)}, y) + \lambda\|y - y^{(k)}\|_2^2 \\ & & \text{subject to} & f_i(x^{(k+1)}, y) \le 0, \quad i = 1, \ldots, m \\ & & & h_i(x^{(k+1)}, y) = 0, \quad i = 1, \ldots, p, \end{array} \tag{3.8}$$

where $\lambda \ge 0$ is the regularization parameter. When $\lambda = 0$, the updates (3.7) and (3.8) reduce to (3.3) and (3.4), respectively. When $\lambda > 0$, the proximal terms in (3.7) and (3.8) can be interpreted as adding soft trust region constraints to the respective optimization variables, which penalize large changes of the variables between two consecutive iterations. By choosing sufficiently large λ, the corresponding optimization problems in (3.7) and (3.8) become strongly convex, which can help improve numerical stability in practice [BPC+11, PB14]. It is also observed that adding the additional proximal regularizers can sometimes lead to better final points, *i.e.*, points with lower objective values [SDU+17], compared to those from the original ACS procedure.

3.3 Initialization

Note that algorithm 3.1 requires a feasible starting point $(x^{(0)}, y^{(0)}) \in \mathcal{D}$, since otherwise, one of the subproblems (3.1) or (3.2) may be infeasible right at the first iteration. Formally, the corresponding feasibility problem of a biconvex problem of the form (2.1) can be written as

$$\begin{array}{ll} \text{find} & (x, y) \\ \text{subject to} & f_i(x, y) \le 0, \quad i = 1, \ldots, m \\ & h_i(x, y) = 0, \quad i = 1, \ldots, p \end{array} \tag{3.9}$$

with variables $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Solving (3.9) directly can be as hard as solving the original biconvex problem (2.1), and, actually, it can even be NP-hard [TÖ95]. Here, we consider the following heuristic via relaxation to find a feasible starting point:

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s + \|t\|_1 \\ \text{subject to} & s \succeq 0 \\ & f_i(x,y) \leq s_i, \quad i = 1,\ldots,m \\ & h_i(x,y) = t_i, \quad i = 1,\ldots,p, \end{array} \tag{3.10}$$

where $x \in \mathcal{X}$, $y \in \mathcal{Y}$ are the optimization variables, and $s \in \mathbf{R}^m$ and $t \in \mathbf{R}^p$ are the slack variables to relax the constraints in (3.9). The subscripts i of s_i and t_i denote the ith entry of the vectors s and t, respectively. Note that the relaxed biconvex feasibility problem (3.10) is always feasible, since by choosing sufficiently large s and t, all constraints can be satisfied. To solve (3.10), we can again use the ACS procedure introduced in §3.1. The full algorithm is given below.

Algorithm 3.2 *Finding a feasible starting point.*

given a starting point $(x^{(0)}, y^{(0)}) \in \mathcal{X} \times \mathcal{Y}$.
$k := 0$.
repeat

1. $(x^{(k+1)}, s^\star, t^\star) := \underset{\substack{x\in\mathcal{X} \\ s\in\mathbf{R}^m, t\in\mathbf{R}^p}}{\operatorname{argmin}} \left\{ \mathbf{1}^T s + \|t\|_1 \;\middle|\; \begin{array}{ll} s \succeq 0 \\ f_i(x, y^{(k)}) \leq s_i, & i = 1,\ldots,m \\ h_i(x, y^{(k)}) = t_i, & i = 1,\ldots,p \end{array} \right\}.$
   quit with $(x^{(k+1)}, y^{(k)})$ **if** $\mathbf{1}^T s^\star + \|t^\star\|_1 = 0$.
2. $(y^{(k+1)}, s^\star, t^\star) := \underset{\substack{y\in\mathcal{Y} \\ s\in\mathbf{R}^m, t\in\mathbf{R}^p}}{\operatorname{argmin}} \left\{ \mathbf{1}^T s + \|t\|_1 \;\middle|\; \begin{array}{ll} s \succeq 0 \\ f_i(x^{(k+1)}, y) \leq s_i, & i = 1,\ldots,m \\ h_i(x^{(k+1)}, y) = t_i, & i = 1,\ldots,p \end{array} \right\}.$
   quit with $(x^{(k+1)}, y^{(k+1)})$ **if** $\mathbf{1}^T s^\star + \|t^\star\|_1 = 0$.
3. $k := k + 1$.

until maximum iterations are reached.

Different from algorithm 3.1, initiating algorithm 3.2 only requires a point in $\mathcal{X} \times \mathcal{Y}$, which is usually easy to obtain. If algorithm 3.2 quit with $\mathbf{1}^T s^\star + \|t^\star\|_1 = 0$ in some iteration, then the returned point is a feasible point of the original biconvex problem (2.1), which can then be used as a starting point for the ACS procedure in algorithm 3.1. However, we must note that there is no guarantee that the algorithm 3.2 will find a feasible point for any instance of the biconvex problem (2.1), even if such a point exists. In practice, as a generic practical method, this approach seems to work quite well.

3.4 Infeasible start

Now we integrate the relaxation, which transforms the biconvex feasibility problem (3.9) into (3.10), directly into the original biconvex problem (2.1), so that the ACS procedure can be applied even when starting from an infeasible point. Let $s \in \mathbf{R}^m$ and $t \in \mathbf{R}^p$ be slack variables to relax the inequality and equality constraints of (2.1), respectively. We consider

the following relaxed biconvex optimization problem:

$$\begin{array}{ll} \text{minimize} & f_0(x,y) + \nu(\mathbf{1}^T s + \|t\|_1) \\ \text{subject to} & s \succeq 0 \\ & f_i(x,y) \leq s_i, \quad i = 1,\ldots,m \\ & h_i(x,y) = t_i, \quad i = 1,\ldots,p, \end{array} \tag{3.11}$$

where $\nu > 0$ is a penalty parameter. The problem (3.11) is always feasible, since by choosing sufficiently large s and t, all constraints can be satisfied. Moreover, if the original biconvex problem (2.1) is feasible, then for sufficiently large ν, applying ACS to (3.11) will yield a final point $(x^\star, y^\star, s^\star, t^\star)$, such that $\mathbf{1}^T s^\star + \|t^\star\|_1 = 0$, *i.e.*, $(x^\star, y^\star)$ is feasible and partially optimal for (2.1) [NW06, SDU+17]. The full algorithm is given as follows.

Algorithm 3.3 *Infeasible start alternate convex search.*

given a starting point $(x^{(0)}, y^{(0)}) \in \mathcal{X} \times \mathcal{Y}$ and sufficiently large $\nu > 0$.
$k := 0$.
repeat

1. $x^{(k+1)} := \underset{x \in \mathcal{X}}{\operatorname{argmin}} \left\{ \begin{array}{l|l} f_0(x, y^{(k)}) & s \in \mathbf{R}^m, \quad t \in \mathbf{R}^p, \quad s \succeq 0 \\ \quad +\nu(\mathbf{1}^T s + \|t\|_1) & f_i(x, y^{(k)}) \leq s_i, \quad i = 1,\ldots,m \\ & h_i(x, y^{(k)}) = t_i, \quad i = 1,\ldots,p \end{array} \right\}.$
2. $y^{(k+1)} := \underset{y \in \mathcal{Y}}{\operatorname{argmin}} \left\{ \begin{array}{l|l} f_0(x^{(k+1)}, y) & s \in \mathbf{R}^m, \quad t \in \mathbf{R}^p, \quad s \succeq 0 \\ \quad +\nu(\mathbf{1}^T s + \|t\|_1) & f_i(x^{(k+1)}, y) \leq s_i, \quad i = 1,\ldots,m \\ & h_i(x^{(k+1)}, y) = t_i, \quad i = 1,\ldots,p \end{array} \right\}.$
3. $k := k + 1$.

until stopping criteria is reached.

Compared to algorithm 3.1, the infeasible start ACS procedure in algorithm 3.3 can start from any point in $\mathcal{X} \times \mathcal{Y}$, and hence, the initialization step via, *e.g.*, algorithm 3.2, can be avoided. The same termination criteria as those discussed in §3.1 can still be used for algorithm 3.3. However, we must note that there is no guarantee that the final point returned by algorithm 3.3 is feasible for the original biconvex problem (2.1), even if such a point exists, since the penalty parameter ν may not be sufficiently large. In practice, the value of ν can be selected in an ad hoc manner, *i.e.*, one may try to increase ν and resolve (3.11) if the final point returned by algorithm 3.3 is still infeasible for (2.1). Finally, the proximal regularizations as in (3.7) and (3.8) can be readily integrated into the subproblems in the algorithm 3.3 to improve numerical stability.

4 Disciplined biconvex programming

We now present the DBCP biconvex ruleset for modeling biconvex optimization problems in a way that the biconvexity is easily verified by construction.

DCP convex ruleset. The DBCP ruleset is heavily based on the DCP convex ruleset, which consists of a library of convex atomic functions, and a convex syntax ruleset that prescribes how these atomic functions may be composed to express (more complex) convex

optimization problems [GBY06]. Specifically, all functions in a DCP-compliant problem must be formed as an *expression* consisting of variables, constants or parameters, and atomic functions. The sign, curvature, and monotonicity of each DCP expression can be determined recursively from those of its constituent parts, based on which the convexity of the overall problem can be verified by checking whether each function in the problem satisfies the DCP composition rules [GBY06, BV04]. We make an observation that is critical for extending DCP to DBCP: The basic arithmetic properties, in particular, the composition property, of biconvex functions, as discussed in §2.2, are compatible with the DCP ruleset. In other words, an optimization problem formed by verifiable biconvex expressions according to the DCP ruleset is still biconvex.

DBCP product rule. To construct disciplined biconvex expressions, we first note that the product of expressions that are both nonconstant is prohibited in DCP, since the convexity of such expressions cannot be determined in general [GBY06]; however, most biconvex expressions that appear in practice are constructed through variable multiplications. Hence, we introduce the following product rule for DBCP:

1. A valid DBCP convex product expression should include variables in both the left-hand and right-hand expressions, and should be one of the following forms:

$$\begin{gathered} \textit{affine} * \textit{affine} \\ \textit{affine-nonneg} * \textit{convex} \quad \text{or} \quad \textit{affine-nonpos} * \textit{concave} \\ \textit{convex-nonneg} * \textit{convex-nonneg} \quad \text{or} \quad \textit{concave-nonpos} * \textit{concave-nonpos} \end{gathered}$$

   The *nonneg* and *nonpos* qualifiers indicate that the expression is known to be nonnegative or nonpositive, respectively.

2. There exists no loop in the variable interaction graph of the overall expression, where the edge between two variables indicates that they appear on different sides in a product expression as described in the above rule.

Note that the above DBCP product rules do not include constant/parameter-variable multiplications, since such expressions are already covered by the DCP product-free ruleset [GBY06]. The second rule is to prevent expressions like `x * y`, `y * z`, and `z * x` from appearing simultaneously in the same optimization problem, which would lead to cyclic interactions between the variables `x`, `y`, and `z`, where the biconvexity of the overall expression cannot be guaranteed. According to the definition and basic properties of the biconvex functions listed in §2.2, expressions formed according to the above DBCP product rule are guaranteed to be biconvex.

5 Implementation

In this section, we introduce our Python implementation of the DBCP framework, as an extension to `CVXPY` [DB16, AVDB18]. The corresponding open-source package, named `dbcp`, is available at

https://github.com/nrgrp/dbcp.

Specifying biconvex problems. Users can define their optimization variables, objective functions, and constraints using the standard `CVXPY` syntax, and a biconvex problem is constructed using:

```
prob = BiconvexProblem(obj, [x_var, y_var], constraints)
```

The argument `obj` is a DBCP-compliant biconvex expression representing the objective function, `x_var` and `y_var` are lists of the biconvex optimization variables, and `constraints` is a list of DBCP-compliant biconvex constraints. The arguments `x_var` and `y_var` define the variable partition for the biconvex problem, so that each group is fixed when optimizing over the other group during the ACS procedure. Note that it is not necessary to include all variables that appear in the problem in `x_var` and `y_var`; those variables that are DCP-compliant, *i.e.*, those related to convex expressions, can be left out, since they do not have to be fixed in any step of the ACS procedure. For example, to specify the biconvex problem

$$\begin{array}{ll} \text{minimize} & \|XY + Z - A\|_F \\ \text{subject to} & \|Z\|_F \leq 1, \end{array}$$

where $X \in \mathbf{R}^{m\times k}$, $Y \in \mathbf{R}^{k\times n}$, $Z \in \mathbf{R}^{m\times n}$ are optimization variables, and $A \in \mathbf{R}^{m\times n}$ is problem data, one may use the following code:

```
import cvxpy as cp
from dbcp import BiconvexProblem

X = cp.Variable((m, k))
Y = cp.Variable((k, n))
Z = cp.Variable((m, n))

obj = cp.Minimize(cp.norm(X @ Y + Z - A, 'fro'))
constraints = [cp.norm(Z, 'fro') <= 1]
prob = BiconvexProblem(obj, [[X], [Y]], constraints)
```

Solving a biconvex problem. To solve the specified biconvex problem `prob`, one may simply call `prob.solve()`. There are many optional arguments that can be passed to `prob.solve()` to customize the solution procedure, where most of them are directly inherited from `CVXPY` for the configuration of the convex solvers used to solve the subproblems. In addition to these standard arguments, `dbcp` also provides several specific options for controlling the ACS procedure. One of the most important ones is `lbd`, which sets a value for the proximal regularization parameter λ in (3.7) and (3.8). When `lbd` is zero, no proximal regularization is added, *i.e.*, the original ACS procedure (algorithm 3.1) is used; if `lbd` is positive, then the proximal regularized updates in (3.7) and (3.8) are used in replacement of the original updates (3.3) and (3.4). It is recommended for the user to specify (feasible) initial values for all optimization variables before calling `prob.solve()`. Otherwise, `dbcp` will try to generate random initial values for the unspecified variables from a standard normal distribution, which is not guaranteed to work well for all problems. Every time the `prob.solve()` method is called, `dbcp` first checks the feasibility of the current initial point. If the initial point is infeasible, then algorithm 3.2 will be used to try to find a feasible starting point, based on the current values of the variables, before launching the ACS procedure. We implement the stopping criteria (3.6) as the termination condition for the ACS procedure, with a default threshold value of $\epsilon = 10^{-6}$, which can be modified by passing the argument `gap_tolerance` to `prob.solve()`.

Solving with infeasible starts. We provide another problem class for solving biconvex problems by directly solving the relaxed biconvex problem (3.11) via the infeasible start ACS (algorithm 3.3), which can be specified using:

```
prob = BiconvexRelaxProblem(obj, [x_var, y_var], constraints)
```

The usage of `BiconvexRelaxProblem` is mostly the same as that of `BiconvexProblem`, except that there is one additional argument `nu` for the `prob.solve()` method, which sets the penalty parameter ν in (3.11). Calling `prob.solve()` for a `BiconvexRelaxProblem` instance will directly launch the infeasible start ACS procedure (algorithm 3.3) from the current variable values (if not specified, random initial values will be generated as in `BiconvexProblem`). Note that although the problem (3.11) is always feasible, there is no guarantee that the final point returned by `prob.solve()` (under the ACS stopping criteria (3.6) with default $\epsilon = 10^{-6}$) is feasible for the original biconvex problem (2.1). To monitor the feasibility progress of the algorithm, the value of the total slack $\mathbf{1}^T s + \|t\|_1$ is reported after each ACS iteration. In the case that the final point is still infeasible with nonzero total slack, the user may try to increase the value of `nu` and resolve the problem.

Verification of biconvexity. When a biconvex problem is specified as an instance of either `BiconvexProblem` or `BiconvexRelaxProblem`, `dbcp` will automatically verify the biconvexity of the objective function and all constraints according to the DBCP ruleset introduced in §4. The user can check whether a problem is DBCP-compliant by calling `prob.is_dbcp()`, which returns `True` if the problem is a valid DBCP biconvex problem, and `False` otherwise. If a user is trying to solve a non-DBCP-compliant biconvex problem, `dbcp` will raise an error directly after the `prob.solve()` method is called.

Generalized inequality constraints. Lastly, we should mention that although all the previous discussion about solving and finding a feasible initial point for biconvex problems is based on standard inequality constraints as shown in (2.1), the `dbcp` package is implemented in a way that natively supports biconvex problems with generalized inequality constraints, such as second-order cone constraints and positive semidefinite constraints. Interested readers may refer to §A for more theoretical details about how these constraints are handled in the backend, while practitioners can directly use the standard `CVXPY` syntax for specifying generalized inequality constraints when defining biconvex problems using `dbcp`. The appearance of such constraints will be automatically detected by `dbcp`, and the appropriate adaptations will be made in the solving procedure without any extra effort from the user.

6 Examples

In this section, we present several numerical examples of specifying and solving biconvex optimization problems that appear frequently in practice using the `dbcp` package. We only show snippets of the code for defining and solving the biconvex problems, while omitting the data generation and result visualization parts for brevity. Interested readers can find the full code examples at

https://github.com/nrgrp/dbcp.

6.1 Nonnegative matrix factorization

We start with a basic nonnegative matrix factorization problem. Suppose that we are given a matrix $A \in \mathbf{R}^{m\times n}$, and are interested in finding two nonnegative matrices $X \in \mathbf{R}^{m\times k}$ and $Y \in \mathbf{R}^{k\times n}$, such that $A \approx XY$. This can be formulated as the following biconvex optimization problem:

$$\begin{array}{ll} \text{minimize} & \|XY - A\|_F^2 \\ \text{subject to} & X_{ij} \geq 0, \quad i = 1, \ldots, m, \quad j = 1, \ldots, k \\ & Y_{ij} \geq 0, \quad i = 1, \ldots, k, \quad j = 1, \ldots, n \end{array}$$

with variables X and Y. To specify this problem using `dbcp`, one may use the following code:

```
X = cp.Variable((m, k), nonneg=True)
Y = cp.Variable((k, n), nonneg=True)

obj = cp.Minimize(cp.sum_squares(X @ Y - A))
prob = BiconvexProblem(obj, [[X], [Y]])
```

In this example, we set $m = 5$, $n = 10$, and $k = 5$, and generate the data matrix A as the product of two random matrices in $\mathbf{R}^{m\times k}$ and $\mathbf{R}^{k\times n}$ from the standard normal distribution. After calling `prob.solve()`, we obtain a final point with an objective value around 6×10^{-6}, which indicates that the original matrix A is well approximated by the product of the recovered nonnegative matrices X and Y.

6.2 Bilinear logistic regression

We consider a bilinear logistic regression problem for binary classification [DCP07]. Suppose that we are given a dataset (X_i, y_i), $i = 1, \ldots, m$, where each sample consists of a feature matrix $X_i \in \mathbf{R}^{n\times k}$ and a binary label $y_i \in \{0, 1\}$. Our goal is to construct a bilinear classifier $\hat{y} = 1$ if $\mathbf{tr}(U^T X V) > 0$, and $\hat{y} = 0$ otherwise, where $U \in \mathbf{R}^{n\times r}$ and $V \in \mathbf{R}^{k\times r}$ are the bilinear logistic regression coefficients with a predefined (maximum) rank r, and $\mathbf{tr}(M)$ denotes the trace of some square matrix M. To fit a bilinear logistic regression model to the dataset, we would like to solve the following bilinear maximum likelihood estimation problem:

$$\text{maximize} \quad \sum_{i=1}^m y_i \, \mathbf{tr}(U^T X_i V) - \log(1 + \exp(\mathbf{tr}(U^T X_i V)))$$

with variables U and V. This problem can be specified using `dbcp` as follows:

```
U = cp.Variable((n, r))
V = cp.Variable((k, r))

obj = 0
for X, y in zip(Xs, ys):
    obj += cp.sum(
        cp.multiply(y, cp.trace(U.T @ X @ V))
            - cp.logistic(cp.trace(U.T @ X @ V))
    )
prob = BiconvexProblem(cp.Maximize(obj), [[U], [V]])
```

In this example, we set $m = 300$, $n = 20$, $k = 10$, and $r = 5$. The dataset (X_i, y_i), $i = 1, \ldots, m$, is generated synthetically using the `make_classification` function from the

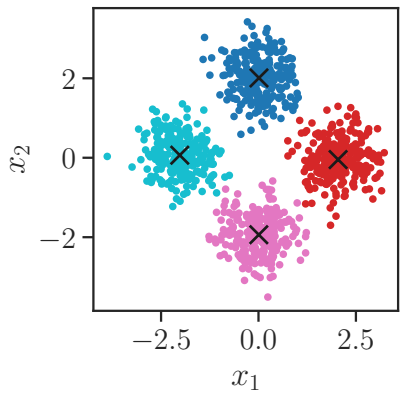

Figure 3 Results of the k-means clustering example.

`scikit-learn` library [PVG+11]. After calling `prob.solve()` with the proximal regularization weight `lbd=1` and the termination threshold `gap_tolerance=1e-4`, we obtain a final point with an objective value -5×10^{-3}.

6.3 k-means clustering

Suppose that we are given a set of data points $x_i \in \mathbf{R}^n$, $i = 1, \ldots, m$, and we would like to cluster them into k groups, using the k-means clustering method. This problem can be formulated as the following biconvex optimization problem [ZYHB25]:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m z_i^T(\|\bar{x}_1 - x_i\|_2^2, \ldots, \|\bar{x}_k - x_i\|_2^2) \\ \text{subject to} & 0 \preceq z_i \preceq \mathbf{1}, \quad \mathbf{1}^T z_i = 1, \quad i = 1, \ldots, m \end{array} \tag{6.1}$$

with variables $\bar{x}_i \in \mathbf{R}^n$, $i = 1, \ldots, k$, and $z_i \in \mathbf{R}^k$, $i = 1, \ldots, m$. We can interpret the variables in the problem (6.1) as follows: The variables $\bar{x}_1, \ldots, \bar{x}_k$ represent the cluster centroids, and each variable z_i is a soft assignment vector for the data point x_i, where the jth entry of z_i indicates the probability that the sample x_i belongs to cluster j. Then, the objective function in (6.1) represents the total within-cluster variance, which we would like to minimize. To specify the problem (6.1) using `dbcp`, one may use the following code:

```
xbars = cp.Variable((k, n))
zs = cp.Variable((m, k), nonneg=True)

obj = cp.sum(cp.multiply(zs, cp.vstack([
    cp.sum(cp.square(xs - c), axis=1) for c in xbars
]).T))
constr = [zs <= 1, cp.sum(zs, axis=1) == 1]
prob = BiconvexProblem(cp.Minimize(obj), [[xbars], [zs]], constr)
```

We generate a synthetic dataset of $m = 1000$ points in $\mathbf{R}^2$ using the `make_blobs` function of `scikit-learn`, and set the number of clusters $k = 4$ with the ground truth centroids $(0, 2)$, $(0, -2)$, $(2, 0)$, and $(-2, 0)$. The k-means clustering results based on the formulation (6.1) after calling `prob.solve()` are shown in figure 3, where the colors indicate the clus-

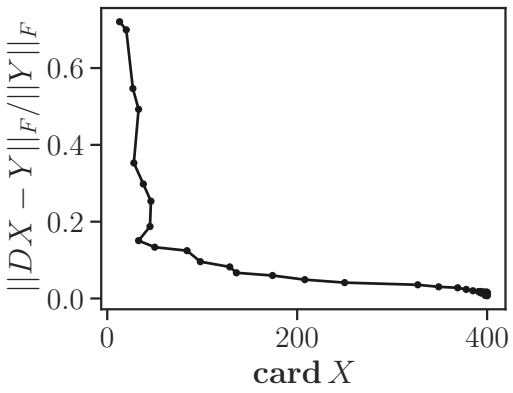

Figure 4 Trade off curve of the sparse dictionary learning example.

ter assignments of the data points, and the black crosses represent the recovered cluster centroids.

6.4 Dictionary learning

We consider the sparse dictionary learning problem [AEB06], which aims to find a dictionary matrix $D \in \mathbf{R}^{m\times k}$ and a sparse code matrix $X \in \mathbf{R}^{k\times n}$, such that the data matrix $Y \in \mathbf{R}^{m\times n}$ can be well approximated by their product DX, while the matrix X is sparse and the matrix D has bounded Frobenius norm. The dictionary learning problem can be formulated as the following biconvex optimization problem:

$$\begin{array}{ll} \text{minimize} & \|DX - Y\|_F^2 + \alpha\|X\|_1 \\ \text{subject to} & \|D\|_F \leq \beta \end{array} \tag{6.2}$$

with variables D and X, where $\alpha > 0$ is the sparsity regularization parameter, and $\beta > 0$ is the bound on the Frobenius norm of the dictionary matrix. To specify this problem using `dbcp`, one may use the following code:

```
D = cp.Variable((m, k))
X = cp.Variable((k, n))

obj = cp.Minimize(cp.sum_squares(D @ X - Y) + alpha * cp.norm1(X))
prob = BiconvexProblem(obj, [[D], [X]], [cp.norm(D,'fro') <= beta])
```

In this example, we set $m = 10$, $n = 20$, $k = 20$, $\beta = 1$, and generate the data matrix $Y \in \mathbf{R}^{m\times n}$ from the standard normal distribution. The problem (6.2) is then solved for different values of the sparsity regularization parameter α ranging from 10^{-5} to 1. For each value of α, the values of the problem variables D and X are reinitialized randomly before `prob.solve()` is called. Figure 4 shows the trade off curve between the relative approximation error $||DX - Y||_F/||Y||_F$ and the cardinality (*i.e.*, the number of nonzero entries) of X.

6.5 Blind deconvolution

Blind deconvolution is a technique used to recover some sharp signal or image from a blurred observation when the blur itself is unknown [CVR14]. It jointly estimates both the original signal and the blur kernel, with some prior knowledge about their structures. Suppose that we are given a data vector $d \in \mathbf{R}^{m+n-1}$, which is the convolution of an unknown sparse signal $x \in \mathbf{R}^n$ and an unknown smooth vector $y \in \mathbf{R}^m$ with bounded ℓ_∞-norm (*i.e.*, bounded largest entry). Additionally, we have the prior knowledge that both the vectors x and y are nonnegative. The corresponding blind deconvolution problem can be formulated as the following biconvex optimization problem:

$$\begin{array}{ll} \text{minimize} & \|x \otimes y - d\|_2^2 + \alpha_{\text{sp}}\|x\|_1 + \alpha_{\text{sm}}\|Dy\|_2^2 \\ \text{subject to} & x \succeq 0, \quad y \succeq 0 \\ & \|y\|_\infty \leq \beta \end{array}$$

with variables x and y, where $\alpha_{\text{sp}}, \alpha_{\text{sm}} > 0$ are the regularization parameters for the sparsity of x and smoothness of y, respectively, and $\beta > 0$ is the bound on the ℓ_∞-norm of the vector y. The matrix $D \in \mathbf{R}^{(m-1)\times m}$ is the first-order difference operator, given by

$$D = \begin{bmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{bmatrix} \in \mathbf{R}^{(m-1)\times m},$$

so that Dy computes the vector of successive differences of y. The convolution $x \otimes y$ of the vectors x and y is given by

$$(x \otimes y)_k = \sum_{i+j=k} x_i y_j, \quad k = 1, \ldots, m+n-1.$$

To specify this problem using `dbcp`, one may use the following code:

```
x = cp.Variable(n, nonneg=True)
y = cp.Variable(m, nonneg=True)

obj = cp.sum_squares(conv(x, y) - d) + alpha_sp * cp.norm1(x) +
    alpha_sm * cp.sum_squares(cp.diff(y))
constr = [cp.norm(y, "inf") <= beta]
prob = BiconvexProblem(cp.Minimize(obj), [[x], [y]], constr)
```

In this example, we consider the vectors $x \in \mathbf{R}^{120}$, $y \in \mathbf{R}^{40}$, and the ℓ_∞-norm bound $\beta = 1$. The ground truth vectors were generated according to a similar example used by Shen *et al.* [SDU+17]. The ground truth and the recovered signals after calling `prob.solve()` with regularization parameters $\alpha_{\text{sp}} = 0.1$ and $\alpha_{\text{sm}} = 0.2$ are shown in figure 5.

6.6 Fitting input-output hidden Markov models

We consider the fitting problem of a logistic input-output hidden Markov model (IO-HMM) to some dataset [JAP24]. Suppose that we are given a dataset $(x(t), y(t))$, $t = 1, \ldots, m$, where each sample consists of an input feature vector $x(t) \in \mathbf{R}^n$ and an output label

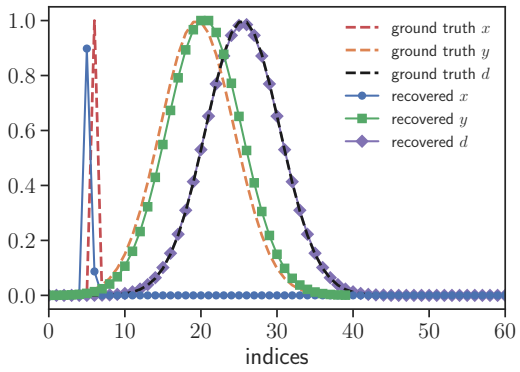

Figure 5 Results of the blind deconvolution example.

$y(t) \in \{0,1\}$, generated from a K-state IO-HMM, according to the following procedure: Let $\hat{z}(t) \in \{1,\ldots,K\}$, $t = 1,\ldots,m$, be the state label of the IO-HMM with initial state distribution $p_{\text{init}} \in \mathbf{R}^K$ with $\mathbf{1}^T p_{\text{init}} = 1$ and transition matrix $P_{\text{tr}} \in \mathbf{R}^{K\times K}$ with $P_{\text{tr}}\mathbf{1} = \mathbf{1}$. At the time step t, the state label $\hat{z}(t)$ is sampled according to

$$\hat{z}(t) \sim \begin{cases} \text{Cat}(p_{\text{init}}) & t = 0 \\ \text{Cat}(p_{\hat{z}(t-1)}) & t > 0, \end{cases}$$

where the vector $p_{\hat{z}(t-1)} \in \mathbf{R}^K$ denotes the $\hat{z}(t-1)$th row of the matrix P_{tr}, and $\text{Cat}(p)$ denotes the categorical distribution with p being the vector of category probabilities. Then, given the feature vector $x(t) \in \mathbf{R}^n$, the output $y(t) \in \{0,1\}$ of this IO-HMM at time step t is then generated from a logistic model, *i.e.*,

$$\mathbf{prob}(y(t) = 1) = \frac{1}{1 + \exp(-x(t)^T\theta_{\hat{z}(t)})},$$

where $\theta_{\hat{z}(t)} \in \{\theta_1,\ldots,\theta_K\} \subseteq \mathbf{R}^n$ is the coefficient.

Given the dataset $(x(t), y(t))$, $t = 1,\ldots,m$, we are interested in recovering the transition matrix P_{tr}, the model parameters $\theta_1,\ldots,\theta_K$, and the unobserved state labels $\hat{z}(1),\ldots,\hat{z}(m)$. Noticing that the transition matrix P_{tr} can be easily estimated from the state labels $\hat{z}(t)$, $t = 1,\ldots,m$, we consider the following biconvex optimization problem for fitting the IO-HMM [ZYHB25]:

$$\begin{array}{ll} \text{minimize} & -\sum_{t=1}^m z(t)^T \Big(y(t)x(t)^T\theta_k - \log(1+\exp(x(t)^T\theta_k))\Big)_{k=1}^K \\ & \quad +\alpha_\theta \sum_{k=1}^K \|\theta_k\|_2^2 + \alpha_z \sum_{t=1}^{m-1} D_{\text{kl}}(z(t), z(t+1)) \\ \text{subject to} & 0 \preceq z(t) \preceq \mathbf{1}, \quad \mathbf{1}^T z(t) = 1, \quad t = 1,\ldots,m \\ & \theta_k \in \mathcal{C}_k, \quad k = 1,\ldots,K, \end{array} \tag{6.3}$$

where the optimization variables are $\theta_k \in \mathbf{R}^n$, $k = 1,\ldots,K$, and $z(t) \in \mathbf{R}^K$, $t = 1,\ldots,m$. Note that the variable $z(t)$ is a soft assignment vector for the hidden state label $\hat{z}(t)$, where the kth entry of $z(t)$ indicates the probability of the state being k at the time step t, and $\hat{z}(t)$ can be estimated as the index of the largest entry of $z(t)$ after solving the problem

(6.3). Each component of the problem (6.3) can be interpreted as follows: The first term in the objective function is the negative log-likelihood of the observed data under the IO-HMM model, given the state assignment probabilities $z(t)$, $t = 1, \ldots, m$, and the model parameters θ_k, $k = 1, \ldots, K$. The second term is a Tikhonov regularization on the model parameters θ_k, with the regularization parameter $\alpha_\theta > 0$. The third term is a temporal smoothness regularization on the state assignment probabilities, where $D_{\mathrm{kl}}(p, q)$ denotes the Kullback-Leibler divergence between two probability distributions p and q, and $\alpha_z > 0$ is the corresponding regularization parameter. The constraints on the variables $z(t)$, $t = 1, \ldots, m$, ensure that they are valid probability distributions. The sets $\mathcal{C}_k \subseteq \mathbf{R}^n$, $k = 1, \ldots, K$, are closed nonempty convex sets that encode potential prior knowledge about the model parameters θ_k.

In this example, we generate $m = 1800$ data samples from an IO-HMM with $K = 3$ hidden states and input feature dimension $n = 2$. The feature vector for each sample is generated according to

$$x(t) \sim (\mathcal{U}(-5, 5),\ 1),$$

where $\mathcal{U}(a, b)$ denotes a uniform distribution over the interval $[a, b]$, and the second entry of $x(t)$ is always 1 to account for the bias term. The ground truth coefficients, initial state distribution, and transition matrix are given by

$$\theta_1 = (-1, 0), \quad \theta_2 = (2, 6), \quad \theta_3 = (2, -6),$$
$$p_{\mathrm{init}} = (1, 0, 0), \quad P_{\mathrm{tr}} = \begin{bmatrix} 0.95 & 0.025 & 0.025 \\ 0.025 & 0.95 & 0.025 \\ 0.025 & 0.025 & 0.95 \end{bmatrix}.$$

To fully specify the problem (6.3), it is assumed that we are given the following prior knowledge about the coefficients:

$$\theta_{1,1} \leq 0, \quad \theta_{2,1} \geq 0, \quad \theta_{3,1} \geq 0, \quad \theta_{2,2} \geq \theta_{3,2},$$

where $\theta_{i,j}$ denotes the jth entry of the vector θ_i. The corresponding code snippets is as follows:

```
thetas = cp.Variable((K, n))
zs = cp.Variable((m, K), nonneg=True)

rs = [
    -cp.multiply(ys, xs @ thetas[k]) + cp.logistic(xs @ thetas[k])
    for k in range(K)
]
obj = cp.Minimize(
    cp.sum(cp.multiply(zs, cp.vstack(rs).T))
    + alpha_theta * cp.sum_squares(thetas)
    + alpha_z * cp.sum(cp.kl_div(zs[:-1], zs[1:])))
constr = [
    thetas[0][0] <= 0,
    thetas[1][0] >= 0,
    thetas[2][0] >= 0,
    thetas[1][1] >= thetas[2][1],
    zs <= 1, cp.sum(zs, axis=1) == 1
]
```

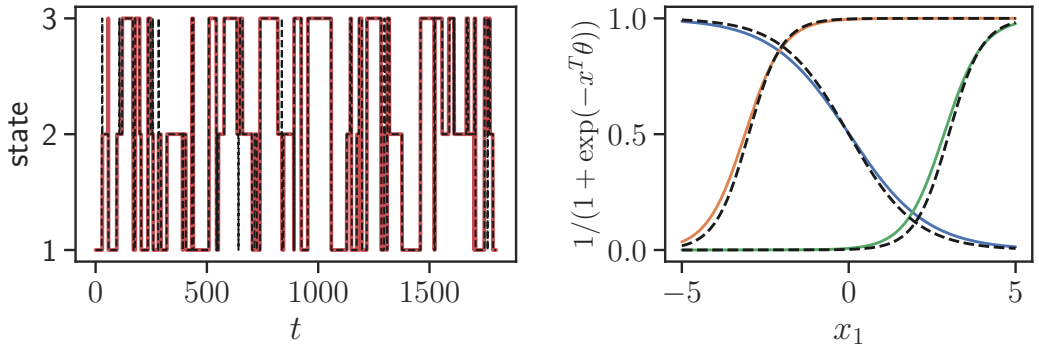

Figure 6 Results of the IO-HMM example. The ground truth state labels (*left*) and the response curve of the output probability (*right*) are shown in black dashed lines, whereas the corresponding estimations are shown in solid lines.

```
19
20 prob = BiconvexRelaxProblem(obj, ([zs], [thetas]), constr)
```

Note that for this example, we use the `BiconvexRelaxProblem` class to solve the relaxed biconvex problem (3.11) via the infeasible start ACS procedure (algorithm 3.3).

The problem (6.3) is then solved with regularization parameters $\alpha_\theta = 0.1$ and $\alpha_z = 2$. The arguments when calling the `prob.solve()` method are set to `nu=1e2`, `lbd=0.1`, and `gap_tolerance=1e-3`. The final total slack is around 4.21×10^{-8}, indicating that the returned point is feasible (within numerical roundoff error) for the original biconvex problem (6.3). Figure 6 shows the estimated state label for each sample and the response curve corresponding to each state. The estimated transition matrix of the Markov chain is

$$\begin{bmatrix} 0.96 & 0.02 & 0.02 \\ 0.03 & 0.95 & 0.02 \\ 0.01 & 0.02 & 0.97 \end{bmatrix}.$$

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”.

A Biconvex problem with generalized inequality constraints

Consider a biconvex optimization problem with generalized inequality constraints of the form

$$\begin{array}{ll} \text{minimize} & f_0(x,y) \\ \text{subject to} & f_i(x,y) \preceq_{\mathcal{K}_i} 0, \quad i=1,\ldots,m \\ & h_i(x,y)=0, \quad i=1,\ldots,p, \end{array} \tag{A.1}$$

where $f_0 \colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$ is a biconvex objective, $h_i \colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}$, $i=1,\ldots,p$, are biaffine equality constraint functions, $f_i \colon \mathcal{X} \times \mathcal{Y} \to \mathbf{R}^{q_i}$, $i=1,\ldots,m$, are biconvex inequality constraint functions with respect to proper cones $\mathcal{K}_i \subseteq \mathbf{R}^{q_i}$, $i=1,\ldots,m$. The ACS procedure (algorithm 3.1) and the proximal regularizations given by (3.7) and (3.8) can be directly applied to the generalized inequality constrained biconvex problem (A.1) without modification. However, the initialization procedure (algorithm 3.2) and the infeasible start ACS procedure (algorithm 3.3) need to be slightly modified to accommodate the generalized inequality constraints. Specifically, the relaxed biconvex feasibility problem (3.10) is changed to

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T s + \|t\|_1 \\ \text{subject to} & s \succeq 0 \\ & f_i(x,y) \preceq_{\mathcal{K}_i} s_i e_{\mathcal{K}_i}, \quad i=1,\ldots,m \\ & h_i(x,y)=t_i, \quad i=1,\ldots,p, \end{array} \tag{A.2}$$

where $e_{\mathcal{K}_i} \succeq_{\mathcal{K}_i} 0$ is any positive element of the proper cone $\mathcal{K}_i$. For example, for a second-order cone $\mathcal{K}=\{(x,t) \in \mathbf{R}^q \mid \|x\|_2 \leq t\}$, we can choose $e_{\mathcal{K}}=(0,1) \in \mathbf{R}^q$; for a positive semidefinite cone $\mathcal{K}=\mathbf{S}_+^q$, we can choose $e_{\mathcal{K}}=I \in \mathbf{R}^{q \times q}$, where I is the identity matrix. Similarly, the relaxed biconvex optimization problem (3.11) is changed to

$$\begin{array}{ll} \text{minimize} & f_0(x,y) + \nu(\mathbf{1}^T s + \|t\|_1) \\ \text{subject to} & s \succeq 0 \\ & f_i(x,y) \preceq_{\mathcal{K}_i} s_i e_{\mathcal{K}_i}, \quad i=1,\ldots,m \\ & h_i(x,y)=t_i, \quad i=1,\ldots,p. \end{array} \tag{A.3}$$

We implement the `dbcp` package in a way that these generalized inequality constraints are natively supported. When a user specifies a DBCP-compliant biconvex problem, `dbcp` will automatically detect the presence of generalized inequality constraints, then determine whether the problem formulations (A.2) and (A.3) should be used instead of (3.10) and (3.11), and finally, modify the initialization procedure (algorithm 3.2) and infeasible start the ACS procedure (algorithm 3.3) accordingly. From a user's perspective, all these modifications are done in the backend and no extra input is required.

References

[AB20] A. Agrawal and S. Boyd. Disciplined quasiconvex programming. *Optimization Letters*, 14(7):1643–1657, 2020.

[ADB19] A. Agrawal, S. Diamond, and S. Boyd. Disciplined geometric programming. *Optimization Letters*, 13(5):961–976, 2019.

[AEB06] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[AH86] R. J. Aumann and S. Hart. Bi-convexity and bi-martingales. *Israel Journal of Mathematics*, 54(2):159–180, 1986.

[AKDB15] A. Ali, J. Z. Kolter, S. Diamond, and S. Boyd. Disciplined convex stochastic programming: A new framework for stochastic optimization. In *UAI*, pages 62–71, 2015.

[AKF83] F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983.

[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.

[BM07] S. Boyd and J. Mattingley. Branch and bound methods. *Notes for EE364b, Stanford University*, 2006:07, 2007.

[Bor86] J. M. Borwein. Partially monotone operators and the generic differentiability of convex-concave and biconvex mappings. *Israel Journal of Mathematics*, 54(1):42–50, 1986.

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

[Bur81] D. L. Burkholder. A geometrical characterization of Banach spaces in which martingale difference sequences are unconditional. *The Annals of Probability*, pages 997–1011, 1981.

[Bur86] D. L. Burkholder. Martingales and Fourier analysis in Banach spaces. In A. Dold and B. Eckmann, editors, *Probability and Analysis*, Lecture Notes in Mathematics, pages 61–108. Springer, 1986. Lectures given at the 1st 1985 Session of the Centro Internazionale Matematico Estivo (CIME) held at Varenna (Como), Italy May 31–June 8, 1985.

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

[CP09] W. Chu and S.-T. Park. Personalized recommendation on dynamic content using predictive bilinear models. In *Proceedings of the 18th International Conference on World Wide Web*, pages 691–700, 2009.

[CVR14] S. Chaudhuri, R. Velmurugan, and R. Rameshan. *Blind Image Deconvolution: Methods and Convergence*. Springer, 2014.

[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.

[DCB13] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076, 2013.

[DCP07] M. Dyrholm, C. Christoforou, and L. C. Parra. Bilinear discriminant component analysis. *Journal of Machine Learning Research*, 8(5), 2007.

[DL94] J. De Leeuw. Block-relaxation algorithms in statistics. In *Information Systems and Data Analysis: Prospects—Foundations—Applications*, pages 308–324. Springer, 1994.

[Flo95] C. A. Floudas. *Nonlinear and Mixed-integer Optimization: Fundamentals and Applications*. Oxford University Press, 1995.

[Flo00] C. A. Floudas. *Deterministic Global Optimization: Theory, Methods and Applications*, volume 37 of *Nonconvex Optimization and its Applications*. Springer Science & Business Media, 2000.

[FNB20] A. Fu, B. Narasimhan, and S. Boyd. CVXR: An R package for disciplined convex optimization. *Journal of Statistical Software*, 94:1–34, 2020.

[Fos18] S. M. Fosson. A biconvex analysis for Lasso ℓ_1 reweighting. *IEEE Signal Processing Letters*, 25(12):1795–1799, 2018.

[FS69] J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, 1969.

[FV90] C. A. Floudas and V. Visweswaran. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs — I. Theory. *Computers & Chemical Engineering*, 14(12):1397–1417, 1990.

[FV93] C. A. Floudas and V. Visweswaran. Primal-relaxed dual global optimization approach. *Journal of Optimization Theory and Applications*, 78(2):187–225, 1993.

[GB14] M. Grant and S. Boyd. CVX: MATLAB software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, 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 Preprint arXiv:2405.12762*, 2024.

[GH00a] Z. Geng and L. Huang. Robust stability of systems with both parametric and dynamic uncertainties. *Systems & Control Letters*, 39(2):87–96, 2000.

[GH00b] Z. Geng and L. Huang. Robust stability of the systems with mixed uncertainties under the IQC descriptions. *International Journal of Control*, 73(9):776–786, 2000.

[GPK07] J. Gorski, F. Pfeuffer, and K. Klamroth. Biconvex sets and optimization with biconvex functions: A survey and extensions. *Mathematical Methods of Operations Research*, 66(3):373–407, 2007.

[GTS+94] K. C. Goh, L. Turan, M. G. Safonov, G. P. Papavassilopoulos, and J. H. Ly. Biaffine matrix inequality properties and computational methods. In *Proceedings of 1994 American Control Conference*, volume 1, pages 850–855. IEEE, 1994.

[HKV08] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *IEEE International Conference on Data Mining*, pages 263–272. IEEE, 2008.

[JAP24] A. Jha, Z. C. Ashwood, and J. W. Pillow. Active learning for discrete latent variable models. *Neural Computation*, 36(3):437–474, 2024.

[JDMV20] H. Javanmardi, M. Dehghani, M. Mohammadi, and N. Vafamand. Bilinear matrix inequality-based nonquadratic controller design for polytopic-linear parameter varying systems. *International Journal of Robust and Nonlinear Control*, 30(17):7655–7669, 2020.

[Joh14] C. C. Johnson. Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems*, 27(78):1–9, 2014.

[JT85] M. Jouak and L. Thibault. Directional derivatives and almost everywhere differentiability of biconvex and concave-convex operators. *Mathematica Scandinavica*, pages 215–224, 1985.

[Lee93] J. M. Lee. On Burkholder's biconvex-function characterization of Hilbert spaces. *Proceedings of the American Mathematical Society*, 118(2):555–559, 1993.

[Llo82] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[Lof04] J. Lofberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the IEEE International Symposium on Computed Aided Control Systems Design*, pages 294–289, 2004.

[LS99] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[LW66] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[LWDF09] A. Levin, Y. Weiss, F. Durand, and W. T. Freeman. Understanding and evaluating blind deconvolution algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1964–1971. IEEE, 2009.

[MDC17] A. Mishra, D. K. Dey, and K. Chen. Sequential co-sparse factor regression. *Journal of Computational and Graphical Statistics*, 26(4):814–825, 2017.

[NW06] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2006.

[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.

[PB14] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[Pow73] M. J. D. Powell. On search directions for minimization algorithms. *Mathematical Programming*, 4(1):193–201, 1973.

[PT94] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Roc70] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[SBG+20] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[SDGB16] X. Shen, S. Diamond, Y. Gu, and S. Boyd. Disciplined convex-concave programming. In *IEEE 55th Conference on Decision and Control (CDC)*, 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.

[SGL94] M. G. Safonov, K. C. Goh, and J. H. Ly. Control system synthesis via bilinear matrix inequalities. In *Proceedings of 1994 American Control Conference*, volume 1, pages 45–49. IEEE, 1994.

[SLB24] P. Schiele, E. Luxenberg, and S. Boyd. Disciplined saddle programming. *Transactions on Machine Learning Research*, 2024.

[SMD22] G. So, G. Mahajan, and S. Dasgupta. Convergence of online k-means. In *International Conference on Artificial Intelligence and Statistics*, pages 8534–8569. PMLR, 2022.

[SXB14] J. V. Shi, Y. Xu, and R. G. Baraniuk. Sparse bilinear logistic regression. *arXiv Preprint arXiv:1404.4104*, 2014.

[Thi84] L. Thibault. Continuity of measurable convex and biconvex operators. *Proceedings of the American Mathematical Society*, 90(2):281–284, 1984.

[TÖ95] O. Toker and H. Özbay. On the NP-hardness of solving bilinear matrix inequalities and simultaneous stabilization with static output feedback. In *Proceedings of 1995 American Control Conference*, volume 4, pages 2525–2526. IEEE, 1995.

[UHZB16] M. Udell, C. Horn, R. Zadeh, and S. Boyd. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.

[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.

[VB00] J. G. VanAntwerp and R. D. Braatz. A tutorial on linear and bilinear matrix inequalities. *Journal of Process Control*, 10(4):363–385, 2000.

[War63] J. Warga. Minimizing certain convex functions. *Journal of the Society for Industrial and Applied Mathematics*, 11(3):588–593, 1963.

[WHJ76] R. E. Wendell and A. P. Hurter Jr. Minimization of a non-separable objective function subject to disjoint constraints. *Operations Research*, 24(4):643–657, 1976.

[WYZ12] Z. Wen, W. Yin, and Y. Zhang. Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm. *Mathematical Programming Computation*, 4(4):333–361, 2012.

[ZYHB25] H. Zhu, S. Yan, J. Hoffmann, and J. Boedecker. Multi-convex programming for discrete latent factor models prototyping. *arXiv Preprint arXiv:2504.01431*, 2025.