Technische Universität Berlin

Master Thesis

Robust conic optimization in Python

presented by
Maximilian Stahlberg

supervision
Dr. Guillaume Sagnol

examination
Prof. Dr. Martin Skutella
Prof. Dr. Manfred Opper

December 2020

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin,

Date	*Signature*

Abstract

We review finite convex reformulations of robust and distributionally robust optimization problems and document their implementation in the conic optimization Python library PICOS. The data of a decision optimization problem is rarely known with certainty. In robust optimization the true data is assumed in an uncertainty set and the decision is made with respect to the most adverse point in that set. In distributionally robust stochastic programming one considers the expectation under a worst-case distribution from a distributional ambiguity set. PICOS users may now solve scenario-uncertain conic programs, linear programs with conic uncertainty, conic quadratic programs with ellipsoidal uncertainty and least squares and piecewise linear stochastic programs where the data generating distribution is defined ambiguously through a Wasserstein ball or through bounds on its first two moments. For Wasserstein-ambiguous expected least squares, we give an application in linear signal estimation. We further discuss the implementation of biaffine expressions within the algebraic modeling language provided by PICOS.

Kurzfassung

Wir untersuchen endliche und konvexe Darstellungen von Modellen aus der Robusten Optimierung und aus der Stochastischen Programmierung mit ungewisser Wahrscheinlichkeitsverteilung und wir dokumentieren die Implementierung der Resultate in PICOS, einer Python-Bibliothek für konische Optimierungsverfahren. In der Mathematischen Optimierung sind die zugrunde liegenden Daten oft ungewiss. Die Robuste Optimierung erwartet die tatsächlichen Daten innerhalb eines gegebenen Bereichs und trifft die Optimierungsentscheidung bezüglich des schlimmstmöglichen Datums aus dieser Menge. Bei der Stochastischen Programmierung unter ungewisser Verteilung wird bezüglich einer ungünstigsten datengebenden Verteilung optimiert. Wir implementieren sieben Modelle aus beiden Bereichen. Für ein Modell, bei dem der Erwartungswert einer Quadratsumme bezüglich der ungünstigsten Verteilung aus einer Wasserstein-Kugel minimiert wird, präsentieren wir eine Anwendung aus der Signalschätzung. Zudem diskutieren wir die Erweiterung von PICOS um die Darstellung von mehrdimensionalen biaffinen Abbildungen.

In memory of my mother
Verena Stahlberg,
whose encouraging words
I have missed writing this thesis.

Contents

1. Introduction

For almost two decades, Python has made the top twenty of the most popular programming languages according to the TIOBE index, with greatly increasing popularity since 2018 and a second place peek in 2020 [46]. A detailed developers survey performed by JetBrains in 2018 shows that data analysis has become the most prominent use case of the open source scripting language, performed by more than half of the questioned developers, with machine learning a noteworthy contender that more than a third of Python users employ the language for [23]. A part of this development can certainly be attributed to a growing body of accessible mathematical libraries that provide a powerful kit of recent algorithmic and numeric tools to users beyond those who are intimately familiar with the mathematical foundation that powers these instruments. Prominent examples of such libraries are the scientific computing package SciPy [47] or the machine learning libraries scikit-learn [34] and TensorFlow [1]. This work aims to further extend the Python ecosystem in this spirit of accessibility by providing application developers painless access to powerful results from robust optimization, a field concerned with finding error-resistant numeric solutions to mathematical optimization problems.

The potential for error in applications that analyze data to compute optimal decisions is manifold. The data could be an estimate of a value that is revealed in the future, it could be obtained from an array of sensors whose precision is limited or it could be a (precise) measurement of a process that is inherently stochastic, with a known or unknown data generating distribution. Even under the assumption that the data of an optimization problem is an exact representation of the decision environment, it may not be possible to implement a precise solution to this problem without making a physical error in the process, for instance when an antenna design with ideal theoretical properties is manufactured as a physical object. The last case of implementation error is in a mathematical sense equivalent to uncertainty in the data of an optimization problem, so that we can focus our attention on the latter [5]. The cost of ignoring data uncertainties can be high as traditional solution techniques can produce optimal solutions that are very sensitive to perturbations and will perform poorly or turn out vastly infeasible if the assumption of exactness is not met by reality [5, 6]. Various fields have opened up to better understand and mitigate the pitfalls of optimization under uncertainty [5], which we outline briefly below.

Sensitivity analysis. Sensitivity analysis, in optimization, is the study of the stability properties of a given solution. It is closely related to Lagrangian duality theory and can predict the quantitative effect of (small) perturbations in the data on the solution's quality. Being a post-optimization analytical tool, sensitivity analysis does not recover a more robust solution, which raises the need for additional techniques that consider inaccuracies already during solution search.

Stochastic and chance constrained programming. When there is reason to believe that the data is of stochastic nature and its generating distribution is known or can be estimated, then it seems natural to search a solution with optimal *expected* value, and, in case of constrained optimization, one that is either expected to obey all constraints or does so with a prescribed probability. This approach is referred to as *stochastic programming* (SP) and *chance constrained programming* (CCP), respectively.

Robust optimization. While SP produces solutions that have, in general, a nonzero chance of being infeasible, *robust optimization* (RO) takes a more conservative approach. Here, the "true" data is assumed to live anywhere in a safety region surrounding the given *nominal* data and one seeks a decision that is optimal for the worst-case objective function over all possible realizations of the data in that region. A constraint in the RO model is only satisfied by a solution candidate if it holds, again, for all realizations of the data in the safety region. Unlike SP and CCP this approach gives deterministic, "hard" guarantees on the reliability of the obtained solution: If the "true" data actually falls into the safety region, then the objective value will be no worse than claimed and all constraints will be fulfilled with certainty.

Distributionally robust optimization. Apart from a lack of deterministic guarantees, a major shortcoming of SP is that it assumes complete knowledge of the data generating distribution, a requirement that is rarely met when dealing with real world data. While in practice one would estimate a distribution and claim that it was the "correct" one, this estimate is itself a new source of uncertainty which limits the credibility of the obtained solution. *Distributionally robust (stochastic) optimization* (DRO) is an approach that unites ideas from SP and RO to overcome this limitation. Here, objective function and constraint satisfaction are again measured with respect to expected values as for SP but now also the data generating distribution is *ambiguous* and only known to belong to a given safety region of distributions. As for RO, the worst-case distribution is assumed for the objective function's expected value and for each expectation constraint separately.

Multi-stage models. In some scenarios the decision maker, after making an initial decision based on uncertain data, can make an additional *recourse* decision after parts of the uncertainty are revealed. This generalizes to any number of decision stages where with each successive stage more data is certain and a different subset of decision variables may be adjusted. Multi-stage models are not a paradigm of their own but extend the frameworks discussed above in a natural way. They are most prominent in the context of stochastic linear programs where a single stage formulation makes little sense due to linearity of expectation.

In this work we focus exclusively on the single-stage RO and DRO paradigms, which have a "worst-case approach" to uncertainty management in common. This choice is not

very limiting: On one hand, it is easy to see that SP is in fact a special case of DRO that is obtained by limiting the ambiguity set of distributions to a singleton. On the other hand, CCP models can often be *safely approximated* by RO models whose safety region is chosen such that it establishes the required probabilistic guarantees. As a detailed survey of the relation between these methods is outside the scope of this document, we refer the interested reader to Ben-Tal et al. [4] and Rahimian and Mehrotra [35].

To provide application developers a seamless access to uncertainty management once they need it, we opt to extend an existing Python library, PICOS [38], that already provides extensive modeling capabilities in the classical field of constrained optimization. PICOS is the abbreviation of "Python interface to conic optimization solvers" and provides a solver-agnostic, high level modeling language for optimization problems that can be put into conic form and solved efficiently by a variety of open source and proprietary solvers. The open source library follows a "what you see is what you get" principle of allowing users to input mathematical models much like they would write them on paper, to the extend of what is possible within the Python programming language. This is in contrast to most solver software that accepts optimization models only in a very specific, low level form. Our extension stays true to this principle and comprises significant enhancement of the symbolic computation backend powering PICOS to allow robust and distributionally robust models to be input in a variety of natural ways, as showcased in section 5 and detailed in section 6.

1.1. Related work

Despite a vibrant landscape of theory and techniques, surprisingly little software exists to aid with the implementation of (distributionally) robust optimization models that have a tractable representation [21]. Most notably, the modeling extension presented here appears to be unprecedented in the Python ecosystem where existing libraries concerned with optimization under uncertainty all focus on stochastic programming:

- PySP [48] is an extension to the Python-based optimization modeling language Pyomo [22] that is capable of solving large-scale, multi-stage stochastic mixed-integer programs. As models in this general setting are particularly hard to solve, the focus of PySP is performance and scalability through the use of heuristics.
- MSPPy [16] is a young, more narrowly scoped Python library implementing recent results in multi-stage stochastic linear and integer programming.
- APLEpy [25] is a discontinued, early version of a Python-embedded modeling language with support for multi-stage stochastic programs.

The situation with respect to RO and DRO is not as forlorn in other programming environments. There are two options for the commercial MATLAB environment:

- YALMIP [30] is a MATLAB-embedded optimization modeling language with a scope similar to PICOS and with explicit support for robust optimization [29]. It can solve

linear programs whose data depends affinely on conically bounded uncertainty as well as conic programs with scenario uncertainty (see section 3 for a definition). Additionally, some uncertain formulations involving polynomials are supported.

- ROME [21] is a MATLAB package dedicated to multi-stage robust and distributionally robust optimization. In the latter case, an unknown distribution can be specified in terms of moment constraints or directional derivatives. While the support for robust methods is comparatively rich, ROME's deterministic optimization capabilities do not extend beyond conic quadratic programming.

In addition to the time-honored MATLAB and the rising star of Python, the young Julia programming language is an upcoming contender in the scientific computing landscape that already provides rich support for mathematical optimization applications. In terms of robust methods,

- JuMPeR [18], an extension to the Julia optimization package JuMP [17], implements both single and multi-stage approaches from the toolbox of classical robust optimization. It ships with built-in support for polyhedral and ellipsoidal uncertainty sets but allows more advanced sets to be implemented by the user that are then handled with a cutting plane approach.

Lastly, there are several standalone algebraic modeling languages (AMLs) that do not come attached to a general purpose programming environment. Out of these frameworks,

- AIMMS [36], a commercial AML, stands out for having built-in support for a wide range of methods to solve uncertain linear and mixed integer problems, including multi-stage robust optimization with polyhedral and ellipsoidal uncertainty sets.

Since this work is of applied nature, we do not list theoretical results here. A short introduction mentioning mathematical milestones in robust and distributionally robust optimization is found in sections 3 and 4, respectively.

1.2. Summary of results

Applied contributions. We review and extend a number of results from the fields of robust and distributionally robust optimization and make them available in the Python ecosystem through the PICOS library. For robust optimization, we discuss and implement the following models:

Problem type	Uncertainty set	Theory	Example
General conic	Scenario-based	Section 3.1	Section 5.2.1
Linear	Conically bounded	Section 3.2	Section 5.2.2
Conic quadratic	Ellipsoidal	Section 3.3	

For distributional robust optimization, we support the worst-case expectation of the following loss functions under the given distributional ambiguity sets:

Loss function	Ambiguity set	Theory	Example
Least squares	Bounded moments	Section 4.1.1	—
	Type 2 Wasserstein ball	Section 4.2.1	Section 5.2.4
Piecewise linear	Bounded moments	Section 4.1.2	Section 5.2.3
	Type 1 Wasserstein ball	Section 4.2.2	—

To allow for a convenient input of uncertain data, we further extend the PICOS backend with a powerful representation of biaffine expressions that could be leveraged in the future for other applications involving parameterized data.

Theoretical contributions. In the following we write *uncertain affine expression* to denote a (multidimensional) expression $a(x, \xi)$ that is biaffine in a decision vector x and in a perturbation parameter ξ that represents the uncertainty.

For robust conic quadratic programming, we generalize the uncertainty set discussed by Ben-Tal et al. [4] from an Euclidean unit ball to any solid, arbitrarily shifted ellipsoid.

For the moment ambiguity set introduced by Delage and Ye [13] that bounds the first two moments and the support of the unknown distribution, we extend a result that originally applies to convex or concave piecewise linear functions of the inner product of the decision vector with the perturbation parameter to apply more generally to the pointwise minimum or maximum of finitely many scalar uncertain affine expressions. Similarly, we extend a follow-up result by Mehrotra and Zhang [31] that applies to a least squares loss function where the uncertainty is linearly separable in the inputs and outputs of a regression model to apply more generally to the squared Euclidean norm of any uncertain affine expressions. Additionally, for both least squares and piecewise linear loss, we provide new variants of the included results where the mean, the covariance matrix, the support, and both the mean and the covariances are unbounded.

For Wasserstein-ambiguous expected least squares, we apply a recent result by Kuhn et al. [28] to obtain a finite semidefinite reformulation of a constraint that poses an upper bound on the worst-case expectation of a squared Euclidean norm of an uncertain affine expression. We further give a novel hypothetical application in linear signal estimation where Wasserstein-ambiguous stochastic programming has a reduced out-of-sample error compared to a L_2-regularized minimum mean squared error approach.

Lastly, we derive a collection of identities that allow biaffine matrix expressions to be implemented in an algebraic modeling framework where coefficient matrices are stored with respect to vectorized expressions. The resulting formulations make use of sparse Kronecker products involving almost only identity and commutation matrices so that they can be implemented efficiently using available numerical computation libraries.

Some of these identities are derived using the remarkable blockwise vectorization operator "vecb" discussed in Koning et al. [26] and elsewhere.

1.3. Outline

The remainder of this document is structured as follows. Section 2 introduces mathematical notation and basic definitions as well as instructions to understand and reproduce the Python listings throughout this document. Assuming basic knowledge of linear algebra, conic optimization and the Python programming language, it can be skipped on a first read and consulted whenever notation is not clear. Sections 3 and 4 introduce basic concepts of RO and DRO and provide a short review of robust programs that admit a tractable representation. The focus is on models that can be solved using PICOS as a result of this work. Section 5.1 introduces the reader to the PICOS library. After reading this section, one should be able to define and solve constrained optimization problems in Python. Section 5.2 then showcases the interface to the new features of PICOS and includes the mentioned application in linear signal estimation. Lastly, section 6 describes the implementation work that constitutes the practical part of this thesis. The section is written in a mathematical tone and is accompanied by appendix A that depicts the formal scope of the practical work in terms of added and modified source code.

2. Preliminaries

This section introduces notation and definitions used throughout the document and gives instructions on how to read and reproduce the Python code listings that showcase implementation features. While greatly condensed, this section can by no means provide an adequate introduction to the field of conic optimization that builds the foundation for this work. We refer the reader to Boyd and Vandenberghe [11] for that purpose.

2.1. Discrete structures

Notation 1 (Integers)**.** We write $\mathbb{Z}_{\geqslant 1}$ for the positive and $\mathbb{Z}_{\geqslant 0}$ for the nonnegative integers.

Definition 1 (Integer range)**.** It is $[n] := \{i \in \mathbb{Z}_{\geqslant 1} \mid i \leqslant n\}$ and $[(m, n)] := [m] \times [n]$.

Definition 2 (Floored division)**.** For $m, n \in \mathbb{Z}$, $m \div n := \lfloor \frac{m}{n} \rfloor$ denotes floored division and $m \mathbin{\%} n := m - (m \div n)n$ denotes the modulo operation.

Definition 3 (Mapping)**.** A set M whose elements are of the form $a \mapsto b$ is called a *mapping* or *map* for short. We refer to a and b as a key and a value of M, respectively, and we implicitly require uniqueness of keys, formally $|\{a \mid \exists b : a \mapsto b \in M\}| = |M|$. We use mappings to represent objects of the built-in ***`dict`*** type of the Python language.

2.2. Linear algebra

Notation 2 (Fields and vector spaces)**.** We write $\mathbb{R}$ for the reals, $\mathbb{C}$ for the complex numbers and $\mathbb{K}$ as a placeholder for either of these fields. We further write $\mathbb{S}^n \subseteq \mathbb{R}^{n\times n}$ for the space of $n \times n$ symmetric matrices.

Notation 3 (Closures and interiors)**.** For a set $A \subseteq \mathbb{K}^n$, we write $\operatorname{cl}(A)$ for the topological closure and $\operatorname{Conv}(A)$ for the convex hull of A. We further write $\operatorname{int}(A)$ for the topological interior and $\operatorname{relint}(A)$ for the interior relative to the affine closure of A.

Notation 4 (Matrix indexing)**.** For a matrix $A \in \mathbb{K}^{m\times n}$ and $(i,j) \in [(m,n)]$ we write

- $A_{i,j} \in \mathbb{K}$ for the element in the i-th row and j-th column,
- $A_{i,:} \in \mathbb{K}^{1\times n}$ for the i-th row as a row vector,
- $A_{:,j} \in \mathbb{K}^{m\times 1}$ for the j-th column as a column vector,
- $A_i \in \mathbb{K}$ for the element in the i-th row if $n = 1$ and
- $A_j \in \mathbb{K}$ for the element in the j-th column if $m = 1$.

On the other hand, the notation $A_{(\cdot)}$ always refers to a matrix in a family of matrices.

Notation 5 (Vector orientation)**.** Wherever a distinction between a row and a column vector is relevant, a vector $a \in \mathbb{K}^n$ is understood as the column vector $a \in \mathbb{K}^{n\times 1}$.

As in notation 4, a_i with $i \in [n]$ denotes the i-th element of a while $a_{(\cdot)}$ refers to a vector in a family of vectors.

Notation 6 (Transposition)**.** For a matrix A we write A^H short for the conjugate transpose $\overline{A}^T$ and, if A is invertible, A^{-T} short for the transposed inverse $(A^{-1})^T$.

Definition 4 (Complex inner product)**.** In accordance with the choice made in PICOS, we define the complex inner product between matrices $A, B \in \mathbb{C}^{m\times n}$ as

$$\langle A,\ B\rangle := \operatorname{tr}\left(AB^H\right) = \overline{\operatorname{tr}\left(A^H B\right)}.$$

Notation 7 (Special products)**.** We write $A \otimes B$ and $A \odot B$ to denote the Kronecker product and the elementwise Hadamard product between matrices A and B, respectively.

Definition 5 (Reshaping and vectorization)**.** For a matrix $A \in \mathbb{K}^{m\times n}$ and $p, q \in \mathbb{Z}_{\geqslant 1}$ with $p * q = m * n$, $\operatorname{reshape}_{p,q}(A) \in \mathbb{K}^{p\times q}$ with

$$\left(\operatorname{reshape}_{p,q}(A)\right)_{i,j} := A_{(jm+i) \,\%\, p,\,(jm+i)\div p} \quad \forall\ (i,j) \in [(p,q)]$$

denotes column-major reshaping and

$$\operatorname{vec}(A) := \operatorname{reshape}_{mn,1}(A)$$

denotes column-major column-vectorization.

Definition 6 (Special matrices)**.** We write

- $I_n \in \mathbb{Z}^{n\times n}$ with $(I_n)_{i,j} := 0^{|i-j|} \ \forall \ i,j \in [n]$ for the identity matrix,
- $E_{(m,n/k,l)} \in \mathbb{Z}^{m\times n}$ with $(E_{(m,n/k,l)})_{i,j} = 0^{|k-i|+|l-j|} \ \forall \ (i,j) \in [(m,n)]$ for the $m \times n$ matrix that has a one at (k,l) and is zero elsewhere and
- $K_{(m,n)} \in \mathbb{Z}^{mn\times mn}$ for the unique commutation matrix defined through

$$K_{(m,n)} \operatorname{vec}(A) = \operatorname{vec}\left(A^T\right) \ \forall \ A \in \mathbb{K}^{m\times n}.$$

Definition 7 (Blockwise vectorization [26])**.** For a blockwise partitioned matrix

$$A = \sum_{(i,j)\in[(p,q)]} E_{(m,n/i,j)} \otimes A_{(i,j)} \in \mathbb{K}^{pm\times qn}$$

with $A_{(i,j)} \in \mathbb{K}^{m\times n} \ \forall \ (i,j) \in [(p,q)]$ and $m,n,p,q \in \mathbb{Z}_{\geqslant 1}$ we write

$$\begin{aligned}\operatorname{vecb}_{(p,q),(m,n)}(A) &:= (I_q \otimes K_{(p,n)} \otimes I_m) \operatorname{vec}(A) \\ &= \begin{pmatrix} \operatorname{vec}\left(A_{(1,1)}\right) \\ \vdots \\ \operatorname{vec}\left(A_{(p,1)}\right) \\ \vdots \\ \operatorname{vec}\left(A_{(1,q)}\right) \\ \vdots \\ \operatorname{vec}\left(A_{(p,q)}\right) \end{pmatrix} \in \mathbb{K}^{pqmn}\end{aligned}$$

for the vertically stacked column-major column-vectorizations of each $(m \times n)$-sized block in block-column-major order.

Definition 8 (Biaffine expression)**.** Let $k \in \mathbb{Z}_{\geqslant 1}$ and let $x_{(i)} \in \mathbb{R}^{\alpha_i}$ with $\alpha_i \in \mathbb{Z}_{\geqslant 1}$ be real variable vectors for all $i \in [k]$. A $m \times n$ biaffine expression over a field $\mathbb{K}$ concerning these variables has the form

$$A := \sum_{1\leqslant i<j\leqslant k} A_{(i,j)}(x_{(i)}, x_{(j)}) + \sum_{i\in[k]} A_{(i)}(x_{(i)}) + A_{()} \in \mathbb{K}^{m\times n}$$

where $A_{()} \in \mathbb{K}^{m\times n}$ is constant and, for all $i,j \in [k]$ with $i < j$, $A_{(i)}(x_{(i)}) \in \mathbb{K}^{m\times n}$ is linear in $x_{(i)}$ and $A_{(i,j)}(x_{(i)}, x_{(j)}) \in \mathbb{K}^{m\times n}$ is bilinear in $x_{(i)}$ and $x_{(j)}$.

We say that A is *properly biaffine* if $A_{(i,j)} \neq 0$ for some $i,j \in [k]$ with $i < j$. If A is not properly biaffine, then we call it *affine*. Further, we say that A is *properly affine* if it is affine and $A_{(i)} \neq 0$ for some $i \in [k]$. It follows that if A is affine but not properly affine, then $A = A_{()}$ is constant.

2.3. Conic optimization

Definition 9 (Proper cone)**.** Let $K \subseteq \mathbb{R}^n$ be a convex set. We call K a *(convex) cone* if it is closed under a linear combination with nonnegative coefficients. We further call K a *proper cone* if, additionally, it is closed, $\operatorname{cl}(K) = K$, pointed, $K \cap (\mathbb{R}^n \setminus K) = \{0\}$, and solid, $\operatorname{int}(K) \neq \emptyset$.

Definition 10 (Special cones)**.** For each $n \in \mathbb{Z}_{\geqslant 1}$, the following are proper cones:

- The nonnegative orthant $\mathbb{R}^n_+ := \{x \in \mathbb{R}^n \mid x_i \geqslant 0 \ \forall \ i \in [n]\}$.
- The second order cone $Q_n := \left\{x \in \mathbb{R}^n \mid \sqrt{\sum_{i=2}^n x_i^2} \leqslant x_1\right\}$.
- The cone of positive semidefinite matrices $\mathbb{S}^n_+ := \left\{X \in \mathbb{S}^n \mid z^T X z \geqslant 0 \ \forall \ z \in \mathbb{R}^n\right\}$.

We call the special case of $\mathbb{R}_+ := \mathbb{R}^1_+$ the nonnegative ray.

Definition 11 (Dual cone)**.** Let $X \subseteq \mathbb{R}^n$. The closed convex cone

$$X^* := \left\{y \in \mathbb{R}^n \mid \forall x \in X : y^T x \geqslant 0\right\}$$

is called the *dual cone* of X. Note that $(X^*)^* = X$ if already X is a closed convex cone.

Definition 12 (Product cone)**.** If K_i are (proper) cones for all $i \in [n]$, then also the Cartesian product $K := \prod_{i=1}^n K_i$ is a (proper) cone and we refer to K as a *product cone*.

Definition 13 (Conic inequality)**.** Given a proper cone $K \subseteq \mathbb{R}^n$, we define the partial order

$$\preceq_K := \left\{(a, b) \in (\mathbb{R}^n)^2 \mid b - a \in K\right\}$$

and we call the term $a \preceq_K b$ for some $a, b \in \mathbb{R}^n$ a *conic inequality*.

For $n \in \mathbb{Z}_{\geqslant 2}$ we write $\preceq$ short for both the elementwise inequality $\preceq_{\mathbb{R}^n_+}$ and the Loewner order $\preceq_{\mathbb{S}^n_+}$. In the latter case the relation is defined on symmetric matrices and we refer to it as a *linear matrix inequality*.

Definition 14 (Generalized convex function)**.** A function $f : \mathbb{R}^n \to R^m$ is K-*convex* for a proper cone $K \subseteq \mathbb{R}^n$ if, for all $x, y \in \mathbb{R}^n$ and $0 \leqslant \theta \leqslant 1$, it is

$$f(\theta x + (1-\theta)y) \preceq_K \theta f(x) + (1-\theta) f(y).$$

It follows that f is *convex* if it is $\mathbb{R}_+$-convex.

Definition 15 (Equivalent problem)**.** We call two optimization problems *equivalent* if from a solution to one, a solution to the other can be obtained easily. We further say that two problems are *trivially equivalent* if the definition of one can be transformed into the definition of the other by means of simple equivalent transformations. We refer to Boyd and Vandenberghe [11] for a discussion of such transformations.

Definition 16 (Conic problem)**.** A conic optimization problem has the standard form

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & f(x) \\ \text{subject to} & g(x) \preceq_K 0 \end{array}$$

where f and g are affine in x and K is a proper cone. We also call an optimization problem a conic problem if it is trivially equivalent to a problem in the above form.

Definition 17 (Essential strict feasibility)**.** The conic problem in definition 16 is *strictly feasible* if it exists $x \in \mathbb{R}^n$ with $g(x) \prec_k 0$. If a conic problem is trivially equivalent to the conic problem in definition 16 with an additional equivalence constraint $g'(x) = 0$ with g' affine in x, then it cannot be strictly feasible but is called *essentially strictly feasible* if there is $x \in \mathbb{R}^n$ with $g(x) \prec_k 0$ and $g'(x) = 0$.

Definition 18 (Problem types)**.** We call an optimization problem a linear program (LP) or a semidefinite program (SDP) if it is trivially equivalent to a conic problem in standard form with $K = \mathbb{R}^m_+$ or $K = \mathbb{S}^m_+$, respectively, for some $m \in \mathbb{Z}_{\geqslant 1}$. We further call a problem a second order conic problem (SOCP) if it is trivially equivalent to a conic problem where K is a Cartesian product of the nonnegative orthant and any number of second order cones. LPs are a subclass of SOCPs and SOCPs are one of SDPs [43].

2.4. Python

Notation 8 (Code listings)**.** Python code is typeset in `monospace`. There are two types of code listings:

- *Interactive listings* have all statements prefixed with "`>>> `" ("`... `" when continuing long statements) and can be reproduced by copying the statements without the prefix into an interactive Python interpreter.[1]
- *Raw listings* do not have such a prefix and cannot be reproduced directly.

Convention 1 (Imports)**.** The following imports are assumed throughout the document:

```
>>> import cvxopt, numpy, picos
>>> import scipy.signal, scipy.spatial
>>> import textwrap
>>> from matplotlib import pyplot
```

Convention 2 (Reproducibility)**.** To allow the reader to reproduce our interactive listings verbatim, we seed NumPy's random number generator with a fixed value of 1 **at the start of every section**:

```
>>> numpy.random.seed(1)
```

[1] An interactive Python interpreter can be launched from a terminal with the command `python`.

Furthermore, we let PICOS use only the free CVXOPT [2] solver for solution search, which is a dependency of PICOS and thus available to every user:

```
>>> picos.settings.SOLVER_WHITELIST = ["cvxopt"]
```

Convention 3 (Formatting)**.** PICOS returns multidimensional numeric results in the form of CVXOPT matrices. We want to print those more compactly:

```
>>> cvxopt.matrix_str = \
...     lambda X: cvxopt.printing.matrix_str_default(X).strip()
```

By default, PICOS produces unicode output to typeset math. If your terminal does not support unicode symbols, you can disable them as follows:

```
>>> picos.ascii()
```

Most figures in this document are produced directly from the code listings. Their appearance is adjusted through additional commands that are not shown in the text.

Notation 9 (Omissions)**.** In raw code listings, text in parenthesis that is typeset in proportional font marks omitted code:

```
for i in range(10):
    (Do something.)
```

Notation 10 (Command sequence)**.** In interactive listings we often chain an assignment and its target variable separated by a semicolon:

```
>>> a = "some statement"; a
'some statement'
```

This is valid Python code that first assigns the value of a statement to a variable and then produces terminal output as if the statement was executed without the assignment.

Notation 11 (Throwaway assignment)**.** Following a Python convention, we use assignments of the form

```
>>> _ = "some statement"
```

when a statement returns a value that we do not want to either show or store.

Convention 4 (Software versions)**.** Interactive listings were produced and validated using Python 3.9.1, PICOS 2.1.1, CVXOPT 1.2.5, NumPy 1.19.4 and SciPy 1.5.4. In particular, results of this work are released as part of the PICOS 2.1 milestone release, of which 2.1.1 is the first stable version with a fix to the installation routine of 2.1.0.

3. Robust optimization

Robust optimization methods date back to a 1973 note by Soyster [45] where a solution strategy to an "inexact linear program" is derived as an application of a novel approach to specify a problem's feasible region through a set inclusion constraint. Soyster assumes therein that the "exact" value of each column of the LP's constraint matrix lies in a hypersphere centered at a per-column nominal value. Using his framework, he defines an auxiliary problem whose feasible region is defined through a set inclusion constraint and requires a solution to be feasible, in the original LP, for every possible realization of the uncertainty in each column. Soyster then obtains another LP that is equivalent to the auxiliary problem and that depends explicitly on the nominal data and the spheres' radii. The author compares his result to the established concept of stochastic programming and concludes that his solution strategy is an "ultraconservative" approach to deal with uncertainty. Indeed, the field of robust optimization that he pioneered and that would later stand in competition with stochastic approaches is often considered too conservative by proponents of the latter discipline, which refers to the opinion that the price one pays to hedge against the absolute worst case scenario, as unlikely as it may be, is not justified in many practical applications. Despite this preconception and after a slow start, robust optimization has gained a surge of interest in the 1990s, when more general frameworks were discovered that allow a wider variety of data uncertainties and problem types to be considered. The defining element of these approaches is a *robust counterpart scheme*, where the uncertain problem is replaced by a certain approximation that allows only decisions that are safe with respect to any realization of the uncertainty in the original problem. The term makes its first appearance in Ben-Tal and Nemirovski [5] who look to "lay the foundation of robust convex optimization". Therein the authors attribute the first "general" robust counterpart scheme to their earlier work on truss topology design [8] and, independently, to a preliminary version of an article by El Ghaoui and Lebret [19]. While not limited to a single application, both of these papers focus on the case where the uncertain data is bounded by an ellipsoid, which yields tractability results for both LPs and SOCPs. Further research has pushed the boundary, quite literally, of how data uncertainty sets may be defined. Most notably, Ben-Tal et al. [4] and Ben-Tal and Nemirovski [7, 9] show that in the case of an LP, bounding the data by conic inequalities results in a robust counterpart that can be represented by finitely many conic inequalities with respect to the dual cones, which offers a great deal of flexibility when defining uncertain LPs.

The remainder of this section is a review of robust counterpart schemes that we have implemented in PICOS and of a few more important results. Formally, we consider throughout this section uncertain conic optimization problems of the form

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x, \theta) && \text{(3.1a)} \\ \text{subject to} \quad & g(x, \theta) \preceq_K 0 && \text{(3.1b)} \end{aligned}$$

where K is a proper (product) cone, f and g are both biaffine functions in x and θ, and where $\theta \in \Theta$ is an uncertain *perturbation (parameter)* living in a nonempty *perturbation set* $\Theta \subseteq \mathbb{R}^m$. Within the robust optimization paradigm we are interested in a solution that is separately feasible and optimal for a worst case realization of θ. To this end we want to solve the *robust counterpart* of problem (3.1), defined as

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & \sup_{\theta \in \Theta} f(x, \theta) && \text{(3.2a)} \\ \text{subject to} \quad & g(x, \theta) \preceq_K 0 \quad \forall \theta \in \Theta. && \text{(3.2b)} \end{aligned}$$

We refer to objective (3.2a) and constraint (3.2b), respectively, as the *robust* versions of the *uncertain* (3.1a) and (3.1b). Note that, even though it constitutes a whole family of inequalities, we call (3.2b) just *a* robust constraint.

Without loss of generality, we restrict the uncertain objective (3.1a) to be *certain*, formally $f(x, \theta) = c^T x + d$ for some $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$, which allows us to omit the supremum in the robust objective (3.2a). This is not a limitation as problem (3.1) admits an equivalent formulation with a linear objective and, otherwise, the same general structure:

$$\begin{aligned} \underset{x \in \mathbb{R}^n,\, t \in \mathbb{R}}{\text{minimize}} \quad & t && \text{(3.3a)} \\ \text{subject to} \quad & \begin{pmatrix} f(x, \theta) \\ g(x, \theta) \end{pmatrix} \preceq_{(\mathbb{R}_+ \times K)} \begin{pmatrix} t \\ 0 \end{pmatrix} && \text{(3.3b)} \end{aligned}$$

Proof. Consider an epigraph reformulation of problem (3.1) where a new scalar variable t replaces the objective (3.1a) by virtue of the additional constraint $f(x, \theta) \leqslant t$. Since the scalar inequality $\leqslant$ can be written as the conic inequality $\preceq_{\mathbb{R}_+}$, we obtain the new constraint (3.3b) by forming the product cone $\mathbb{R}_+ \times K$. □

Unless Θ is finite, the robust counterpart (3.2) is semi-infinite with a finite number of variables but an infinite number of constraints, parameterized over Θ. To solve it numerically with off-the-shelf optimization solvers, we are looking to find an equivalent, finite representation that does not depend on θ and is by some authors referred to as a *deterministic representation* for that reason (e.g. in Averbakh and Zhao [3]). In the following we give a short review of cones K and perturbation sets Θ that allow for such a representation.

To narrow down the cones of interest, note that if $K = \prod_{i=1}^{q} K_i$ is a product cone, then we can replace the robust constraint (3.2b) with an equivalent set of constraints

$$C := \{ g_i(x, \theta) \preceq_{K_i} 0 \;\; \forall \;\; \theta \in \Theta \mid i \in [q] \}$$

where

$$g_i(x, \theta) := \begin{bmatrix} (g(x, \theta))_{1 + \sum_{j<i} \dim(K_j)} & \cdots & (g(x, \theta))_{\sum_{j \leqslant i} \dim(K_j)} \end{bmatrix}^T$$

is again biaffine in x and θ for all $i \in [q]$.

Proof. Let x be fixed. For a forward implication, assume towards a contradiction that a constraint in C is violated but constraint (3.2b) holds. Let $(g_i(x,\theta) \preceq_{K_i} 0 \ \forall \ \theta \in \Theta)$ for some $i \in [q]$ be the violated constraint and choose θ such that $g_i(x,\theta) \not\preceq_{K_i} 0$. Clearly, also $g(x,\theta) \not\preceq_K 0$ and thereby constraint (3.2b) is violated. For a backward implication, assume towards a contradiction that constraint (3.2b) is violated but all constraints in C hold. Choose θ such that $g(x,\theta) \not\preceq_K 0$. By construction of K, there must be some $i \in [q]$ with $g_i(x,\theta) \not\preceq_{K_i} 0$ so that $(g_i(x,\theta) \preceq_{K_i} 0 \ \forall \ \theta \in \Theta) \in C$ is violated. □

In other words, if constraint (3.1b) is a shorthand notation for multiple uncertain conic constraints (whose cones are simpler in structure than K), then the robust counterpart acts independently on each of these constraints. We will thus focus on a small number of basic cones that are supported by general purpose optimization solvers and are used by PICOS to represent all kinds of convex constraints. In particular, we will consider the nonnegative ray (which the nonnegative orthant is a product cone of), the second order cone and the cone of positive semidefinite matrices. For a start, however, we will look at the special case of *scenario uncertainty* which yields a tractable robust counterpart already when K is a product of the above cones (or any other "tractable" cone).

3.1. General conic programming

As we shall observe throughout our review of robust optimization techniques, when the goal is to obtain a tractable robust counterpart, then one needs to accept a tradeoff between the expressive power of the uncertain program's structure and the flexibility in defining the uncertainty set (see also Ben-Tal and Nemirovski [8]). A particularly simple uncertainty set is that of scenario uncertainty, where Θ is a convex polytope defined as the convex hull of a (small) number of vertices, the *scenarios*. Intuitively, the modeler may have observed a number of realizations of the uncertainty and would like to conclude that the most "extreme" cases are already among them. Future realizations of the uncertainty are thus believed to be representable as a convex combination of the scenarios seen so far. Of course, such an approach can only be reasonable if the observed perturbations are low-dimensional as otherwise the number of scenarios necessary to define a meaningful safety region is typically exponential in the number of dimensions [4, 5]. Despite this drawback, the concept of scenario uncertainty can be attractive when the goal is to quickly increase a solution's stability properties in an environment where no thorough analysis of the uncertainty is available and where meaningful robustness guarantees are secondary. The reward for such modest modeling requirements concerning the uncertainty is great flexibility with respect to the problem's structure, which lets us formulate the following theorem with no additional restrictions on the cone K:

Theorem 3.1. *For* $\Theta = \operatorname{Conv}\left(\left\{a_{(i)} \in \mathbb{R}^m \mid i \in [k]\right\}\right)$, $k \in \mathbb{Z}_{\geqslant 1}$, *the robust constraint* (3.2b) *is equivalent to the finite set of constraints* $\{g(x,a_i) \preceq_K 0 \mid i \in [k]\}$.

Proof. Let x be fixed. For the forward implication, let constraint (3.2b) hold, that is $g(x, \theta) \preceq_K 0$ for all $\theta \in \Theta$. Since by definition $a_{(i)} \in \Theta$ for all $i \in [k]$, it follows immediately that also $g(x, a_{(i)}) \preceq_K 0$ for all $i \in [k]$. For the backward implication, let $g(x, a_{(i)}) \preceq_K 0$ hold for all $i \in [k]$ and let $\theta \in \Theta$ be fixed. By definition of Θ, we have $\theta = \sum_{i=1}^{k} \gamma_i a_{(i)}$ for a vector $\gamma \in \mathbb{R}^k_+$ with $\sum_{i=1}^{k} \gamma_i = 1$. Since, for all $i \in [k]$, $g(x, a_{(i)}) \preceq_K 0$ is short for $-g(x, a_{(i)}) \in K$ and $-g(x, a_{(i)})$ is affine in $a_{(i)}$ while K is convex, it follows that also $-g(x, \sum_{i=1}^{k} \gamma_i a_{(i)}) \in K$ and, equivalently, $g(x, \theta) \preceq_K 0$. □

In PICOS, scenario uncertainty is supported in the objective with uncertain affine expressions, uncertain convex or concave piecewise linear functions and with uncertain (squared) Euclidean or Frobenius norms. Additionally, all constraint types that have an immediate conic representation, including linear matrix inequalities, admit a scenario-robust version. At this point this is the only option to define uncertain SDPs.

3.2. Linear programming

If we choose K as the nonnegative orthant, then problem (3.1) becomes a linear program, arguably the best understood and most widely applied special case of conic programs. As we have argued before, we can reduce to the case where K is the nonnegative ray and consider only scalar constraints of the form

$$g(x, \theta) := \theta^T A x + a_x^T x + a_\theta^T \theta + \alpha \leqslant 0 \tag{3.4}$$

with $A \in \mathbb{R}^{m \times n}$, $a_x \in \mathbb{R}^n$, $a_\theta \in \mathbb{R}^m$ and $\alpha \in \mathbb{R}$. We call $a_x^T x + \alpha$ the certain and $\theta^T A x + a_\theta^T \theta$ the uncertain part of g. If we fix the parameter θ, then $\theta^T A x + a_x^T x$ constitutes the linear and $a_\theta^T \theta + \alpha$ the constant part of g.

While a scalar affine constraint is already a powerful building block for mathematical models, it remains the least expressive that we are going to investigate and as such it admits the greatest flexibility when it comes to modeling the perturbation set Θ. We will see that, under mild technical conditions, we can efficiently solve the robust version of the uncertain constraint (3.4) if Θ is given as the feasible region of a conic problem whose dual problem we can solve. To this end we define

$$\Theta := \{\theta \in \mathbb{R}^m \mid \exists \nu \in \mathbb{R}^w : P\theta + Q\nu + p \in L\} \tag{3.5}$$

where $L \subseteq \mathbb{R}^l$ is a proper cone with $P\theta + Q\nu + p \in \operatorname{relint}(L)$ for some $\theta \in \mathbb{R}^m$ and $\nu \in \mathbb{R}^w$ and where $P \in \mathbb{R}^{l \times m}$, $Q \in \mathbb{R}^{l \times w}$ and $p \in \mathbb{R}^l$ are of the necessary sizes. The following theorem states that this careful choice yields a finite representation for the robust problem (3.2):

Theorem 3.2 (Ben-Tal et al. [4])**.** *The robust constraint* (3.2b) *with* g *and* Θ *defined as in* (3.4) *and* (3.5) *is equivalent to the following system of constraints concerning*

variables x *and* $y \in \mathbb{R}^l$*:*

$$\begin{aligned} p^T y + a_x^T x + \alpha &\leqslant 0, & Q^T y &= 0, \\ P^T y + Ax + a_\theta &= 0, & y &\in L^*. \end{aligned}$$

Proof. Let x be fixed. We have that

$$\begin{aligned} & & &\text{constraint (3.2b) holds} \\ &\iff & &\forall \theta \in \Theta : \theta^T Ax + a_x^T x + a_\theta^T \theta + \alpha \leqslant 0 \\ &\iff & &\sup_{\theta \in \Theta} \left(\theta^T Ax + a_x^T x + a_\theta^T \theta + \alpha \right) \leqslant 0 \\ &\iff & &\omega \leqslant 0 \end{aligned}$$

where ω is the optimal value of the conic problem

$$\begin{aligned} \underset{\theta \in \mathbb{R}^m}{\text{maximize}} \quad & \theta^T Ax + a_x^T x + a_\theta^T \theta + \alpha \\ \text{subject to} \quad & \theta \in \Theta, \end{aligned} \tag{3.6}$$

if that problem is bounded, and $\omega = \infty$, otherwise. By definition of Θ, we can write problem (3.6) as

$$\begin{aligned} \underset{\theta \in \mathbb{R}^m,\, \nu \in \mathbb{R}^w}{\text{maximize}} \quad & \theta^T Ax + a_x^T x + a_\theta^T \theta + \alpha \\ \text{subject to} \quad & -(P\theta + Q\nu) \preceq_L p. \end{aligned} \tag{3.7}$$

Recall the assumption that $P\theta + Q\nu + p \in \operatorname{relint}(L)$ for some $\theta \in \mathbb{R}^m$ and $\nu \in \mathbb{R}^w$. This means that problem (3.7) is essentially strictly feasible. Since further L is a proper cone and both the objective function and $P\theta + Q\nu + p$ are affine (thus convex and L-convex, respectively), strong duality holds by Slater's condition [11]. The Lagrange dual problem of problem (3.7) is

$$\begin{aligned} \underset{y \in \mathbb{R}^l}{\text{minimize}} \quad & p^T y + a_x^T x + \alpha \\ \text{subject to} \quad & -P^T y = Ax + a_\theta, \\ & -Q^T y = 0, \\ & y \succeq_{L^*} 0 \end{aligned} \tag{3.8}$$

and its optimal value is equal to ω unless $\omega = \infty$. For brevity we define the dual feasibility predicate

$$\Omega(y) := \left(-P^T y = Ax + a_\theta \wedge -Q^T y = 0 \wedge y \succeq_{L^*} 0 \right).$$

With the convention of $\inf \emptyset = \infty$ we have

$$\begin{aligned} & & &\omega \leqslant 0 \\ &\iff & &\inf \left\{ p^T y + a_x^T x + \alpha \mid \exists y \in \mathbb{R}^l : \Omega(y) \right\} \leqslant 0 \\ &\iff & &\exists y \in \mathbb{R}^l : \left(p^T y + a_x^T x + \alpha \leqslant 0 \wedge \Omega(y) \right) \end{aligned}$$

and the theorem follows from a straightforward rearrangement of $\Omega(y)$. □

The above representation of Θ, where L may be a product cone and where ν plays the role of an auxiliary variable, is particularly well suited to be defined within a conic optimization framework like PICOS. As we will see in section 5, a PICOS user can input Θ by providing a number of bound constraints that have a conic representation, much like they would define the feasible region of a general conic program like (3.1). This removes the need to specify the meta-parameters P, Q, p and L by hand and leverages theorem 3.2 as a highly accessible modeling tool.

3.3. Conic quadratic programming

We consider next the case where $K = Q_{k+1}$ is the $(k+1)$-dimensional second order cone and where the conic inequality $g(x,\theta) \preceq_K 0$ can be written as

$$\left\| \begin{pmatrix} g(x,\theta)_2 \\ \vdots \\ g(x,\theta)_{k+1} \end{pmatrix} \right\| \leqslant g(x,\theta)_1. \tag{3.9}$$

Ben-Tal and Nemirovski [5] have shown that if $g(x,\theta)_1$ is certain and Θ is given as the intersection of ellipsoids, then it is NP-hard to decide for a fixed x whether $g(x,\theta) \preceq_K 0$ holds for all $\theta \in \Theta$. Therefore we need to restrict the uncertainty set Θ more strongly than in the case of an affine constraint. In the following we present a restriction that still yields a powerful framework to define uncertain SOCPs. First, we require that the uncertainty affects both sides of inequality (3.9) independently. This leaves us with the study of robust constraints of the form

$$\|A(x)\eta + a(x)\| \leqslant \zeta^T B x + b_x^T x + b_\zeta^T \zeta + \beta \quad \forall \ (\eta,\zeta) \in H \times Z \tag{3.10}$$

where

- $H \subseteq \mathbb{R}^h$ and $Z \subseteq \mathbb{R}^z$ are perturbation sets with dimensions $h, z \in \mathbb{Z}_{\geqslant 1}$,
- $A(x) \in \mathbb{R}^{k \times h}$ and $a(x) \in \mathbb{R}^k$ are both affine in x and where
- $B \in \mathbb{R}^{z \times n}$, $b_x \in \mathbb{R}^n$, $b_\zeta \in \mathbb{R}^z$ and $\beta \in \mathbb{R}$ define the upper bound.

This restriction is not too bad in practice: In a typical application, the expression under the norm and the upper bound of constraint (3.9) would represent distinct quantities that are unlikely to depend on the same uncertain data. In particular, when the objective (3.1a) is to minimize an uncertain norm, then an epigraph reformulation of problem (3.1) produces a conic quadratic constraint where the upper bound is just an auxiliary variable. Next we restrict the uncertainty sets H and Z. Since the upper bound of (3.10) is affine in ζ, we assume once more that Z is given as the feasible region of an essentially strictly feasible conic program, formally

$$Z := \{\zeta \in \mathbb{R}^z \mid \exists \nu \in \mathbb{R}^w : P\zeta + Q\nu + p \in L\} \tag{3.11}$$

with $L \subseteq \mathbb{R}^l$ a proper cone, $P \in \mathbb{R}^{l\times z}$, $Q \in \mathbb{R}^{l\times w}$ and $p \in \mathbb{R}^l$ of proper sizes and where $P\zeta + Q\nu + p \in \operatorname{relint}(L)$ for some $\zeta \in \mathbb{R}^z$ and $\nu \in \mathbb{R}^w$. Lastly, we require H to be the Euclidean norm unit ball, formally

$$H := \left\{\eta \in \mathbb{R}^h \mid \|\eta\| \leqslant 1\right\}. \tag{3.12}$$

We will later see that this is equivalent to the requirement that H is an ellipsoid, which allows for more meaningful uncertainty sets. The following theorem states that under these conditions we obtain a tractable representation of the robust counterpart of an uncertain SOCP:

Theorem 3.3 (Ben-Tal et al. [4]). *The robust constraint* (3.10) *with* Z *and* H *defined according to* (3.11) *and* (3.12) *is equivalent to the following system of constraints in* x *and in the auxiliary variables* $y \in \mathbb{R}^l$ *and* $\tau, \lambda \in \mathbb{R}$*:*

$$\begin{aligned} \tau + p^T y &\leqslant b_x^T x + \beta, \\ P^T y &= Bx + b_\zeta, \end{aligned} \qquad \begin{aligned} Q^T y &= 0, \\ y &\in L^*, \end{aligned} \qquad \begin{bmatrix} \tau - \lambda & a(x)^T & 0 \\ a(x) & \tau I_k & A(x) \\ 0 & A(x)^T & \lambda I_h \end{bmatrix} \succeq 0.$$

We will need a few fundamental results for the proof. The first lemma concerns a class of positive semidefinite block matrices.

Lemma 3.4 (Schur complement lemma). *Let* $P \in \mathbb{S}^p$, $Q \in \mathbb{R}^{r\times p}$, $R \in \mathbb{S}^r$ *and*

$$A := \begin{bmatrix} P & Q^T \\ Q & R \end{bmatrix}$$

for some $r, p \in \mathbb{Z}_{\geqslant 1}$*. Then, the following implications are true:*

$$R \succ 0 \implies (A \succeq 0 \iff P - Q^T R^{-1} Q \succeq 0), \tag{3.13}$$

$$P \succ 0 \implies (A \succeq 0 \iff R - Q P^{-1} Q^T \succeq 0). \tag{3.14}$$

Proof. We prove only implication (3.13), the proof of (3.14) is similar. It is well known that for $u \in \mathbb{R}^p$ fixed, the convex quadratic problem

$$\underset{v \in \mathbb{R}^r}{\text{minimize}} \quad v^T R v + 2u^T Q^T v$$

has the unique optimum solution $-R^{-1}Qu$ (to see this, solve $\nabla\left(v^T R v + 2u^T Q^T v\right) = 0$). This gives

$$\begin{aligned} & A \succeq 0 \\ \iff \quad & \forall z \in \mathbb{R}^{p+r} : z^T A z \geqslant 0 \\ \iff \quad & \forall (u, v) \in \mathbb{R}^p \times \mathbb{R}^r : u^T P u + 2u^T Q^T v + v^T R v \geqslant 0 \\ \iff \quad & \forall u \in \mathbb{R}^p : \inf_{v \in \mathbb{R}^v} \left(u^T P u + 2u^T Q^T v + v^T R v\right) \geqslant 0 \\ \iff \quad & \forall u \in \mathbb{R}^p : u^T P u + 2u^T Q^T (-R^{-1} Q u) + (-R^{-1} Q u)^T R (-R^{-1} Q u) \geqslant 0 \\ \iff \quad & \forall u \in \mathbb{R}^p : u^T \left(P - Q^T R^{-1} Q\right) u \geqslant 0 \\ \iff \quad & P - Q^T R^{-1} Q \succeq 0. \end{aligned}$$

□

The second lemma builds upon the first and allows us to write a conic quadratic inequality as a linear matrix inequality.

Lemma 3.5 (SDP representation of SOCP). *For* $t \in \mathbb{R}$ *and* $y \in \mathbb{R}^n$ *it is*

$$\begin{pmatrix} t \\ y \end{pmatrix} \in \mathcal{Q}_{n+1} \quad \iff \quad \begin{bmatrix} t & y^T \\ y & tI_n \end{bmatrix} \succeq 0.$$

Proof. Let A be the block matrix in the lemma. We distinguish three cases with respect to t. If $t < 0$, then clearly $\|y\| \not\leqslant t$ and $A \not\succeq 0$. If $t = 0$, then $\|y\| \leqslant t$ if and only if $y = 0$ and also $A \succeq 0$ if and only if $y = 0$ as otherwise $\begin{bmatrix} -\frac{1}{2} & y^T \end{bmatrix} A \begin{bmatrix} -\frac{1}{2} & y^T \end{bmatrix}^T = -\|y\|^2 < 0$. If $t > 0$, then lemma 3.4 gives us

$$A \succeq 0 \;\Leftrightarrow\; t - y^T(tI_n)^{-1}y \succeq 0 \;\Leftrightarrow\; t^{-1}y^Ty \leqslant t \;\Leftrightarrow\; \|y\|^2 \leqslant t^2 \;\Leftrightarrow\; \|y\| \leqslant t. \qquad \square$$

The third lemma establishes a connection between dependent quadratic inequalities and semidefinite programming.

Lemma 3.6 (Homogeneous S-procedure). *Let* $A, B \in \mathbb{S}^n$, $n \in \mathbb{Z}_{\geqslant 1}$, *with* $z^TAz > 0$ *for some* $z \in \mathbb{R}^n$. *Then,*

$$\forall x \in \mathbb{R}^n : \left(x^TAx \geqslant 0 \implies x^TBx \geqslant 0\right) \quad \iff \quad \exists \lambda \geqslant 0 : B \succeq \lambda A.$$

Proof. The concise proof of the forward implication is due to Derinkuyu and Pınar [14]. Assume that $x^TAx \geqslant 0 \Rightarrow x^TBx \geqslant 0$ holds for all $x \in \mathbb{R}^n$ and define the sets

$$\begin{aligned} Q &:= \left\{(x^TAx, x^TBx) \mid x \in \mathbb{R}^n\right\} \quad \text{and} \\ R &:= \left\{(r_1, r_2) \in \mathbb{R}^2 \mid r_1 \geqslant 0 \wedge r_2 < 0\right\}. \end{aligned}$$

By a well-known theorem by Dines [15], the set Q is convex. Further, it is easy to see that the set R is a convex cone. By our assumption and the definition of R we have $Q \cap R = \emptyset$ so that a hyperplane through the origin separating Q and R exists. That is, there exists a point $p = (p_1, p_2) \in \mathbb{R}^2$ with $p \neq 0$ such that $\langle p, q\rangle \leqslant 0$ for all $q \in Q$ and $\langle p, r\rangle \geqslant 0$ for all $r \in R$. Since $(0, -1) \in R$, it is $p_2 \leqslant 0$. Since $(1, -\varepsilon) \in R$ for any $\varepsilon > 0$, it is $p_1 \geqslant \varepsilon p_2$ as otherwise $\langle p, r\rangle = p_1 - \varepsilon p_2 < 0$. For $\varepsilon \to 0$, we obtain $p_1 \geqslant 0$. With $\langle p, q\rangle \leqslant 0$ for all $q \in Q$ and with the additional assumption that there is a $z \in \mathbb{R}^n$ with $z^TAz > 0$, we further obtain $p_1z^TAz + p_2z^TBz \leqslant 0$. It follows that $p_2 < 0$ as otherwise it is $p_2 = 0$ (due to $p_2 \leqslant 0$) which requires also $p_1 = 0$ (due to $z^TAz > 0$) which in turn contradicts the assumption that $p \neq 0$. With $\lambda := -\frac{p_1}{p_2} \geqslant 0$, the condition of $\langle p, q\rangle \leqslant 0$ for all $q \in Q$ gives $p_1x^TAx + p_2x^TBx \leqslant 0 \Leftrightarrow x^TBx \geqslant \lambda x^TAx$ for all $x \in \mathbb{R}^n$, which is equivalent to the condition that $B \succeq \lambda A$.

For the reverse implication, assume that there is $\lambda \geqslant 0$ with $B \succeq \lambda A$. Then, it is $x^T(B - \lambda A)x \geqslant 0 \Leftrightarrow \lambda x^TAx \leqslant x^TBx$ for all $x \in \mathbb{R}^n$. We distinguish two cases. If $\lambda = 0$, then $x^TBx \geqslant 0$ follows immediately. If $\lambda > 0$, then the assumption of $x^TAx \geqslant 0$ yields $\frac{1}{\lambda}x^TBx \geqslant 0 \Leftrightarrow x^TBx \geqslant 0$ as well. $\square$

The fourth lemma is a small technical result that enables the S-procedure.

Lemma 3.7. *Let* $a \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ *and* $c \in \mathbb{R}$. *Then, it is*

$$-\|a\|\|b\| \geqslant c \quad \iff \quad \left(\forall x \in \mathbb{R}^n : \|x\| \leqslant \|b\| \implies a^T x \geqslant c\right).$$

Proof. For the forward implication, the Cauchy-Schwarz inequality gives us

$$\begin{aligned} -\|a\|\|b\| \geqslant c \quad &\implies \quad (\forall x \in \mathbb{R}^n : \|x\| \leqslant \|b\| \implies -\|a\|\|x\| \geqslant c) \\ &\implies \quad \left(\forall x \in \mathbb{R}^n : \|x\| \leqslant \|b\| \implies -|a^T x| \geqslant c\right) \\ &\implies \quad \left(\forall x \in \mathbb{R}^n : \|x\| \leqslant \|b\| \implies a^T x \geqslant c\right). \end{aligned}$$

For the reverse implication, choose $x = -a\|a\|^{-1}\|b\|$ so that $\|x\| = \|b\|$ and

$$-\|a\|\|b\| = -\frac{\|a\|^2}{\|a\|}\|b\| = -a^T a\frac{\|b\|}{\|a\|} = a^T x \geqslant c. \qquad \square$$

We are now prepared to prove the theorem.

Proof of theorem 3.3. The independent perturbation sets let us handle both sides of constraint (3.10) separately by virtue of

$$\begin{aligned} & \|A(x)\eta + a(x)\| \leqslant \zeta^T B x + b_x^T x + b_\zeta^T \zeta + \beta \ \ \forall \ \ (\eta, \zeta) \in H \times Z \\ \iff \quad & \underbrace{\sup_{\eta \in H} \|A(x)\eta + a(x)\|}_{\varphi} \leqslant \underbrace{\inf_{\zeta \in T} \left(\zeta^T B x + b_x^T x + b_\zeta^T \zeta + \beta\right)}_{\omega} \\ \iff \quad & \exists \tau \in \mathbb{R} : \varphi \leqslant \tau \wedge \omega \geqslant \tau. \end{aligned}$$

First, we establish an explicit representation for $\omega \geqslant \tau$. We have

$$\begin{aligned} & \omega \geqslant \tau \\ \iff \quad & \inf_{\zeta \in T} \left(\zeta^T B x + b_x^T x + b_\zeta^T \zeta + \beta\right) \geqslant \tau \\ \iff \quad & \sup_{\zeta \in T} \left(-\zeta^T B x - b_x^T x - b_\zeta^T \zeta - \beta\right) \leqslant -\tau \\ \iff \quad & \sup_{\zeta \in T} \left(\tau - \zeta^T B x - b_x^T x - b_\zeta^T \zeta - \beta\right) \leqslant 0. \end{aligned}$$

The remainder of the argument is analogous to the proof of theorem 3.2; we identify the coinciding terms as follows:

Proof of	
theorem 3.2	theorem 3.3
θ	ζ
A	$-B$
a_x	$-b_x$
a_θ	$-b_\zeta$
α	$\tau - \beta$

All other relevant identifiers are equivalent in both proofs. This gives us

$$\omega \geqslant \tau \iff \exists y \in \mathbb{R}^l : \begin{cases} p^T y - b_x^T + (\tau - \beta) \leqslant 0, \\ P^T y - Bx - b_\zeta = 0, \\ Q^T y = 0, \\ y \in L^*. \end{cases}$$

By straightforward rearrangement we obtain four out of the five constraints posed in theorem 3.3. Next, we establish an explicit representation for $\varphi \leqslant \tau$. In the following $\forall(t,u)$ is short for $\forall(t,u) \in \mathbb{R} \times \mathbb{R}^k$ and $\forall(t,u,v)$ is short for $\forall(t,u,v) \in \mathbb{R} \times \mathbb{R}^k \times \mathbb{R}^h$. It is

$$\begin{aligned}
& & \varphi &\leqslant \tau \\
&\iff & \forall \eta \in H : \|A(x)\eta + a(x)\| &\leqslant \tau \\
&\overset{(a)}{\iff} & \forall \eta \in H : \begin{bmatrix} \tau & (A(x)\eta + a(x))^T \\ A(x)\eta + a(x) & \tau I_k \end{bmatrix} &\succeq 0 \\
&\iff & \forall \eta \in H : \forall(t,u) : \tau t^2 + 2tu^T (A(x)\eta + a(x)) + \tau u^T u &\geqslant 0 \\
&\iff & \forall(t,u) : \tau t^2 + 2 \inf_{\eta \in H} tu^T A(x)\eta + 2tu^T a(x) + \tau u^T u &\geqslant 0 \\
&\overset{(b)}{\iff} & \forall(t,u) : \tau t^2 - 2\left\|u^T A(x)\right\|_2 |-t| + 2tu^T a(x) + \tau u^T u &\geqslant 0 \\
&\overset{(c)}{\iff} & \forall(t,u,v) : \left(v^T v \leqslant t^2 \Rightarrow \tau t^2 + 2u^T A(x) v + 2tu^T a(x) + \tau u^T u \geqslant 0\right)
\end{aligned}$$

where $\|\cdot\|_2$ denotes the operator 2-norm. Equivalence (a) is due to lemma 3.5. Equivalence (b) is due to the inf/sup duality and by the definition of the operator 2-norm which is absolutely homogeneous. Equivalence (c) is due to lemma 3.7 with

$$\left\|u^T A(x)\right\|_2 = \left\|A(x)^T u\right\|, \qquad |-t| = \|t\|, \qquad v^T v \leqslant t^2 \iff \|v\| \leqslant \|t\|.$$

Now, the last statement is equivalent to the claim that the implication

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix}^T \begin{bmatrix} 1 & & \\ & 0 & \\ & & -I_h \end{bmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} \geqslant 0 \quad \Longrightarrow \quad \begin{pmatrix} t \\ u \\ v \end{pmatrix}^T \begin{bmatrix} \tau & a(x)^T & 0 \\ a(x) & \tau I_k & A(x) \\ 0 & A(x)^T & 0 \end{bmatrix} \begin{pmatrix} t \\ u \\ v \end{pmatrix} \geqslant 0$$

holds for all $(t,u,v) \in \mathbb{R} \times \mathbb{R}^k \times \mathbb{R}^h$. By lemma 3.6, this is further equivalent to

$$\exists \lambda \geqslant 0 : \begin{bmatrix} \tau - \lambda & a(x)^T & 0 \\ a(x) & \tau I_k & A(x) \\ 0 & A(x)^T & \lambda I_h \end{bmatrix} \succeq 0.$$

The restriction that λ must be nonnegative can be omitted as λ is a diagonal element of the above matrix, which cannot be positive semidefinite for $\lambda < 0$. This yields the remaining constraint posed in theorem 3.3 and concludes the proof. □

We remark that the theorem proven by Ben-Tal et al. [4] is slightly more powerful: The perturbation parameter η is allowed to be a matrix in the operator 2-norm unit ball. For this to work, the expression under the norm must be decomposed in a certain way. This extension is not available in PICOS yet.

It remains to show that we can relax the restriction on H and allow it to be a (shifted) ellipsoid, which renders theorem 3.3 significantly more useful in practice. More precisely, we allow the parameter η to be supported on a bijective affine transformation of the Euclidean unit ball, formally $\eta \in H' := \{C\xi + c \mid \xi \in \mathbb{R}^h \wedge \|\xi\| \leqslant 1\}$ with $C \in \mathbb{R}^{h\times h}$ invertible and $c \in \mathbb{R}^h$. Then, it is $A(x)\eta + a(x) = A(x)C\xi + A(x)c + a(x)$. Since $A(x)C$ and $A(x)c + a(x)$ are both affine in x and the support of ξ is the original perturbation set H, we can immediately invoke theorem 3.3 by identifying the terms in the theorem as follows:

$\eta \in H$	$\eta \in H'$
$A(x)$	$A(x)C$
$a(x)$	$A(x)c + a(x)$

Note that we did not use the fact that C is invertible. However, this assumption is significant when a PICOS user defines the uncertainty set H' by providing suitable bounds on η, producing an alternative representation of the form $H' = \{\eta \in \mathbb{R}^h \mid \|D\eta + d\| \leqslant r\}$ with $D \in \mathbb{R}^{h\times h}$ invertible, $d \in \mathbb{R}^h$ and $r > 0$. This can be rearranged into the form above by setting $C := rD^{-1}$ and $c := D^{-1}d$. PICOS will attempt this transformation whenever a general form conic perturbation set is used to represent H.

4. Distributionally robust optimization

It is common practice to assume that uncertain data is generated by a random distribution. Given that the distribution is completely known and has nice analytic properties, one can use the robust optimization methods discussed in section 3 to model a data confidence region corresponding to the bounded support or to a desired percentile of the distribution. This yields a solution that captures the worst-case scenario with high probability but that disregards all distributional information except for the shape of one particular superlevel set of the probability density function. An alternative, classic approach that leverages all distributional information provided by the modeler is stochastic programming. Here, optimization is performed with respect to *expected* as opposed to worst-case function values.[2] While posing a constraint on an expected value may be of questionable use, minimizing expected costs or maximizing expected gain are very natural optimization goals that are neatly captured by stochastic programming.

In practice, however, data generating distributions are rarely known with certainty and need to be estimated from the observed data and auxiliary assumptions. Such an

[2] We consider only expectation throughout this document, though other functionals are possible.

estimate is in itself a new source of errors, which dilutes the probabilistic guarantees provided by robust methods and reduces the appeal of expectation objectives. It is therefore the natural next step to give not just the anticipated data uncertainty but also the associated *estimation uncertainty* an explicit representation in the optimization model, so that we can capture our beliefs about the data more accurately.

Distributionally robust (stochastic) optimization (DRO), an approach first applied in a 1958 paper by Scarf [40] to tackle a newsvendor problem, can be understood as stochastic programming with robust optimization applied to the estimation of distributional information. More precisely, the objective function (and, if desired, constraint satisfaction) is evaluated with respect to the expected value of a function of the random data but the underlying distribution is chosen in a worst-case manner from an *ambiguity set* $\mathcal{D}$ of distributions that we believe to contain the "true" distribution. From a modeling perspective, the ambiguity set controls the conservatism of the model that ranges anywhere between robust optimization and stochastic programming. If $\mathcal{D}$ is a singleton, then we obtain stochastic programming as a special case. If $\mathcal{D}$ contains all distributions with a prescribed support, then we recover robust optimization. DRO is therefore a generalization of both competing frameworks.

Following the seminal work by Scarf [40], who considers the ambiguity set of all scalar distributions with known mean and variance in order to maximize a worst-case expectation over a piecewise linear function, a great variety of ambiguity sets have been defined and studied in the context of a cautious selection of objective functions. Two solution techniques are common to deal with the semi-infinite nature of DRO models [35]: Cutting-plane methods that solve for a series of finite inner approximations of the ambiguity set and methods that solve a finite representation of the dual problem. Since not all solvers supported by PICOS can leverage information from previous solution attempts, we prefer approaches powered by strong duality arguments which transform the DRO model into just one equivalent problem that we can hand to a solver at once. For the definition of the ambiguity set, there are again two major schools of thought [35]: Moment-based ambiguity sets contain all distributions whose first few moments satisfy desired properties while discrepancy-based ambiguity sets contain all distributions that are close to a nominal distribution with respect to a certain metric. We will discuss and implement results for both approaches.

Formally, a minimization objective in the DRO framework is given as

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \sup\limits_{\Xi \in \mathcal{D}} \mathbb{E}_{\xi \sim \Xi}\left[f(x, \xi)\right] \end{array}$$

where $\mathcal{D}$ is a nonempty set of random distributions and where f is Ξ-integrable and usually nonlinear in the random variable $\xi \in \mathbb{R}^m$. This corresponds to the robust objective (3.2a) in the robust optimization framework. Performing an epigraph reformulation, it can be shown that we lose nothing when we limit our study to distributionally robust

expectation constraints of the form

$$\sup_{\Xi \in \mathcal{D}} \mathbb{E}_{\xi \sim \Xi} [g(x, \xi)] \leqslant \omega \tag{4.1}$$

where g is Ξ-integrable and where ω is an auxiliary variable or more generally a scalar affine expression whose coefficients are known with certainty. In the following we will establish conditions on $\mathcal{D}$ and g that enable the handling of such constraints using general purpose conic programming solvers.

4.1. Moment-ambiguous stochastic programming

If we are presented with repeated measurements under equal circumstances, then it may be reasonable to estimate the first few moments of the data generating distribution without making an assumption on the family that it belongs to. If the estimate is limited to the first two moments, mean and variance, and if we assume that the "true" moments are bounded by a function of these estimates, then the permissible distributions are said to belong to a *Chebyshev ambiguity set* [35]. If we further define the bounds on the mean as an ellipsoid shaped according to the estimated covariance matrix, take a multiple of the latter as an upper bound (with respect to the Loewner order) to the "true" variance and if we additionally pose a bound on the distribution's support, then we arrive at the ambiguity set studied by Delage and Ye [13],

$$\mathcal{D}_{\text{DY}} := \left\{ \Xi \in \mathcal{M} \;\middle|\; \begin{array}{r} \mathbb{P}(\xi \in \mathcal{S}) = 1, \\ (\mathbb{E}[\xi] - \mu)^T \Sigma^{-1} (\mathbb{E}[\xi] - \mu) \leqslant \alpha, \\ \mathbb{E}\left[(\xi - \mu)(\xi - \mu)^T\right] \preceq \beta \Sigma \end{array} \right\},$$

where $\mathcal{M}$ is the set of all Borel probability measures on $\mathbb{R}^m$, probability and expectation are measured with respect to $\xi \sim \Xi$, $\mathcal{S} \subseteq \mathbb{R}^m$ is the closed, convex sample space posing a reasonable bound on the support (such that $\mathcal{D}_{\text{DY}}$ is nonempty), $\mu \in \mathbb{R}^m$ and $\Sigma \in \mathbb{S}^m$ with $\Sigma \succ 0$ are the estimated mean and covariance matrix and where $\alpha \geqslant 0$ and $\beta \geqslant 1$ are parameters representing our degree of trust in these estimates.

The authors give a thorough probabilistic justification for using this particular ambiguity set in the context of data-driven optimization, which goes beyond the scope of this document. Modelers using PICOS' capabilities are advised to refer to [13] if they are looking to parameterize their instance of $\mathcal{D}_{\text{DY}}$ with probabilistic guarantees in mind.

A central result concerning the ambiguity set $\mathcal{D}_{\text{DY}}$ is the following lemma.

Lemma 4.1 (Delage and Ye [13])**.** *The constraint* (4.1) *for* $\mathcal{D} = \mathcal{D}_{DY}$ *is equivalent to the following semi-infinite system of constraints in auxiliary variables* $Q \in \mathbb{S}^m$*,*

$q \in \mathbb{R}^m$ *and* $r, t \in \mathbb{R}$*:*

$$\langle \beta\Sigma + \mu\mu^T, Q \rangle + \mu^T q + r + t \leqslant \omega, \tag{4.2a}$$

$$g(x, \xi) - \xi^T Q \xi - q^T \xi \leqslant r \ \ \forall \ \ \xi \in \mathcal{S}, \tag{4.2b}$$

$$\| \sqrt{\alpha} \Sigma^{\frac{1}{2}} (2Q\mu + q) \| \leqslant t, \tag{4.2c}$$

$$Q \succeq 0. \tag{4.2d}$$

Proof. We first rewrite the left hand side of constraint (4.1) as a conic program. By definition of $\mathcal{D}_{\text{DY}}$, the problem

$$\underset{\Xi \in \mathcal{D}_{\text{DY}}}{\text{maximize}} \quad \mathbb{E}_{\xi \sim \Xi}[g(x, \xi)] \tag{4.3}$$

is trivially equivalent (in the sense of definition 15) to the conic moment problem

$$\underset{\Xi \in \mathcal{M}}{\text{maximize}} \quad \int_{\mathcal{S}} g(x, \xi) \, \mathrm{d}\Xi(\xi) \tag{4.4a}$$

$$\text{subject to} \quad \int_{\mathcal{S}} \mathrm{d}\Xi(\xi) = 1, \tag{4.4b}$$

$$\int_{\mathcal{S}} \begin{bmatrix} \Sigma & \xi - \mu \\ (\xi - \mu)^T & \alpha \end{bmatrix} \mathrm{d}\Xi(\xi) \succeq 0, \tag{4.4c}$$

$$\int_{\mathcal{S}} (\xi - \mu)(\xi - \mu)^T \, \mathrm{d}\Xi(\xi) \preceq \beta\Sigma \tag{4.4d}$$

where the linear matrix inequality form of constraint (4.4c) is due to the Schur complement lemma 3.4. Following a recipe by Shapiro [42], Delage and Ye state the dual of problem (4.4) as

$$\underset{P, Q, p, r, s}{\text{minimize}} \quad \langle \beta\Sigma - \mu\mu^T, Q \rangle + r + \langle \Sigma, P \rangle - 2\mu^T p + \alpha s \tag{4.5a}$$

$$\text{subject to} \quad \xi^T Q \xi - 2\xi^T (p + Q\mu) + r - g(x, \xi)) \geqslant 0 \ \ \forall \ \ \xi \in \mathcal{S}, \tag{4.5b}$$

$$\begin{bmatrix} P & p \\ p^T & s \end{bmatrix} \succeq 0, \tag{4.5c}$$

$$Q \succeq 0 \tag{4.5d}$$

where $P \in \mathbb{S}^m$, $p \in \mathbb{R}^m$ and $s \in \mathbb{R}$ jointly form the dual variable for constraint (4.4c) while $Q \in \mathbb{S}^m$ and $r \in \mathbb{R}$ are the dual variables for constraints (4.4b) and (4.4d), respectively. Invoking a strong duality argument also found in [42], the authors find that there is no duality gap, so that the optimal value of problem (4.5) remains equivalent to the left hand side of constraint (4.1).[3]

Next, we solve problem (4.5) analytically for P, p and s to get rid of constraint (4.5c). To this end we distinguish three cases with respect to s. If $s < 0$, then constraint (4.5c)

[3] Unless problem (4.3) is unbounded, in which case both problem (4.5) and constraint (4.1) are infeasible.

is infeasible as diagonal elements of positive semidefinite matrices are nonnegative. If $s = 0$, then $p = 0$ is required for constraint (4.5c) to hold as positive semidefinite matrices are diagonally dominant. In this case, constraint (4.5c) reduces to $P \succeq 0$. Since also $\Sigma \succ 0$ by assumption, we have $\langle \Sigma, P \rangle \geqslant 0$ as the cone $\mathbb{S}^m_+$ is self-dual. This means choosing $P = 0$ is optimal in this case. If $s > 0$, then s is invertible and we can invoke the Schur complement lemma 3.4. Thereby, constraint (4.5c) is equivalent to the condition that $P - \frac{1}{s} p p^T \succeq 0$. Now, $\langle \Sigma, P \rangle = \langle \Sigma, P - \frac{1}{s} p p^T \rangle + \langle \Sigma, \frac{1}{s} p p^T \rangle$ with $\langle \Sigma, P - \frac{1}{s} p p^T \rangle \geqslant 0$ by the same conic duality argument as in the last case. This means choosing $P = \frac{1}{s} p p^T$ is optimal in this case. Since p appears also in constraint (4.5b), we solve for $s > 0$ which appears only in the objective (4.5a). The latter reduces to minimizing the scalar function $h(s) := \langle \Sigma, \frac{1}{s} p p^T \rangle + \alpha s$. It is $h'(s) = -\langle \Sigma, p p^T \rangle s^{-2} + \alpha$ and $h''(s) = 2\langle \Sigma, p p^T \rangle s^{-3} \geqslant 0$, so h is convex and obtains its minimum for $\alpha = \langle \Sigma, p p^T \rangle s^{-2}$, that is at $s = \alpha^{-\frac{1}{2}} \|\Sigma^{\frac{1}{2}} p\|$.

Now, we perform the change of variables $q := -2(Q\mu + p)$ so that we can write constraint (4.5b) as constraint (4.2b). In the objective (4.5a), we substitute $p = -(Q\mu + \frac{1}{2} q)$ and obtain, with the additional substitutions from solving for P, p and $s > 0$,

$$
\begin{aligned}
& \langle \beta\Sigma - \mu\mu^T, Q \rangle + r + \langle \Sigma, P \rangle - 2\mu^T p + \alpha s \\
= \; & \langle \beta\Sigma - \mu\mu^T, Q \rangle + r + \left(\alpha^{-\frac{1}{2}} \|\Sigma^{\frac{1}{2}} p\| \right)^{-1} \langle \Sigma, p p^T \rangle + 2\mu^T \left(Q\mu + \frac{1}{2} q \right) + \alpha \alpha^{-\frac{1}{2}} \|\Sigma^{\frac{1}{2}} p\| \\
= \; & \langle \beta\Sigma - \mu\mu^T, Q \rangle + r + \sqrt{\alpha} \|\Sigma^{\frac{1}{2}} p\| + 2\mu^T Q\mu + \mu^T q + \sqrt{\alpha} \|\Sigma^{\frac{1}{2}} p\| \\
= \; & \langle \beta\Sigma + \mu\mu^T, Q \rangle + \mu^T q + r + \|\sqrt{\alpha} \Sigma^{\frac{1}{2}} (2Q\mu + q)\|.
\end{aligned}
$$

For the remaining case where P, p and s are all zero we can prove the same equivalence. Here it is $q = -2Q\mu$ and thus

$$
\begin{aligned}
& \langle \beta\Sigma - \mu\mu^T, Q \rangle + r + \langle \Sigma, P \rangle - 2\mu^T p + \alpha s \\
= \; & \langle \beta\Sigma - \mu\mu^T, Q \rangle + r \\
= \; & \langle \beta\Sigma + \mu\mu^T, Q \rangle - 2\mu^T Q\mu + r + \|\sqrt{\alpha} \Sigma^{\frac{1}{2}} (2Q\mu - 2Q\mu)\| \\
& \langle \beta\Sigma + \mu\mu^T, Q \rangle + \mu^T q + r + \|\sqrt{\alpha} \Sigma^{\frac{1}{2}} (2Q\mu + q)\|.
\end{aligned}
$$

Lastly, we introduce an auxiliary variable t and the additional constraint (4.2c) so that we can replace the nonlinear term $\|\sqrt{\alpha} \Sigma^{\frac{1}{2}} (2Q\mu + q)\|$ in the objective by t. We finally arrive at the system of constraints (4.2) by substituting the resulting linear objective function for the left hand side of constraint (4.1) to obtain constraint (4.2a). □

As constraint (4.2b) represents an infinite number of constraints, parameterized over $\mathcal{S}$, Delage and Ye [13] build upon the lemma with a set of conditions on the set $\mathcal{S}$ and the function g that yield a tractable solution technique based on the ellipsoid method. While we state this interesting result in the following, we are unable to make good use of it within PICOS as the library depends on the user's choice of a third party conic optimization solver and does not have reliable access to low level solution algorithms

such as the ellipsoid method. However, in the next section we will obtain a finite representation of constraint (4.2b) for an interesting class of problems that includes a distributionally robust variant of the *stochastic robust least-squares* problem.

Proposition 4.2 (Delage and Ye [13])**.** *Let the conditions posed in lemma 4.1 be extended as follows:*

1. *In addition to being closed and convex, the sample space $\mathcal{S}$ is bounded and equipped with an oracle that can, in time polynomial in m, either confirm that $\xi \in \mathcal{S}$ holds for some $\xi \in \mathbb{R}^m$ or provide a hyperplane that separates ξ and $\mathcal{S}$.*
2. *The function g has the form $g(x,\xi) = \max_{k=1}^{K} g_k(x,\xi)$ for some $K \in \mathbb{Z}_{\geqslant 1}$ and, for every $k \in [K]$,*
   a) *g_k is concave in ξ,*
   b) *one can evaluate g_k in polynomial time and*
   c) *one can find a subgradient of g_k in ξ in polynomial time.*

Then, problem (4.3) *is convex and feasible and one can solve it to any precision ε in time polynomial in $\log(\frac{1}{\varepsilon})$ and the size of the problem.*

As we will not make theoretical or practical use of this proposition, we refer the reader to Delage and Ye [13] for a proof.

4.1.1. Application to least-squares problems

In order to leverage the moment-ambiguous stochastic programming framework developed by Delage and Ye [13] within PICOS, we are looking for functions g and sets $\mathcal{S}$ that equip constraint (4.1) with significant modeling power and that yield a finite and conic representation for constraint (4.2b) of lemma 4.1. It turns out that both boxes are ticked when we choose $g(x,\xi)$ as the squared norm $\|a(x,\xi)\|^2$ where $a(x,\xi) \in \mathbb{R}^k$ is biaffine in x and ξ and when $\mathcal{S}$ is a solid ellipsoid. A special case of this choice of g is the uncertain least-squares formulation $\|A(\xi)x - b(\xi)\|^2$ (with A and b affine in ξ), wich means that we can apply the framework in the context of linear regression models. As this choice of g does not meet the assumption that g is concave in ξ, we cannot confirm tractability by invoking proposition 4.2. However, we are able to extend a subsequent result by Mehrotra and Zhang [31], who show tractability for the special case of $g(x,\xi) = \|(A + \xi_A)x - (b + \xi_b)\|^2$ and $\mathcal{S} = \{y \mid \|y\| \leqslant \rho\}$ where A, b and $\rho > 0$ are constant and where $\xi^T = \begin{pmatrix} \operatorname{vec}(\xi_A)^T & \xi_b \end{pmatrix}$. More precisely, we build upon a finite representation of constraint (4.2b) in the form of a linear matrix inequality that was proposed by an anonymous referee of [31]. This yields the following result that we have implemented in PICOS.

Theorem 4.3. *The constraint* (4.1) *for $\mathcal{D} = \mathcal{D}_{DY}$ and $g(x,\xi) = \|a(x,\xi)\|^2$ with $a(x,\xi) \in \mathbb{R}^k$ biaffine in x and ξ, and for $\mathcal{S} = \{Dy + d \mid \|y\| \leqslant 1\}$ with $D \in \mathbb{R}^{m \times m}$*

invertible, $d \in \mathbb{R}^m$ *and with* $G := D^{-T}D^{-1}$ *is equivalent to the following system of constraints in auxiliary variables* $Q \in \mathbb{S}^m$, $q \in \mathbb{R}^m$ *and* $r, t, \lambda \in \mathbb{R}$*:*

$$\langle \beta\Sigma + \mu\mu^T, Q\rangle + \mu^T q + r + t \leqslant \omega, \tag{4.6a}$$

$$\|\sqrt{\alpha}\Sigma^{\frac{1}{2}}(2Q\mu + q)\| \leqslant t, \tag{4.6b}$$

$$\begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & \lambda G + Q & \frac{1}{2}q - \lambda G d \\ b(x)^T & \left(\frac{1}{2}q - \lambda G d\right)^T & \lambda\left(d^T G d - 1\right) + r \end{bmatrix} \succeq 0, \tag{4.6c}$$

$$Q \succeq 0,\ \lambda \geqslant 0, \tag{4.6d}$$

where $B(x)\xi + b(x)$ *is the unique factorization of* $a(x, \xi)$ *with* $B(x) \in \mathbb{R}^{k\times m}$ *and* $b(x) \in \mathbb{R}^k$ *both affine in* x*.*

To prove the theorem, we need another variant of the S-procedure that we introduced as lemma 3.6.

Lemma 4.4 (Inhomogeneous S-procedure)**.** *Let* $n \in \mathbb{Z}_{\geqslant 1}$, $A, B \in \mathbb{S}^n$, $a, b \in \mathbb{R}^n$ *and* $\alpha, \beta \in \mathbb{R}$ *with* $z^T A z + 2a^T z + \alpha < 0$ *for some* $z \in \mathbb{R}^n$*. Then,*

$$\forall x \in \mathbb{R}^n : \left(x^T A x + 2a^T x + \alpha \leqslant 0 \Rightarrow x^T B x + 2b^T x + \beta \leqslant 0\right)$$

$$\iff \quad \exists \lambda \geqslant 0 : \begin{bmatrix} A & a \\ a^T & \alpha \end{bmatrix} \lambda - \begin{bmatrix} B & b \\ b^T & \beta \end{bmatrix} \succeq 0.$$

This variant follows from lemma 3.6 by a homogenization argument: The quadratic functions are mapped to quadratic forms with respect to an augmented variable vector of length $n + 1$. It is shown that both the assumption of strict feasibility as well as both statements that the lemma claims to be equivalent are invariant with respect to this mapping. Lemma 4.4 then follows by applying lemma 3.6. We refer the reader to the appendices of Ben-Tal et al. [4] for a formal proof.

We are now prepared to prove the theorem.

Proof of theorem 4.3. In order to obtain a finite representation of constraint (4.2b) for our choice of g and $\mathcal{S}$, we first rewrite g under the factorization $a(x, \xi) = B(x)\xi + b(x)$:

$$\begin{aligned} g(x, \xi) &= \|B(x)\xi + b(x)\|^2 \\ &= \xi^T \left(B(x)^T B(x)\right) \xi + 2\left(B(x)^T b(x)\right)^T \xi + \|b(x)\|^2. \end{aligned}$$

With $G := D^{-T}D^{-1} \in \mathbb{S}^m$, we can write the condition $\xi \in \mathcal{S}$ as a quadratic inequality:

$$\begin{aligned} & \xi \in \{Dy + d \mid \|y\| \leqslant 1\} \\ \iff \quad & \xi \in \{y \mid \|D^{-1}(y - d)\| \leqslant 1\} \\ \iff \quad & \|D^{-1}(\xi - d)\|^2 \leqslant 1 \\ \iff \quad & (\xi - d)^T G(\xi - d) \leqslant 1 \\ \iff \quad & \xi^T G \xi - 2(Gd)^T \xi + d^T G d - 1 \leqslant 0. \end{aligned}$$

In combination, this lets us rewrite constraint (4.2b) as follows:

$$
\begin{aligned}
& g(x,\xi) \leqslant \xi^T Q \xi + q^T \xi + r \;\; \forall \;\; \xi \in \mathcal{S} \\
\iff \quad & \xi^T \left(B(x)^T B(x)\right) \xi + 2 \left(B(x)^T b(x)\right)^T \xi + \|b(x)\|^2 \leqslant \xi^T Q \xi + q^T \xi + r \;\; \forall \;\; \xi \in \mathcal{S} \\
\iff \quad & \xi^T \underbrace{\left(B(x)^T B(x) - Q\right)}_{C(x,Q)} \xi + 2 \underbrace{\left(B(x)^T b(x) - \frac{1}{2} q\right)^T}_{c(x,q)^T} \xi + \underbrace{\|b(x)\|^2 - r}_{\gamma(x,r)} \leqslant 0 \;\; \forall \;\; \xi \in \mathcal{S} \\
\iff \quad & \left(\xi \in \{Dy + d \mid \|y\| \leqslant 1\} \implies \xi^T C(x,Q) \xi + 2c(x,q)^T \xi + \gamma(x,r) \leqslant 0\right) \\
\iff \quad & \forall \xi \in \mathbb{R}^m : \left(\begin{array}{rc} & \xi^T G \xi - 2(Gd)^T \xi + d^T G d - 1 \leqslant 0 \\ \implies & \xi^T C(x,Q) \xi + 2c(x,q)^T \xi + \gamma(x,r) \leqslant 0 \end{array} \right).
\end{aligned}
$$

This is the required form for lemma 4.4; the left hand side inequality is strictly feasible by the definition of $\mathcal{S}$ as an ellipsoid with nonempty interior (due to D invertible). It follows that constraint (4.2a) is further equivalent to the statement that there is a $\lambda \in \mathbb{R}$ with $\lambda \geqslant 0$ such that

$$
\begin{aligned}
& \begin{bmatrix} \lambda G - B(x)^T B(x) + Q & -\lambda G d - B(x)^T b(x) + \frac{1}{2} q \\ -\lambda (Gd)^T - \left(B(x)^T b(x)\right)^T + \frac{1}{2} q^T & \lambda \left(d^T G d - 1\right) - \|b(x)\|^2 + r \end{bmatrix} \succeq 0 \\
\iff \quad & \begin{bmatrix} \lambda G + Q & \frac{1}{2} q - \lambda G d \\ \left(\frac{1}{2} q - \lambda G d\right)^T & \lambda \left(d^T G d - 1\right) + r \end{bmatrix} - \begin{bmatrix} B(x)^T \\ b(x)^T \end{bmatrix} \begin{bmatrix} B(x) & b(x) \end{bmatrix} \succeq 0 \\
\iff \quad & \begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & \lambda G + Q & \frac{1}{2} q - \lambda G d \\ b(x)^T & \left(\frac{1}{2} q - \lambda G d\right)^T & \lambda \left(d^T G d - 1\right) + r \end{bmatrix} \succeq 0
\end{aligned}
$$

where the last equivalence is due to the Schur complement lemma 3.4. Now, the theorem follows immediately from lemma 4.1. □

It is possible that a user does not want to bound all three of mean, covariance and support, that is they may want to parameterize $\mathcal{D}_{DY}$ with $\alpha \to \infty$, $\beta \to \infty$ or $\mathcal{S} = \mathbb{R}^m$. In the following we provide extensions of theorem 4.3 that handle these special parameters. To remove the bound on the support, we will need the following lemma.

Lemma 4.5 (Semidefinite quadratic expression). *Let $A \in \mathbb{S}^n$, $a \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$ and*

$$q(x) := x^T A x + 2a^T x + \alpha.$$

Then,

$$\underbrace{\forall x \in \mathbb{R}^n : q(x) \geqslant 0}_{P(q)} \quad \iff \quad \begin{bmatrix} A & a \\ a^T & \alpha \end{bmatrix} \succeq 0.$$

Proof. This is a corollary of lemma 4.4 but we provide a short direct proof. Let $r : \mathbb{R}^{(n+1)} \to \mathbb{R}$ with $r(x, z) := x^T A x + 2z a^T x + z^2 \alpha$. By the definition of $\mathbb{S}^n_+$,

$$\begin{bmatrix} A & a \\ a^T & \alpha \end{bmatrix} \succeq 0 \quad \iff \quad \underbrace{\forall (x, z) \in \mathbb{R}^{(n+1)} : r(x, z) \geqslant 0}_{P(r)}.$$

We show $P(q) \Leftrightarrow P(r)$. For $P(r) \Rightarrow P(q)$, choose $z = 1$. For $P(q) \Rightarrow P(r)$, first assume that $P(q)$ holds and let $z \in \mathbb{R}$ with $z \neq 0$. Then, rescaling $x \mapsto \frac{x}{z}$ in $q(x)$ leaves $P(q)$ invariant and multiplying both sides of $q(\frac{x}{z}) \geqslant 0$ with $z^2 > 0$ yields

$$\forall (x, z) \in \mathbb{R}^{(n+1)} : z \neq 0 \Rightarrow r(x, z) \geqslant 0.$$

Next, let $z = 0$ in $r(x, z)$ so that $P(r)$ simplifies to the condition that $A \succeq 0$. Assume towards a contradiction that both $P(q)$ and $\neg P(r)$ hold. Then, due to $\neg P(r) \Leftrightarrow A \not\succeq 0$, there is an $\bar{x}$ such that $\bar{y} := \bar{x}^T A \bar{x} < 0$ and we obtain

$$\begin{aligned} \lim_{\rho \to \infty} q(\rho \bar{x}) &= \lim_{\rho \to \infty} (\rho \bar{x})^T A (\rho \bar{x}) + 2a^T (\rho \bar{x}) + \alpha \\ &= \lim_{\rho \to \infty} \rho^2 \bar{y} + \rho (2a^T \bar{x}) + \alpha < 0. \end{aligned}$$

Since $q(x)$ is continuous, it follows that there is also a $\bar{\rho} < \infty$ with $q(\bar{\rho} \bar{x}) < 0$. □

Unbounded mean. For $\alpha \to \infty$ we have to substitute $q = -2Q\mu$ as otherwise it is $\|\sqrt{\alpha} \Sigma^{\frac{1}{2}} (2Q\mu + q)\| \to \infty$. This lets us choose $t = 0$ so that constraint (4.6a) reduces to $\langle \beta \Sigma - \mu\mu^T, Q \rangle + r \leqslant \omega$ while constraint (4.6b) becomes tautological.

Corollary 4.6. *If we let $\alpha \to \infty$ in theorem 4.3, then the system* (4.6) *reduces to*

$$\begin{gathered} \langle \beta \Sigma - \mu\mu^T, Q \rangle + r \leqslant \omega, \quad Q \succeq 0, \quad \lambda \geqslant 0, \\ \begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & \lambda G + Q & -Q\mu - \lambda G d \\ b(x)^T & (-Q\mu - \lambda G d)^T & \lambda \left(d^T G d - 1 \right) + r \end{bmatrix} \succeq 0. \end{gathered}$$

Unbounded covariance. For $\beta \to \infty$ we have to choose $Q = 0$ as otherwise it is $\langle \beta \Sigma + \mu\mu^T, Q \rangle \to \infty$ due to $\langle \Sigma, Q \rangle > 0$ and $\langle \mu\mu^T, Q \rangle \geqslant 0$ by self-duality of $\mathbb{S}^m_+$.

Corollary 4.7. *If we let $\beta \to \infty$ in theorem 4.3, then the system* (4.6) *reduces to*

$$\begin{gathered} \mu^T q + r + t \leqslant \omega, \quad \|\sqrt{\alpha} \Sigma^{\frac{1}{2}} q\| \leqslant t, \quad \lambda \geqslant 0, \\ \begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & \lambda G & \frac{1}{2} q - \lambda G d \\ b(x)^T & \left(\frac{1}{2} q - \lambda G d \right)^T & \lambda \left(d^T G d - 1 \right) + r \end{bmatrix} \succeq 0. \end{gathered}$$

Unbounded support. For $\mathcal{S} = \mathbb{R}^m$ we can amend the proof of theorem 4.3 such that constraint (4.2b) is now equivalent to

$$\xi^T C(x,Q)\xi + 2c(x,q)^T\xi + \gamma(x,r) \leqslant 0 \;\; \forall \;\; \xi \in \mathbb{R}^m$$

$$\iff \quad \begin{bmatrix} -C(x,Q) & -c(x,q) \\ -c(x,q)^T & -\gamma(x,r) \end{bmatrix} = \begin{bmatrix} Q & \frac{1}{2}q \\ \frac{1}{2}q^T & r \end{bmatrix} - \begin{bmatrix} B(x)^T \\ b(x)^T \end{bmatrix} \begin{bmatrix} B(x) & b(x) \end{bmatrix} \succeq 0.$$

by lemma 4.5. Applying lemma 3.4, constraint (4.2b) is further equivalent to

$$\begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & Q & \frac{1}{2}q \\ b(x)^T & \frac{1}{2}q^T & r \end{bmatrix} \succeq 0.$$

As diagonal blocks of positive semidefinite block matrices are positive semidefinite, the constraint $Q \succeq 0$ becomes redundant. This gives the following corollary.

Corollary 4.8. *The constraint* (4.1) *for* $\mathcal{D} = \mathcal{D}_{DY}$ *and* $g(x,\xi) = \|a(x,\xi)\|^2$ *with* $a(x,\xi) \in \mathbb{R}^k$ *biaffine in* x *and* ξ*, and for* $\mathcal{S} = \mathbb{R}^m$ *is equivalent to the following system of conic constraints in auxiliary variables* $Q \in \mathbb{S}^m$*,* $q \in \mathbb{R}^m$ *and* $r, t \in \mathbb{R}$*:*

$$\begin{aligned} \langle \beta\Sigma + \mu\mu^T, Q\rangle + \mu^T q + r + t &\leqslant \omega, \\ \|\sqrt{\alpha}\Sigma^{\frac{1}{2}}(2Q\mu + q)\| &\leqslant t, \end{aligned} \qquad \begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & Q & \frac{1}{2}q \\ b(x)^T & \frac{1}{2}q^T & r \end{bmatrix} \succeq 0.$$

Unbounded mean and covariance. This resembles a robust optimization model with an ellipsoidal uncertainty set where constraint (4.1) reduces to $\|a(x,\xi)\|^2 \leqslant \omega \;\; \forall \;\; \xi \in \mathcal{S}$. We have $Q = 0$ and $q = -2Q\mu = 0$ and may again choose $t = 0$ so that constraint (4.6a) collapses to just $r \leqslant \omega$. We can simplify further by choosing $r = \omega$ so that only constraints (4.6c) and (4.6d) remain.

Corollary 4.9. *If we let both* $\alpha \to \infty$ *and* $\beta \to \infty$ *in theorem 4.3, then the system of constraints 4.6 reduces to*

$$\begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & \lambda G & -\lambda G d \\ b(x)^T & (-\lambda G d)^T & \lambda\left(d^T G d - 1\right) + \omega \end{bmatrix} \succeq 0, \quad \lambda \geqslant 0.$$

All the unbounded cases above are implemented in PICOS and can be selected by setting the appropriate parameters to **None**.

4.1.2. Application to piecewise linear optimization

Another application of lemma 4.1 that leads to a finite conic program is found already in the fundamental paper by Delage and Ye [13]. The authors show that when g is

given as a convex piecewise linear function in $\xi^T x$ and when the sample space $\mathcal{S}$ is again either ellipsoidal or unbounded, then minimizing the worst-case expectation of g amounts once more to solving a semidefinite program. While this specification of g may at first seem less powerful than the squared norm formulation discussed above, it has a notable application in portfolio optimization [13, 27] and bears additional potential for approximating smooth convex cost functions. To further boost modeling flexibility, we extend Delage and Ye's result such that g may be given as a finite maximum over biaffine scalar functions in x and ξ. This gives the following result.

Theorem 4.10. *The constraint* (4.1) *for* $\mathcal{D} = \mathcal{D}_{DY}$ *and*

$$g(x, \xi) = \max_{i \in [k]} \left(h_i(x)^T \xi + \eta_i(x) \right)$$

with $h_i(x) \in \mathbb{R}^m$ *and* $\eta_i(x) \in \mathbb{R}$ *both affine in* x *for all* $i \in [k]$, $k \in \mathbb{Z}_{\geqslant 1}$, *and for* $\mathcal{S} = \{Dy + d \mid \|y\| \leqslant 1\}$ *with* $D \in \mathbb{R}^{m \times m}$ *invertible,* $d \in \mathbb{R}^m$ *and* $G := D^{-T} D^{-1}$ *is equivalent to the following system of constraints in auxiliary variables* $Q \in \mathbb{S}^m$, $q \in \mathbb{R}^m$, $\lambda \in \mathbb{R}^k$ *and* $r, t \in \mathbb{R}$*:*

$$\langle \beta \Sigma + \mu\mu^T, Q \rangle + \mu^T q + r + t \leqslant \omega, \tag{4.7a}$$

$$\|\sqrt{\alpha} \Sigma^{\frac{1}{2}} (2Q\mu + q)\| \leqslant t, \tag{4.7b}$$

$$\begin{bmatrix} \lambda_i G + Q & \frac{1}{2}(q - h_i(x)) - \lambda_i G d \\ \left(\frac{1}{2}(q - h_i(x)) - \lambda_i G d\right)^T & \lambda_i \left(d^T G d - 1\right) - \eta_i(x) + r \end{bmatrix} \succeq 0 \quad \forall \quad i \in [k], \tag{4.7c}$$

$$Q \succeq 0, \ \lambda \succeq 0. \tag{4.7d}$$

Proof. We condense the proof as it is similar to that of theorem 4.3. It is sufficient to show that we obtain the finite system of constraints (4.7c) from the infinite system (4.2b). We can write the latter as

$$\begin{aligned} & g(x, \xi) - \xi^T Q \xi - q^T \xi \leqslant r \ \ \forall \ \ \xi \in \mathcal{S} \\ \iff \quad & \max_{i \in [k]} \left(h_i(x)^T \xi + \eta_i(x) \right) \leqslant \xi^T Q \xi + q^T \xi + r \ \ \forall \ \ \xi \in \mathcal{S} \\ \iff \quad & h_i(x)^T \xi + \eta_i(x) \leqslant \xi^T Q \xi + q^T \xi + r \ \ \forall \ \ \xi \in \mathcal{S} \ \ \forall \ \ i \in [k] \\ \iff \quad & \xi^T (-Q) \xi + (h_i(x) - q)^T \xi + \eta_i(x) - r \leqslant 0 \ \ \forall \ \ \xi \in \mathcal{S} \ \ \forall \ \ i \in [k]. \end{aligned}$$

With the equivalence

$$\xi \in \mathcal{S} \quad \iff \quad \xi^T G \xi - 2(Gd)^T \xi + d^T G d - 1 \leqslant 0,$$

system (4.2b) is further equivalent to the condition that the implication

$$\forall \xi \in \mathbb{R}^m : \left(\begin{array}{r} \xi^T G \xi - (2Gd)^T \xi + d^T G d - 1 \leqslant 0 \\ \implies \ \xi^T (-Q) \xi + (h_i(x) - q)^T \xi + \eta_i(x) - r \leqslant 0 \end{array} \right)$$

holds for all $i \in [k]$. By the inhomogeneous S-procedure (lemma 4.4), this can be written as the system of linear matrix inequalities

$$\begin{aligned} &\begin{bmatrix} G & -Gd \\ -(Gd)^T & d^T Gd - 1 \end{bmatrix} \lambda_i - \begin{bmatrix} -Q & \frac{1}{2}(h_i(x) - q) \\ \frac{1}{2}(h_i(x) - q)^T & \eta_i(x) - r \end{bmatrix} \succeq 0 \ \ \forall \ \ i \in [k] \\ \iff \quad &\begin{bmatrix} \lambda_i G + Q & \frac{1}{2}(q - h_i(x)) - \lambda_i Gd \\ \left(\frac{1}{2}(q - h_i(x)) - \lambda_i Gd\right)^T & \lambda_i \left(d^T Gd - 1\right) - \eta_i(x) + r \end{bmatrix} \succeq 0 \ \ \forall \ \ i \in [k] \end{aligned}$$

in auxiliary variables $\lambda_i \geqslant 0$. The theorem follows from lemma 4.1 when we identify the scalar λ_i as entries of a vector variable $\lambda \succeq 0 \in \mathbb{R}^k$. □

Again, we want to allow a PICOS user to omit some of the bounds on the uncertain distribution, which gives rise to the following corollaries.

Unbounded mean. The following holds by analogy to corollary 4.6.

Corollary 4.11. *If we let $\alpha \to \infty$ in theorem 4.10, then the system* (4.7) *reduces to*

$$\begin{gathered} \langle \beta\Sigma - \mu\mu^T, Q \rangle + r \leqslant \omega, \quad Q \succeq 0, \quad \lambda \succeq 0, \\ \begin{bmatrix} \lambda_i G + Q & -\left(Q\mu + \frac{1}{2}h_i(x) + \lambda_i Gd\right) \\ -\left(Q\mu + \frac{1}{2}h_i(x) + \lambda_i Gd\right)^T & \lambda_i \left(d^T Gd - 1\right) - \eta_i(x) + r \end{bmatrix} \succeq 0 \ \ \forall \ \ i \in [k]. \end{gathered}$$

Unbounded covariance. The following holds by analogy to corollary 4.7.

Corollary 4.12. *If we let $\beta \to \infty$ in theorem 4.10, then the system* (4.7) *reduces to*

$$\begin{gathered} \mu^T q + r + t \leqslant \omega, \quad \|\sqrt{\alpha}\Sigma^{\frac{1}{2}} q\| \leqslant t, \quad \lambda \geqslant 0, \\ \begin{bmatrix} \lambda_i G & \frac{1}{2}(q - h_i(x)) - \lambda_i Gd \\ \left(\frac{1}{2}(q - h_i(x)) - \lambda_i Gd\right)^T & \lambda_i \left(d^T Gd - 1\right) - \eta_i(x) + r \end{bmatrix} \succeq 0 \ \ \forall \ \ i \in [k]. \end{gathered}$$

Unbounded support. The case of $\mathcal{S} = \mathbb{R}^m$ is also handled by Delage and Ye [13]. As in our proof of corollary 4.8, we may use lemma 4.5 instead of the S-procedure. Thereby, constraint (4.2b) is now equivalent to

$$\begin{aligned} &\xi^T(-Q)\xi + (h_i(x) - q)^T \xi + \eta_i(x) - r \leqslant 0 \ \ \forall \ \ \xi \in \mathbb{R}^m \ \ \forall \ \ i \in [k] \\ \iff \quad &\begin{bmatrix} Q & \frac{1}{2}(q - h_i(x)) \\ \frac{1}{2}(q - h_i(x))^T & r - \eta_i(x) \end{bmatrix} \succeq 0 \ \ \forall \ \ i \in [k]. \end{aligned}$$

Again, the constraint $Q \succeq 0$ becomes redundant and we arrive at the following corollary.

Corollary 4.13. *The constraint* (4.1) *for $\mathcal{D} = \mathcal{D}_{DY}$ and*

$$g(x, \xi) = \max_{i \in [k]} \left(h_i(x)^T \xi + \eta_i(x)\right)$$

with $h_i(x) \in \mathbb{R}^m$ *and* $\eta_i(x) \in \mathbb{R}$ *both affine in* x *for all* $i \in [k]$, $k \in \mathbb{Z}_{\geqslant 1}$, *and for* $\mathcal{S} = \mathbb{R}^m$ *is equivalent to the following system of constraints in auxiliary variables* $Q \in \mathbb{S}^m$, $q \in \mathbb{R}^m$ *and* $r, t, \in \mathbb{R}$*:*

$$\langle \beta\Sigma + \mu\mu^T, Q\rangle + \mu^T q + r + t \leqslant \omega, \tag{4.8a}$$

$$\|\sqrt{\alpha}\Sigma^{\frac{1}{2}}(2Q\mu + q)\| \leqslant t, \tag{4.8b}$$

$$\begin{bmatrix} Q & \frac{1}{2}(q - h_i(x)) \\ \frac{1}{2}(q - h_i(x))^T & r - \eta_i(x) \end{bmatrix} \succeq 0 \ \ \forall \ \ i \in [k]. \tag{4.8c}$$

Unbounded mean and covariance. The case of $\alpha \to \infty$ and $\beta \to \infty$ describes once more a robust optimization scenario where constraint (4.1) reduces to the family of robust constraints $h_i(x)^T\xi + \eta_i(x) \leqslant \omega \ \ \forall \ \ \xi \in \mathcal{S} \ \ \forall \ \ i \in [k]$. We provide two alternative representations of the robust counterpart in the form of an SDP and an SOCP.

Corollary 4.14. *If we let both* $\alpha \to \infty$ *and* $\beta \to \infty$ *in theorem 4.10, then the system of constraints 4.7 reduces to*

$$\begin{bmatrix} \lambda_i G & -\left(\frac{1}{2}h_i(x) + \lambda_i G d\right) \\ -\left(\frac{1}{2}h_i(x) + \lambda_i G d\right)^T & \lambda_i\left(d^T G d - 1\right) - \eta_i(x) + \omega \end{bmatrix} \succeq 0 \ \ \forall \ \ i \in [k], \quad \lambda \geqslant 0.$$

This is equivalent to the system

$$\left.\begin{aligned} \zeta_i - d^T D^{-T} z_{(i)} + \eta_i(x) &\leqslant \omega, \\ D^{-T} z_{(i)} + h_i(x) &= 0, \\ \|z_{(i)}\| &\leqslant \zeta_i \end{aligned}\right\} \ \forall \ \ i \in [k]$$

in auxiliary variables $z_{(i)} \in \mathbb{R}^m$ *and* $\zeta_i \in \mathbb{R}$ *for all* $i \in [k]$.

Proof. The first system is obtained by analogy to corollary 4.9. The second system stems from an application of theorem 3.2. We omit the proof that constraint (4.1) indeed reduces to the given family of robust linear constraints; this follows from the fact that we lose nothing when we limit Ξ to the set of Dirac delta functions supported on $\mathcal{S}$. To apply the theorem we write

$$\begin{aligned} \xi \in \mathcal{S} \quad &\iff \quad \|D^{-1}(\xi - d)\| \leqslant 1 \quad \iff \quad \begin{pmatrix} 1 \\ D^{-1}(\xi - d) \end{pmatrix} \in \mathcal{Q}_{m+1} \\ &\iff \begin{bmatrix} 0 \\ D^{-1} \end{bmatrix} \xi + \begin{pmatrix} 1 \\ -D^{-1}d \end{pmatrix} \in \mathcal{Q}_{m+1} \end{aligned}$$

and identify the terms as follows:

Theorem 3.2	Corollary 4.14
θ	ξ
y^T	$\begin{bmatrix}\zeta_i & z_{(i)}^T\end{bmatrix}$
P^T	$\begin{bmatrix}0 & D^{-T}\end{bmatrix}$
p^T	$\begin{bmatrix}1 & -d^T D^{-T}\end{bmatrix}$
$Ax + \alpha_\theta$	$h_i(x)$
$a_x^T x + \alpha$	$\eta_i(x) - \omega$

The corollary then follows from the definition of the self-dual cone $\mathcal{Q}_{m+1}$. □

In addition to posing an upper bound on the worst-case expectation of a convex piecewise linear loss function, PICOS users may also pose a lower bound on the worst-case expectation of a concave piecewise linear utility function by virtue of the equivalence

$$\inf_{\Xi\in\mathcal{D}} \mathbb{E}_{\xi\sim\Xi}\left[g(x,\xi)\right] \geqslant \omega \quad\Longleftrightarrow\quad \sup_{\Xi\in\mathcal{D}} \mathbb{E}_{\xi\sim\Xi}\left[-g(x,\xi)\right] \leqslant -\omega.$$

This allows both distributionally robust minimization and maximization problems to be solved for piecewise linear objective functions that are convex and concave in the decision variables, respectively.

4.2. Discrepancy-based ambiguity using the Wasserstein metric

In the previous section we discussed a scenario where a modeler would utilize repeated measurements to make a rough estimate of the first two moments of a data generating distribution. This is reasonable whenever one feels comfortable to quantify the degrees of confidence in the estimated mean and covariance matrix, represented respectively by the parameters α and β of the ambiguity set $\mathcal{D}_{\mathrm{DY}}$. In the absence of historical data, however, this is a tedious task (discussed in length in [13]) and it would seem more convenient to have access to just a single parameter that controls the model's degree of conservatism. Such an interface is provided by discrepancy-based ambiguity sets.

Discrepancy-based DRO is a direct extension of stochastic programming where in addition to an estimated, *nominal* distribution, the modeler specifies the largest distance, measured in a metric on the space of probability distributions, that they believe the "true" distribution to be away from the nominal one. Optimization is then performed with respect to the worst-case expectation over all distributions in the resulting confidence ball. A discrepancy-based ambiguity set has the general form

$$\mathcal{D}_d := \{\Xi \in \mathcal{M} \mid d(\Xi, \Xi_N) \leqslant \epsilon\}$$

where again $\mathcal{M}$ is the set of all Borel probability measures on $\mathbb{R}^m$, $\Xi_N \in \mathcal{M}$ is the nominal probability measure, $\epsilon \in \mathbb{R}_{\geqslant 0}$ defines the radius of the distributional confidence ball and where $d : \mathcal{M}\times\mathcal{M} \to \mathbb{R}_{\geqslant 0}\cup\{\infty\}$ measures the distance, or *discrepancy*, between Ξ and Ξ_N.

A variety of metrics has been studied in the literature; a concise review is provided by Rahimian and Mehrotra [35]. We focus on a popular group of metrics that measure, simply put, the effort required to transform one probability measure to resemble the other. More precisely, we study the class of Wasserstein distances of order $p \in \mathbb{R}_{\geqslant 1} \cup \{\infty\}$ induced by the Euclidean metric space $(\mathbb{R}^m, (x, y) \mapsto \|x - y\|)$, that is defined as

$$W_p(\Xi, \Xi') := \left(\inf_{\Phi \in \Pi(\Xi, \Xi')} \int_{\mathbb{R}^m \times \mathbb{R}^m} \|\phi - \phi'\|^p \; \Phi(d\phi \times d\phi') \right)^{\frac{1}{p}}$$

where $\Pi(\Xi, \Xi')$ denotes the set of all couplings of Ξ and Ξ', that is the set of all Borel probability measures on $\mathbb{R}^m \times \mathbb{R}^m$ with marginals Ξ and Ξ'. Each Wasserstein distance denotes the optimal value of a transport problem: The term $\Phi(d\phi \times d\phi')$ represents the probability mass being moved from the source region $d\phi$ to the target region $d\phi'$ according to the *transport plan* Φ while the term $\|\phi - \phi'\|^p$ quantifies the cost of moving one unit of mass from ϕ to ϕ'. Thereby, the integral denotes the total cost of transforming from Ξ to Ξ' according to Φ while the infimum denotes the cheapest total transformation cost over all possible transport plans.

For $p = 1$ we obtain as a special case the Kantorovich metric W_1 and the associated Kantorovich ambiguity set $\mathcal{D}_{W_1}$. The metric itself is interesting as it provides a bound on the absolute difference between the expected value of a random utility function under an estimated and under the true distribution, respectively: Given that the utility function is Lipschitz continuous with constant L in the uncertainty and assuming that the nominal and the true distribution have a Kantorovich distance of at most ϵ, then, as a result of the Kantorovich-Rubinstein theorem [24], the mentioned difference in expectations is no larger than $L\epsilon$ [37]. Note that this difference is closely related to the *post-decision disappointment*, also known as the *optimizer's curse* or the *optimization bias*, that relates the optimal value of a stochastic program to the actual utility that one can expect from its implementation [44]. The Kantorovich ambiguity set has therefore a probabilistic interpretation of protecting the decision maker from post-decision disappointment. This observation motivates the use of Wasserstein-ambiguity in the context of machine learning where such disappointment is reminiscent of overfitting. Indeed, many regularization techniques used in statistical learning methods can be recovered as special cases of distributionally robust optimization under the Wasserstein metric [10, 41].

In the context of data-driven optimization where structural information about the "true" distribution is often hard to come by, it is common to choose the empirical distribution on the measurements as the nominal one. This is not only convenient in the sense that it allows practitioners to disregard the probabilistic background and use the DRO framework as a "robustness black box", which makes it attractive for tasks that are frequently tackled with heuristic techniques such as outlier detection [12], but it can also be justified from a theoretical point of view as Kuhn et al. [28] argue that no discrete estimator converges faster in the Kantorovich metric to any true distribution with compact support. Since representing a variety of random distributions goes beyond

the scope of PICOS as a conic optimization library, we will limit our study to discrete distributions which contain the empirical distribution as a special case. To this end we define the *training samples* $\xi_{(i)} \in \mathbb{R}^m$ for all $i \in [N]$, $N \in \mathbb{Z}_{\geqslant 1}$, and a vector of *sample weights* $w \in \mathbb{R}^N_{\geqslant 0}$ with $\sum_{i=1}^N w_i = 1$. We denote the resulting discrete distribution by

$$\Xi_D := \sum_{i=1}^{N} w_i \delta_{\xi_{(i)}}$$

where $\delta_{\xi_{(i)}}$ denotes the Dirac delta function with unit mass at $\xi_{(i)}$. Note that the empirical distribution is obtained by setting $w_i = \frac{1}{N}$ for all $i \in [N]$.

4.2.1. Application to least-squares problems

Recall that we are looking for a tractable reformulation of the worst-case expectation constraint (4.1) given as $\sup_{\Xi \in \mathcal{D}} \mathbb{E}_{\xi \sim \Xi}[g(x, \xi)] \leqslant \omega$. We will study again the case of $g(x, \xi) := \|a(x, \xi)\|^2$ where $a(x, \xi) \in \mathbb{R}^k$ is biaffine in x and ξ, which already led to a semidefinite representation in the context of moment ambiguity (theorem 4.3). It turns out that also for Wasserstein-ambiguity of order $p = 2$ we can reformulate constraint (4.1) as a finite system containing a number of linear matrix inequalities. The following theorem is an application of a recent result by Kuhn et al. [28].

Theorem 4.15. *Constraint* (4.1) *for* $\mathcal{D} = \mathcal{D}_{W_2}$ *with* $\Xi_N = \Xi_D$ *a discrete distribution and for* $g(x, \xi) = \|B(x)\xi + b(x)\|^2$ *with* $B(x) \in \mathbb{R}^{k \times m}$ *and* $b(x) \in \mathbb{R}^k$ *both affine in* x *is equivalent to the following system of constraints in auxiliary variables* $\gamma \in \mathbb{R}$, $s \in \mathbb{R}^N$, $U \in \mathbb{S}^m$, $u \in \mathbb{R}^m$ *and* $\mu \in \mathbb{R}$*:*

$$\gamma\epsilon^2 + w^T s \leqslant \omega, \tag{4.9a}$$

$$\gamma \geqslant 0, \tag{4.9b}$$

$$\begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & U & u \\ b(x)^T & u^T & \mu \end{bmatrix} \succeq 0, \tag{4.9c}$$

$$\begin{bmatrix} \gamma I_m - U & \gamma\xi_{(i)} + u \\ (\gamma\xi_{(i)} + u)^T & \gamma\|\xi_{(i)}\|^2 + s_i - \mu \end{bmatrix} \succeq 0 \quad \forall \ i \in [N]. \tag{4.9d}$$

Remark. It is possible to state theorem 4.15 without constraint (4.9c) and without the variables U, u and μ at the cost of increasing the size of the linear matrix inequalities that comprise the family of constraints (4.9d) from $m+1$ to $k+m+1$ diagonal elements.

The first part of the following proof, up to and including the application of lemma 4.5, was omitted from [28] and is found in Nguyen et al. [33] and Zhen et al. [49]. We thank the authors for allowing us early access to the proof details. The second part is specific to our choice of g as a squared norm of a biaffine term. In contrast Kuhn et al. allow possibly indefinite quadratic loss functions and arrive at a semidefinite representation that is not necessarily linear in the decision vector x.

Proof of theorem 4.15. Let us first write the loss function as a quadratic expression, that is

$$g(x,\xi) = \|B(x)\xi + b(x)\|^2 = \xi^T \underbrace{B(x)^T B(x)}_{Q(x)} \xi + 2\underbrace{b(x)^T B(x)}_{q(x)^T}\xi + \underbrace{b(x)^T b(x)}_{r(x)}.$$

For $\mathcal{D} = \mathcal{D}_{W_p}$, Kuhn et al. [28] provide a strong dual for the left-hand side of constraint (4.1) in the form of

$$\sup_{\Xi\in\mathcal{D}} \mathbb{E}_{\xi\sim\Xi}[g(x,\xi)] = \inf_{\gamma\in\mathbb{R}_{\geqslant 0}} \gamma\epsilon^p + \mathbb{E}_{\xi\sim\Xi_N}\left[\sup_{z\in\mathbb{R}^m} g(x,z) - \gamma\|z-\xi\|^p\right] \tag{4.10}$$

where the supremum under the expectation is known as the Moreau-Yosida regularization of $g'(\xi) := g(x,\xi)$. We refer the reader to Esfahani Mohajerin and Kuhn [20] for a derivation of the dual for $p = 1$. For $p = 2$ and $\Xi_N = \Xi_D$, the expectation reduces to a weighted sum and we may introduce an auxiliary variable $s \in \mathbb{R}^N$ to state (4.10) as the optimal value of the constrained problem

$$\begin{array}{ll} \underset{\gamma\in\mathbb{R}_{\geqslant 0},\ s\in\mathbb{R}^N}{\text{minimize}} & \gamma\epsilon^2 + w^T s \\ \text{subject to} & \sup\limits_{z\in\mathbb{R}^m} g(x,z) - \gamma\|z-\xi_{(i)}\|^2 \leqslant s_i \ \ \forall\ i\in[N]. \end{array} \tag{4.11}$$

The constraints of (4.11), for all $i \in [N]$, admit the semidefinite form

$$\begin{aligned}
& \sup_{z\in\mathbb{R}^m} g(x,z) - \gamma\|z-\xi_{(i)}\|^2 \leqslant s_i \\
\iff\quad & \gamma\|z-\xi_{(i)}\|^2 - g(x,z) + s_i \geqslant 0 \ \ \forall\ z\in\mathbb{R}^m \\
\iff\quad & z^T(\gamma I_m - Q(x))z - 2\left(\gamma\xi_{(i)} + q(x)\right)^T z + \gamma\xi_{(i)}^T\xi_{(i)} - r(x) + s_i \geqslant 0 \ \ \forall\ z\in\mathbb{R}^m \\
\iff\quad & \begin{bmatrix} \gamma I_m - Q(x) & -\gamma\xi_{(i)} - q(x) \\ \left(-\gamma\xi_{(i)} - q(x)\right)^T & \gamma\xi_{(i)}^T\xi_{(i)} - r(x) + s_i \end{bmatrix} \succeq 0 \\
\iff\quad & \begin{bmatrix} \gamma I_m & -\gamma\xi_{(i)} \\ -\gamma\xi_{(i)}^T & \gamma\xi_{(i)}^T\xi_{(i)} + s_i \end{bmatrix} - \begin{bmatrix} Q(x) & q(x) \\ q(x)^T & r(x) \end{bmatrix} \succeq 0
\end{aligned}$$

where the second to last equivalence follows from lemma 4.5. In order to arrive at a family of matrix inequalities that are linear in x, we introduce additional auxiliary variables $U \in \mathbb{S}^m$, $u \in \mathbb{R}^m$ and $\mu \in \mathbb{R}$ and the auxiliary constraint

$$\begin{aligned}
& \begin{bmatrix} U & u \\ u^T & \mu \end{bmatrix} - \begin{bmatrix} Q(x) & q(x) \\ q(x)^T & r(x) \end{bmatrix} \succeq 0 \\
\iff\quad & \begin{bmatrix} U & u \\ u^T & \mu \end{bmatrix} - \begin{bmatrix} B(x)^T \\ b(x)^T \end{bmatrix}\begin{bmatrix} B(x) & b(x) \end{bmatrix} \succeq 0 \\
\iff\quad & \begin{bmatrix} I_k & B(x) & b(x) \\ B(x)^T & U & u \\ b(x)^T & u^T & \mu \end{bmatrix} \succeq 0
\end{aligned}$$

where the last equivalence follows from the Schur complement lemma 3.4. Note that this constitutes a linear matrix inequality with respect to x. By the transitivity of the Loewner order, we can now write the i-th constraint of problem (4.11) as

$$\begin{aligned}
&\begin{bmatrix} \gamma I_m & -\gamma\xi_{(i)} \\ -\gamma\xi_{(i)}^T & \gamma\xi_{(i)}^T\xi_{(i)} + s_i \end{bmatrix} - \begin{bmatrix} U & u \\ u^T & \mu \end{bmatrix} \succeq 0 \\
\iff \quad &\begin{bmatrix} \gamma I_m - U & -\gamma\xi_{(i)} - u \\ -\gamma\xi_{(i)}^T - u^T & \gamma\xi_{(i)}^T\xi_{(i)} + s_i - \mu \end{bmatrix} \succeq 0 \\
\iff \quad &\begin{bmatrix} \gamma I_m - U & \gamma\xi_{(i)} + u \\ \big(\gamma\xi_{(i)} + u\big)^T & \gamma\|\xi_{(i)}\|^2 + s_i - \mu \end{bmatrix} \succeq 0.
\end{aligned}$$

The theorem follows by substituting the objective function of problem (4.11) for the left hand side of constraint (4.1). □

4.2.2. Application to piecewise linear optimization

In section 4.1.2 we have seen a finite reformulation of a worst-case expectation constraint concerning a convex piecewise linear loss function under moment ambiguity. In the following we state a similar result also for the Kantorovich ambiguity set centered at a discrete distribution.

Theorem 4.16 (Esfahani Mohajerin and Kuhn [20]). *The worst-case expectation constraint* (4.1) *for* $\mathcal{D} = \mathcal{D}_{W_1}$ *with* $\Xi_N = \Xi_D$ *a discrete distribution and for*

$$g(x, \xi) = \max_{i \in [k]} \big(h_i(x)^T\xi + \eta_i(x)\big)$$

with $h_i(x) \in \mathbb{R}^m$ *and* $\eta_i(x) \in \mathbb{R}$ *both affine in* x *for all* $i \in [k]$, $k \in \mathbb{Z}_{\geqslant 1}$, *is equivalent to the following system of constraints in auxiliary variables* $\gamma \in \mathbb{R}$ *and* $s \in \mathbb{R}^N$*:*

$$\gamma\epsilon + w^T s \leqslant \omega, \tag{4.12a}$$

$$h_i(x)^T\xi_{(j)} + \eta_i(x) \leqslant s_j \quad \forall\ i \in [k]\ \ \forall\ j \in [N], \tag{4.12b}$$

$$\|h_i(x)\| \leqslant \gamma \quad \forall\ i \in [k]. \tag{4.12c}$$

Esfahani Mohajerin and Kuhn [20] originally state the theorem in an extended form where the support of Ξ is constrained to a polytope. Such a restricted Wasserstein ball is not implemented in PICOS yet but may be added in the future on user request. Furthermore, we remark that the norm in constraint (4.12c) is in general the dual norm of the norm used to define the Kantorovich distance W_1, which we have fixed as the self-dual Euclidean norm in our formulation of the Wasserstein ambiguity set. What follows is a short direct proof for this simplified result.

Proof of theorem 4.16. Consider again the dual form (4.10) that lets us write the left hand side of constraint (4.1) as the optimal value of the constrained problem

$$\begin{array}{ll} \underset{\gamma \in \mathbb{R}_{\geqslant 0},\ s \in \mathbb{R}^N}{\text{minimize}} & \gamma\epsilon + w^T s \\ \text{subject to} & \sup\limits_{z \in \mathbb{R}^m} g(x,z) - \gamma\|z - \xi_{(j)}\| \leqslant s_j \ \ \forall\ j \in [N]. \end{array} \tag{4.13}$$

For a fixed $j \in [N]$, we can pose the j-th constraint of (4.13) as

$$\begin{aligned} & \sup_{z \in \mathbb{R}^m} g(x,z) - \gamma\|z - \xi_{(j)}\| \leqslant s_j \\ \iff \quad & \sup_{z \in \mathbb{R}^m} \max_{i \in [k]} \left(h_i(x)^T z + \eta_i(x)\right) - \gamma\|z - \xi_{(j)}\| \leqslant s_j \\ \iff \quad & \sup_{z \in \mathbb{R}^m} h_i(x)^T z + \eta_i(x) - \gamma\|z - \xi_{(j)}\| \leqslant s_j \ \ \forall\ i \in [k] \\ \iff \quad & \sup_{z \in \mathbb{R}^m} h_i(x)^T (z + \xi_{(j)}) + \eta_i(x) - \gamma\|z\| \leqslant s_j \ \ \forall\ i \in [k] \\ \iff \quad & h_i(x)^T \xi_{(j)} + \eta_i(x) + \sup_{z \in \mathbb{R}^m} h_i(x)^T z - \gamma\|z\| \leqslant s_j \ \ \forall\ i \in [k]. \end{aligned}$$

The remaining supremum simplifies to

$$\begin{aligned} \sup_{z \in \mathbb{R}^m} h_i(x)^T z - \gamma\|z\| &= \sup_{z \in \mathbb{R}^m} \frac{\|z\|}{\|h_i(x)\|} h_i(x)^T h_i(x) - \gamma\|z\| \\ &= \sup_{z \in \mathbb{R}^m} \left(\|h_i(x)\| - \gamma\right)\|z\| \\ &= \begin{cases} 0, & \text{if } \|h_i(x)\| \leqslant \gamma, \\ \infty, & \text{otherwise,} \end{cases} \end{aligned}$$

for every $i \in [k]$. Therefore, problem (4.13) is equivalent to the finite problem

$$\begin{array}{ll} \underset{\gamma \in \mathbb{R},\ s \in \mathbb{R}^N}{\text{minimize}} & \gamma\epsilon + w^T s \\ \text{subject to} & h_i(x)^T \xi_{(j)} + \eta_i(x) \leqslant s_j \ \ \forall\ i \in [k] \ \ \forall\ j \in [N], \\ & \|h_i(x)\| \leqslant \gamma \ \ \forall\ i \in [k]. \end{array} \tag{4.14}$$

Note that the condition of $\gamma \geqslant 0$ is now implied by the constraints as $k > 0$. The theorem follows by substituting the objective function of problem (4.14) for the left hand side of constraint (4.1). □

As for the case of moment-ambiguity, the result trivially extends to constraints that pose a lower bound on the worst-case expectation of a concave piecewise linear utility function, defined as the pointwise minimum of a finite number of biaffine functions.

5. User interface

In the following we show how the results of sections 3 and 4 can be accessed within the Python programming language using the PICOS library. In section 5.1 we first give a quick tutorial on how to use PICOS to solve classical constrained optimization problems with certain data. Section 5.2 then presents the new functionality that allows the definition of uncertain data parameterized through safety regions or distributional ambiguity sets. In particular, Section 5.2.4 presents an application of Wasserstein-ambiguous stochastic programming in linear signal estimation. We remind the reader that section 2.4 gives instructions on how to read and reproduce the code listings below.

5.1. An introduction to PICOS

In this section we showcase the workflow of using PICOS. Let us solve the following constrained regression model as an introductory example:

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad \|Ax - b\| \tag{5.1a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i = 1, \tag{5.1b}$$

$$x \succeq 0. \tag{5.1c}$$

The problem asks for a least-norm approximation of the vector b by a convex combination of the columns of A. For illustration purposes we consider this model in a geometric setting where the goal is to find the point within the convex hull of the columns of A that is spatially closest to b. To this end we first create a set of random points using NumPy and plot it together with its convex hull, computed by SciPy:

```
>>> n = 30
>>> points = numpy.random.rand(n, 2)
>>> hull = scipy.spatial.ConvexHull(points)
>>> V = hull.vertices
>>> _ = pyplot.fill(points[V, 0], points[V, 1], "lightgray")
>>> for s in hull.simplices:
...     _ = pyplot.plot(points[s, 0], points[s, 1], "k-")
>>> _ = pyplot.plot(points[:, 0], points[:, 1], "k.")
>>> pyplot.show()
```

Up to additional formatting, this will show figure 1. Note that the random points are stored as the rows of the 2-dimensional NumPy array `points`. To obtain A from this array, we need to load its transpose `points.T` with PICOS. This is done using `Constant`:

```
>>> A = picos.Constant("A", points.T); A
<2×30 Real Constant: A>
```

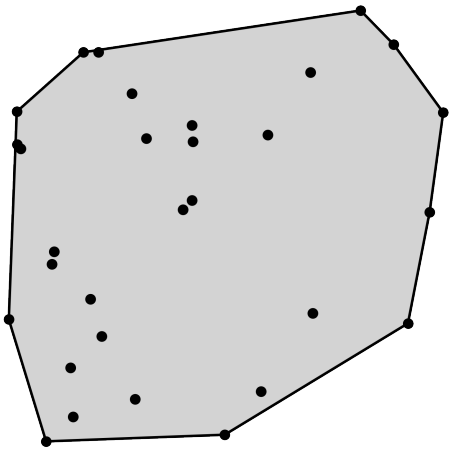

Figure 1: The convex hull (gray area with black outline) of a set of randomly generated points.

The object `A` is now a constant PICOS expression representing the matrix A. We can print its first six columns representing the first six random points as follows:

```
>>> print(A[:, :6])
[ 4.17e-01  1.14e-04  1.47e-01  1.86e-01  3.97e-01  4.19e-01]
[ 7.20e-01  3.02e-01  9.23e-02  3.46e-01  5.39e-01  6.85e-01]
```

Let us now define the target point `b`:

```
>>> b = picos.Constant("b", [1, 0])
```

The list `[1, 0]` is understood as a column vector by default. Next we define the decision variable `x` representing the coefficients of a convex combination of the points. This is done by instantiating the `RealVariable` class with the variable's name as the first and shape as the second argument:

```
>>> x = picos.RealVariable("x", n); x
<30×1 Real Variable: x>
```

We can now define the objective (5.1a) using the Python operators `*`, `-` and ***`abs`***:

```
>>> objective = abs(A*x - b); objective
<Euclidean Norm: ‖A·x - b‖>
```

Now let us define the remainder of problem (5.1) by first creating an instance `P` of the `Problem` class:

```
>>> P = picos.Problem()
```

We can assign our objective function and the search direction to `P` as follows:

```
>>> P.set_objective("min", objective)
```

Constraints are added via the `add_constraint` method of `P`. For the equality constraint (5.1b) we express the summation over the elements of x using the function `picos.sum` which behaves much like Python's built-in function ***`sum`*** but assigns a meaningful string description to the resulting expression. The constraint is then formed using the overloaded `==` operator:

```
>>> P.add_constraint(picos.sum(x) == 1)
<1×1 Affine Constraint: ∑(x) = 1>
```

Note that the constraint is returned for further reference. The elementwise inequality constraint (5.1c) is expressed using the `>=` operator:

```
>>> P.add_constraint(x >= 0)
<30×1 Affine Constraint: x ≥ 0>
```

Here the right hand side `0` is automatically *broadcasted* to a zero vector of same shape as `x`. Let us print the complete problem:

```
>>> print(P)
Optimization Problem
  minimize ‖A·x - b‖
  over
    30×1 real variable x
  subject to
    ∑(x) = 1
    x ≥ 0
```

To hand the problem to an available solver for solution, we call `solve`:

```
>>> P.solve()
<primal feasible solution pair (claimed optimal) from cvxopt>
```

Given that the solver finds an optimal solution, this returns a `Solution` object and, by default, also applies the primal solution to our variable `x`. Both `P` and `x` are now *valued*. We can obtain the optimal value representing the Euclidean distance between b and the convex hull of the columns of A as follows:

```
>>> round(P, 4)  # Short for round(P.value, 4).
0.3
```

Since the distance is nonzero, the point closes to b, $p := Ax$, is on the boundary of the convex hull of our points. We can retrieve it as the value of `A*x`:

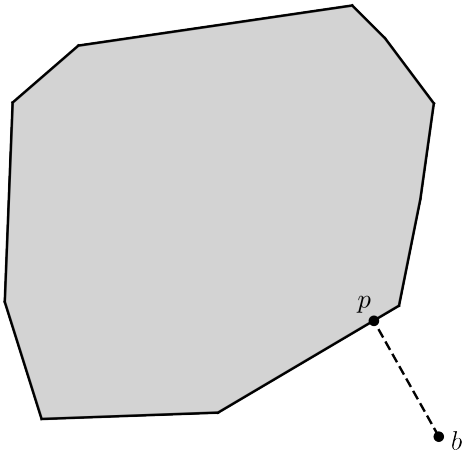

Figure 2: The point p in the convex hull of the columns of A that is closest to b.

```
>>> p = A*x
>>> print(p)
[ 8.50e-01]
[ 2.60e-01]
>>> p.value
<2x1 matrix, tc='d'>
```

While scalars are returned as Python built-in types, vector and matrix values are returned as CVXOPT dense or sparse matrices. Last, let us draw p and a line segment towards b:

```
>>> segment = (b & p).T.value  # Matrix with b in 1st, p in 2nd row.
>>> _ = pyplot.fill(points[V, 0], points[V, 1], "lightgray")
>>> for s in hull.simplices:
...     _ = pyplot.plot(points[s, 0], points[s, 1], "k-")
>>> _ = pyplot.plot(segment[:, 0], segment[:, 1], "k.--")
>>> _ = pyplot.annotate("$b$", (b + [ 0.03, -0.03]).value)
>>> _ = pyplot.annotate("$p$", (p + [-0.04,  0.03]).value)
>>> pyplot.show()
```

The listing shows two more features of PICOS. First, data types that can be loaded by `Constant` can also be passed directly as arguments to algebraic operations involving PICOS expressions. We used this for the location of the annotation labels. Second, expressions can be concatenated vertically with `&` (horizontally with `//`) and their transpose is denoted by the `T` property. The result can be seen in figure 2.

5.2. Modeling uncertain data

In the previous section we have seen the creation of decision variables and constant coefficient matrices through `picos.RealVariable` and `picos.Constant`, respectively. Both variables and constants are represented internally as `AffineExpression` instances which are the building blocks to derive more complex expressions such as norms. In this section we will create and handle instances of the new `UncertainAffineExpression` class which enhances `AffineExpression` with an explicit representation for data uncertainty. More precisely, instances of `UncertainAffineExpression` represent expressions that are biaffine in any number of decision variables and in a single *perturbation parameter* that describes the data uncertainty. In the case of robust optimization, this parameter belongs to a perturbation set while in the case of distributionally robust optimization, the parameter is a random variable whose distribution lives in an ambiguity set of distributions. In PICOS, these sets are jointly referred to as *perturbation universes* and are represented by classes in the `picos.uncertain` namespace:

Perturbation universe	Model of uncertainty
`ScenarioPerturbationSet`	Section 3.1
`ConicPerturbationSet`, `UnitBallPerturbationSet`	Sections 3.2 and 3.3
`MomentAmbiguitySet`	Section 4.1
`WassersteinAmbiguitySet`	Section 4.2

In the following we will discuss these classes in detail. For every class we will try to highlight a different set of features and utilities that we have added to PICOS.

5.2.1. Scenario uncertainty

Recall that a scenario perturbation set is the convex hull of a number of observed or anticipated realizations of the uncertain data, the *scenarios*. In other words, it allows us to give data safety regions as polytopes with reasonably few vertices. For the sake of an example let us produce a list `S` that contains 50 2×2 scenario matrices drawn from a uniform distribution:

```
>>> S = list(numpy.random.random((50, 2, 2)))
```

To create a `ScenarioPerturbationSet`, we provide it with a name for the perturbation parameter, here `"A"`, and our list of scenarios:

```
>>> U = picos.uncertain.ScenarioPerturbationSet("A", S); U
<2×2 Scenario Perturbation Set: conv({37 2×2 scenarios})>
```

Note that by default all scenarios in the interior of the convex hull are sorted out so that we end up with only 37 effective scenarios out of the 50 that we provided.[4]

Every perturbation universe defines a parameter that we can use to denote uncertain expressions. It can be accessed through the `parameter` attribute:

```
>>> A = U.parameter; A
<2×2 Perturbation: A>
```

This object bears similarity with a decision variable created via `picos.RealVariable` but it is never passed to an optimization solver and will never be assigned a value through PICOS. On its own, the parameter `A` merely represents an unknown value contained in the perturbation set `U`. In an optimization context, however, we will see that `A` represents a value under adversarial control. To get there, we also need a decision variable:

```
>>> x = picos.RealVariable("x", 2)
```

We may now treat `A` like a regular coefficient matrix and multiply it with `x`. The result is, like `A` itself, an uncertain affine expression:

```
>>> A*x
<2×1 Uncertain Affine Expression: A·x>
```

Since `A` has no definite value, neither does `A*x` even if `x` is assigned a value:

```
>>> x.value = [1, 1]
>>> (A*x).valued
False
```

However, for the simple case of scenario uncertainty, we can compute both minimal and maximal values of scalar expressions involving `A` and any number of valued variables:

```
>>> sum_A = picos.sum(A*x)
>>> round(sum_A.worst_case_value("min"), 4)
0.3206
>>> round(sum_A.worst_case_value("max"), 4)
3.347
```

Internally, this solves a problem similar to the one that we have solved as an introductory example in section 5.1: PICOS looks for a convex combination of the scenarios of `U` that minimizes or maximizes a function of `A`, which in the example above is just the sum over the elements of `A` as `x` is valued as a vector of all ones.[5] The `worst_case_value` method is used internally to verify worst-case feasibility of constraints or to compute the worst-case value of an objective function under a given decision. This means that we usually do not need to call it directly. Let us now define a robust optimization model:

[4] For anything but scalar scenarios, this feature requires SciPy to be installed.

[5] If the target function is not affine but convex (concave) in the uncertainty and its maximum (minimum) is anticipated, then the problem is solved by scenario enumeration instead.

```
>>> P = picos.Problem()
>>> P.set_objective("min", abs(A*x - 1))
>>> print(P.objective)
minimize max_A ‖A·x - [1]‖
```

Note that in a minimization context, the uncertainty in our objective function is resolved by maximizing over the perturbation parameter `A`. Since we have already assigned a value to `x`, we can now retrieve that decision's worst-case objective value:

```
>>> round(P, 4)
1.1984
```

If we replace `A` with an arbitrary scenario, we can see that the norm that we aim to minimize can only get smaller than that worst case:

```
>>> round(abs(S[0]*x - 1), 4)
0.7109
```

Finally, we can solve `P` to obtain a robust optimal solution to the least-norm problem:

```
>>> _ = P.solve()
>>> print(x)
[ 1.26e-01]
[ 1.69e+00]
>>> round(P, 4)
1.1154
```

Unfortunately, the object-oriented backend of PICOS makes it infeasible to allow scenario robust optimization with all supported expression types as an uncertain version of each class would need to be implemented. We have thus limited the support to expressions and constraints that have an immediate conic representation such as bounds on (squared) Euclidean and Frobenius norms or linear matrix inequalities. Constraints that are equivalent to more than one conic inequality cannot be defined with scenario uncertainty at this point.

5.2.2. Conically bounded uncertainty

While scenario perturbation sets describe polytopes through their vertices, conic perturbation sets can represent polytopes given by their faces as well as any other convex set that is bounded by a finite number of conic inequalities. In PICOS there are two classes that represent such a set: The base class `ConicPerturbationSet` describes the general case and is instantiated with a number of user supplied bound constraints while its subclass `UnitBallPerturbationSet` is limited to the Euclidean or Frobenius unit ball and can be used to represent arbitrary elliptic perturbation through scaling and

translation of the perturbation parameter. For the general purpose class, there are two ways to instantiate it. For the first way, we start with a blank instance of the class which takes the parameter's name and shape as its only two arguments:

```
>>> U = picos.uncertain.ConicPerturbationSet("u", 2)
```

As for constants and variables, the 2 denotes a two-dimensional column vector. We can now add any number of bounds to this unfinished set. To do so, we retrieve a temporary parameter from the `element` property:

```
>>> u = U.element; u
<2×1 Real Variable: u>
```

Note that `u` is not yet a `Perturbation` but a regular `RealVariable`. This allows us to define arbitrary constraints on `u` and pass them to the `bound` method:

```
>>> U.bound(abs(u) <= 1)
>>> U.bound(u[0] + u[1] <= 1)
```

The set `U` now denotes the intersection of the feasible region of both constraints. Before we can use it, we first need to compile it:

```
>>> u = U.compile(); u
<2×1 Perturbation: u>
```

This also gives us the actual perturbation parameter `u`, saving us a call to `parameter`. It is advised to reuse the Python variable used for the temporary parameter as that has done its job and should not be used further to avoid confusion. Compilation uses PICOS' reformulation framework to pose the constraints in conic form (see section 6.1). In fact, we could now obtain our two constraints in the form $Au + Bv + c \in K$ where v represents auxiliary variables introduced with the conic reformulation, if any:

```
>>> U.A, U.B, U.c
(<4×2 Real Constant: A>, None, <4×1 Real Constant: c>)
>>> U.K
<4-dim. Product Cone: ∏(Ci : i ∈ [2])>
>>> U.K.cones[0]
<3-dim. Second Order Cone: {[t; x] : ‖x‖ ≤ t}>
>>> U.K.cones[1]
<1-dim. Nonnegative Orthant: {x ≥ 0}>
```

As for the case of scenario uncertainty, we can now use the parameter `u` to define suitable objective functions and constraints. Since `U` is not ellipsoidal, we are limited to affine expressions (see section 3.2). Let us define an uncertain affine constraint:

```
>>> a = picos.Constant("a", (1, 1))
>>> x = picos.RealVariable("x", 2)
>>> b = (a + u).T*x
>>> C = b <= 1; C
<1×1 Conically Uncertain Affine Constraint: (a + u)ᵀ·x - 1 ≤ 0 ∀ u>
```

If we now look to maximize `a.T*x` under the constraint `C`, then instead of the nominal optimal value of 1 that we would get for letting $u = 0$ in the bounded expression `b`, we can expect a robust optimal value that is smaller than 1:

```
>>> round(picos.maximize(a.T*x, [C]), 4)
0.6667
>>> print(x)
[ 3.33e-01]
[ 3.33e-01]
```

Here `picos.maximize` is just a convenience function that sets up a problem with the given objective function and list of constraints, solves it and returns its optimum value.

We might also be interested in a realization of `u` that attains the worst case in `C` for the current value of `x`. This can be obtained from the `worst_case` method of the constraint's left hand side expression `b`:

```
>>> value, realization = b.worst_case("max")
>>> round(value, 4)  # Verify that the constraint is binding.
1.0
>>> print(realization)
[ 5.00e-01]
[ 5.00e-01]
```

Note that the method `worst_case_value` that we have seen earlier is just a shorthand that gives us the first element of the pair returned by `worst_case`.

The second way to create a `ConicPerturbationSet` skips the creation of a temporary variable and lets the user use an existing variable of matching shape to define the bound constraints. We use the variable `x` to define an ellipsoidal perturbation set:

```
>>> A = picos.Constant("A", [[1.0, 0.0],
...                          [0.0, 0.5]])
>>> c = picos.Constant("c", (2, 2))
>>> E = picos.uncertain.ConicPerturbationSet.from_constraints(
...     "v", abs(A*(x - c)) <= 1); E
<2×1 Conic Perturbation Set: {v : ‖A·(v - c)‖ ≤ 1}>
>>> v = E.parameter
```

With this set and the parameter `v` we can now make use of theorem 3.3 to solve an uncertain second order conic program. This is made possible by the `unit_ball_form` property of `ConicPerturbationSet`:

```
>>> B, M = E.unit_ball_form
>>> B
<2×1 Unit Ball Perturbation Set: {v' : ‖v'‖ ≤ 1}>
>>> M
{<2×1 Perturbation: v>: <2×1 Uncertain Affine Expression: v(v')>}
```

This property returns an instance of `UnitBallPerturbationSet` representing the two-dimensional Euclidean unit ball $B = \{\nu' \mid \|\nu'\| \leqslant 1\}$ and a dictionary that maps the original perturbation parameter ν to an affine expression $f(\nu')$ of the new parameter ν' such that $\nu' \in B \iff f(\nu') \in E$ holds for all $\nu' \in \mathbb{R}^2$. In other words, substituting `M[v]` for `v` within an uncertain affine expression normalizes the perturbation set from an ellipsoid to a ball. PICOS performs this normalization automatically whenever necessary, so that no call to `unit_ball_form` is needed in practice. This leaves the choice between `ConicPerturbationSet` and `UnitBallPerturbationSet` for representing ellipsoidal uncertainty up to user preference.

As a toy application of theorem 3.3, we'll compute the point that is half way between the origin and the point in the uncertainty set that is furthest away from the origin:

```
>>> P = picos.Problem()
>>> P.set_objective("min", picos.max([abs(x - 0), abs(x - v)]))
>>> print(P.objective)
minimize max(‖x‖, max_v' ‖x - v(v')‖)
>>> _ = P.solve()
```

We can see from the objective's string description that the normalization discussed above has been performed. As we defined it, the perturbation set `E` represents an ellipsoid centered at $c = \begin{bmatrix}2 & 2\end{bmatrix}^T$. A worst-case realization of the uncertainty will be further away from the origin than this center point so that we can expect a robust optimal decision that is elementwise larger than 1. Furthermore, the uncertainty is larger along the second axis due to the definition of the matrix `A`. This means that we can expect the second element of the decision vector to be larger than the first. Let us compare these claims with the numeric solution:

```
>>> print(x)
[ 1.14e+00]
[ 1.96e+00]
```

Care must be taken when we want to obtain the robust optimal value. So far we have accessed the property `P.value` for this purpose but now this will give us an exception:

```
>>> try:
...     P.value
... except picos.uncertain.IntractableWorstCase as error:
...     print(textwrap.fill(str(error)))
```

```
PICOS refuses to compute max(‖[x] - v(v')‖) for v' ∈ {v' : ‖v'‖ ≤ 1}
as this is a nonconvex problem.
```

The problem here is that the objective value is always evaluated for the current value of the decision vector (denoted by `[x]` in the error message), which may be changed by the user at any point. Evaluating the worst-case objective value of a given decision is indeed a nonconvex problem and solving it through a convex reformulation would essentially repeat the solution process that just concluded. We can solve this dilemma in two ways: One is to introduce an epigraph variable that will take the optimal value during solution search and can be queried just like the decision vector. The other approach is to access the optimal value as reported by the solver. We can do this as follows:

```
>>> round(P.last_solution.reported_value, 4)
2.2674
```

5.2.3. Moment ambiguity

We will showcase and test the moment-robust stochastic programming implementation by comparing it against an implementation of an approximate model obtained from a discretization of the uncertainty. We base this experiment on the observation that for the special case of scalar random perturbation supported on $N \in \mathbb{Z}_{\geqslant 1}$ fixed points, the set of probability vectors that characterize the distributions in the moment ambiguity set describes a polytope in $\mathbb{R}^N$. To see this, let $s \in \mathbb{R}^N$ define a one-dimensional sample space $\mathcal{S}_s := \{s_i \mid i \in [N]\}$ with cardinality $|\mathcal{S}_s| = N$ and let, for Δ^{N-1} the probability simplex on $\mathbb{R}^N$, $m_s : \Delta^{N-1} \to \mathcal{M}$ with $m_s(p) := \sum_{i=1}^N p_i \delta_{s_i}$ be a function that maps probability vectors to Borel probability measures supported on $\mathcal{S}_s$. Then, we can write the ambiguity set $\mathcal{D}_{DY}$ for $m = 1$ and $\mathcal{S} = \mathcal{S}_s$ equivalently as

$$
\begin{aligned}
& \left\{ \Xi \in \mathcal{M} \;\middle|\; \begin{array}{r} \mathbb{P}(\xi \in \mathcal{S}) = 1, \\ (\mathbb{E}[\xi] - \mu)^T \Sigma^{-1} (\mathbb{E}[\xi] - \mu) \leqslant \alpha, \\ \mathbb{E}\left[(\xi - \mu)(\xi - \mu)^T\right] \preceq \beta\Sigma \end{array} \right\} \\
= \quad & \left\{ \Xi \in \mathcal{M} \;\middle|\; \mathbb{P}(\xi \in \mathcal{S}_s) = 1,\ (\mathbb{E}[\xi] - \mu)^2 \leqslant \alpha\Sigma,\ \mathbb{E}\left[(\xi - \mu)^2\right] \leqslant \beta\Sigma \right\} \\
= \quad & \left\{ m_s(p) \;\middle|\; p \in \Delta^{N-1},\ \left(\sum_{i=1}^N p_i s_i - \mu \right)^2 \leqslant \alpha\Sigma,\ \sum_{i=1}^N p_i (s_i - \mu)^2 \leqslant \beta\Sigma \right\} \\
= \quad & \left\{ m_s(p) \;\middle|\; p \succeq 0,\ p \preceq 1,\ \mathbb{1}^T p = 1,\ q^T p \leqslant \beta\Sigma,\ |s^T p - \mu| \leqslant \sqrt{\alpha\Sigma} \right\}
\end{aligned}
$$

where $\mathbb{1}$ denotes the vector of all ones and where $q \in \mathbb{R}^N$ is given by $q_i = (s_i - \mu)^2$ for all $i \in [N]$. Note that $P_s := \left\{ p \in \mathbb{R}^N \mid m_s(p) \in \mathcal{D}_{DY} \right\}$ with $\mathcal{D}_{DY}$ parameterized as above is a polytope in $\mathbb{R}^N$ as claimed.

Let us next define a moment-robust problem with scalar uncertainty and an approximation thereof where the uncertainty is supported on just N points. For the exact problem we choose $a, b \in \mathbb{R}$ with $a < b$ and we parameterize $\mathcal{D}_{DY}$ with $m = 1$, $\mathcal{S} = [a, b]$ and fixed scalars $\alpha \geqslant 0$, $\beta \geqslant 1$, $\Sigma > 0$ and $\mu \in \left\{a + \frac{i-1}{N-1}(b-a) \mid i \in [N]\right\}$. Notice that in particular the nominal covariance matrix Σ is scalar. The choice of μ along a grid guarantees that the approximate ambiguity set defined below will be nonempty for any $\Sigma > 0$. We define the exact moment-robust problem as

$$\underset{x \,\in\, \mathbb{R}}{\text{minimize}} \quad \sup_{\Xi \in \mathcal{D}_{DY}} \mathbb{E}_{\xi \sim \Xi} \left[\max_{i \in [k]} C_{i,:} \begin{bmatrix} \xi x & x & \xi & 1 \end{bmatrix}^T \right]$$

where $C \in \mathbb{R}^{k \times 4}$ is a coefficient matrix for $k \in \mathbb{Z}$ with $k \geqslant 2$ so that $C_{i,:}$ is a four-dimensional coefficient column vector for every $i \in [k]$. The loss function given by the pointwise maximum can therefor be understood as an arbitrary random convex piecewise linear function with up to k segments (or rays, lines) whose uncertain coefficients are determined through C and Ξ. For the approximate problem we change only the sample space used to parameterize $\mathcal{D}_{DY}$ from the interval $[a, b]$ to the discrete set $\mathcal{S}_s$ defined through $s_i = a + \frac{i-1}{N-1}(b-a)$ for all $i \in [N]$. This gives us the robust problem

$$\underset{x \,\in\, \mathbb{R}}{\text{minimize}} \quad \sup_{p \in P_s} \sum_{j=1}^{N} p_j \max_{i \in [k]} C_{i,:} \begin{bmatrix} s_j x & x & s_j & 1 \end{bmatrix}^T$$

which, due to $p \succeq 0$ for all $p \in P$, is equivalent to the robust linear problem

$$\begin{aligned} \underset{x \,\in\, \mathbb{R},\ y \,\in\, \mathbb{R}^N}{\text{minimize}} \quad & \sup_{p \in P_s} p^T y \\ \text{subject to} \quad & C_{i,:} \begin{bmatrix} s_j x & x & s_j & 1 \end{bmatrix}^T \leqslant y_j \ \ \forall\ (i,j) \in [k] \times [N] \end{aligned}$$

that we can solve by representing the polytope P_s as a `ConicPerturbationSet`.

We can now start with the implementation. Let us first define the data common to both problems:

```
>>> k = 5
>>> a, b = -1.0, 1.0
>>> mu = a + (3.0 / 4.0)*(b - a)  # N will be a multiple of five.
>>> Sigma = min(abs(mu - a), abs(mu - b))**2 / 4.0
>>> alpha, beta = 0.1, 1.1
>>> C = picos.Constant("C", numpy.random.normal(size=(k, 4)))
```

Next, we define the exact problem using the `MomentAmbiguitySet` class. Given that one-dimensional perturbation isn't a common case, this class expects its `sample_space` parameter as an `piocs.Ellipsoid` instance, so we create one first:

```
>>> r = (b - a) / 2.0  # The radius of the ellipsoid [a, b].
>>> S = picos.Ellipsoid(n=1, A=r, c=(a + r)); S
<Centered Unit Ball: {x : ‖x‖ ≤ 1}>
```

Not surprisingly, our interval $[a, b]$ is recognized as the one-dimensional Euclidean unit ball. We now have all that we need to create a moment ambiguity set:

```
>>> D = picos.uncertain.MomentAmbiguitySet(
...     "ξ", (1, 1), mu, Sigma, alpha, beta, S)
>>> print(D)
MAS(mu=0.5, cov=0.0625, a=0.1, b=1.1, S={x : ‖x‖ ≤ 1})
>>> xi = D.parameter
```

As we can see, the set takes as its first argument the name of the random perturbation parameter, followed by the parameter's shape and the values of μ, Σ, α, β and $\mathcal{S}$. The exact version of the distributionally robust problem can now be defined as follows:

```
>>> P_dro = picos.Problem()
>>> x_dro = picos.RealVariable("x")
>>> t = xi*x_dro // x_dro // xi // 1
>>> f_dro = picos.max([C[i, :]*t for i in range(k)])
>>> P_dro.set_objective("min", f_dro)
>>> print(P_dro.objective)
minimize max_pd(ξ) E_ξ(max(C[i,:]·[ξ·x; x; ξ; 1] : i ∈ [0…4]))
```

From the objective's string representation we can see that the uncertainty is resolved by maximizing over all permissible probability distributions of the parameter ξ, the expectation of the uncertain objective function `f_dro`. In other words, uncertain expressions parameterized by a distributionally ambiguous random perturbation implicitly turn into worst-case expectations when used in an optimization context. Let us now solve this distributionally robust problem to optimality and inspect the solution:

```
>>> S_dro = P_dro.solve(); S_dro
<feasible primal solution (claimed optimal) from cvxopt>
>>> round(S_dro.reported_value, 4)
-0.6122
```

Note that there was no prior guarantee that the problem was bounded as this depends on the uncertain slopes of the lines under the expected maximum. For instances where the coefficients in C do not allow the adversary in control of Ξ to render the objective function bounded from below, PICOS would have raised a `SolutionFailure` exception from `solve`. It is also worth mentioning that for DRO models, PICOS will never compute a worst-case expected value explicitly, so that we once more access the optimal value that was reported by the solver (using the `Solution` object returned by `solve`).

Before we turn to an approximation, we take a quick look at the nominal problem obtained by fixing the random variable ξ at its nominal mean μ:

```
>>> P_mu = picos.Problem()
>>> x_mu = picos.RealVariable("x")
>>> t_mu = mu*x_mu // x_mu // mu // 1
>>> f_mu = picos.max([C[i, :]*t_mu for i in range(k)])
>>> P_mu.set_objective("min", f_mu)
>>> S_mu = P_mu.solve()
>>> round(S_mu.reported_value, 4)
-1.003
```

This problem turns out to be bounded as well and gives an improved objective value, as expected. We will later plot the nominal objective function against a worst-case version that we can compute explicitly from the approximate DRO model. Let us now define the latter, starting with the conic perturbation set representing the set P_s. To observe convergence, we define a number of functions that allow us to obtain approximations for varying N. First for the vector s that defines the discretized sample space $\mathcal{S}_s$:

```
>>> def make_s(N):
...     return picos.Constant("s", numpy.linspace(a, b, N))
```

Next for the auxiliary vector q:

```
>>> def make_q(N, s):
...     return picos.Constant("q", (s - mu)^(s - mu))
```

Here `^` denotes the elementwise Hadamard product. Together s and q let us define an uncertain probability vector $p \in P_s$ as the parameter of a `ConicPerturbationSet`:

```
>>> def make_p(N, s):
...     q = make_q(N, s)
...     P = picos.uncertain.ConicPerturbationSet("p", N)
...     p = P.element
...     P.bound(p >= 0)
...     P.bound(p <= 1)
...     P.bound(picos.sum(p) == 1)
...     P.bound(q.T*p <= beta*Sigma)
...     P.bound(abs(s.T*p - mu) <= (alpha*Sigma)**0.5)
...     return P.compile()
```

Finally, we can use this to define an N-discretized approximation of `P_dro`:

```
>>> def make_approximation(N):
...     s = make_s(N)
```

```
...     p = make_p(N, s)
...     P = picos.Problem()
...     x = picos.RealVariable("x")
...     t = s*x // x // xi // 1
...     y = picos.RealVariable("y", N)
...     P.set_objective("min", p.T*y)
...     P.add_list_of_constraints([
...         C[i, :] * (s[j]*x // x // s[j] // 1) <= y[j]
...         for i in range(k) for j in range(N)])
...     return P
```

For a start, let us set $\mathrm{N} = 10$:

```
>>> P_10 = make_approximation(10)
>>> x_10 = P_10.get_variable("x")
>>> S_10 = P_10.solve()
>>> round(S_10.reported_value, 4)
-0.621
```

Compared to the optimal value of the exact model of around -0.6122, we can see that the approximate model is more optimistic. This is again not surprising as the ambiguity set used by the approximation is a strict subset of the original moment ambiguity set. Let us confirm that the approximation becomes better as we increase N:

```
>>> P_20 = make_approximation(20)
>>> x_20 = P_20.get_variable("x")
>>> S_20 = P_20.solve()
>>> round(S_20.reported_value, 4)
-0.6142
```

In the remainder we use the approximate model for $\mathrm{N} = 20$ for illustration purposes. Since the model has a linear objective subject to convex uncertainty, we can let PICOS compute a distribution that attains the worst-case risk for the robust optimal solution:

```
>>> _, p = P_20.objective.function.worst_case("max")
>>> p = numpy.array(p)  # Convert p to NumPy array.
```

The first observation that we can make is that this worst-case distribution is supported sparsely on just four points:

```
>>> support = abs(p) > 1e-6
>>> s = numpy.array(make_s(20).value)  # Load s as a NumPy array.
>>> print(",\n".join("P(ξ = {:6.4f}) = {:6.4f}".format(x, y)
...     for x, y in zip(s[support], p[support])) + ".")
```

```
P(ξ = 0.2632) = 0.1744,
P(ξ = 0.3684) = 0.3202,
P(ξ = 0.7895) = 0.3517,
P(ξ = 0.8947) = 0.1537.
```

This gives us also a finite representation of the associated risk function:

```
>>> J = support.nonzero()[0]  # Indices of possible outcomes.
>>> C = numpy.array(C.value)  # Turn C into a NumPy array.
>>> def f_worst(x):
...     return sum(
...         p[j] * numpy.max(numpy.hstack([
...             C[i, :] @ numpy.hstack([s[j]*x, x, s[j], 1])
...             for i in range(k)]))
...         for j in J)
>>> f_worst = numpy.vectorize(f_worst)
```

For comparison, let's compute also the nominal objective loss function:

```
>>> def f_mu(x):
...     return numpy.max(numpy.hstack([
...         C[i, :] @ numpy.hstack([mu*x, x, mu, 1])
...         for i in range(k)]))
>>> f_mu = numpy.vectorize(f_mu)
```

Lastly, let us plot both:

```
>>> x = numpy.linspace(x_20.value - 3, x_20.value + 3, 1000)
>>> _ = pyplot.plot(x_mu.value, S_mu.reported_value, "k.",
...                 label="nominal optimal solution")
>>> _ = pyplot.plot(x, f_mu(x), "k--", label="nominal loss function")
>>> _ = pyplot.plot(x_20.value, S_20.reported_value, "k*",
...                 label="robust optimal solution \( x^* \)")
>>> _ = pyplot.plot(x, f_worst(x), "k-",
...                 label="worst-case risk for \( x^* \)")
>>> _ = pyplot.legend()
>>> pyplot.show()
```

Up to additional formatting, this will show figure 3. Note that the nominal optimal solution does indeed minimize the nominal loss function but the robust optimal solution does not minimize its associated worst-case risk. This is no contradiction as a solution that minimizes this particular risk function (that is $x \approx 2$) could still perform worse than the robust optimal solution under a different realization of the ambiguous distribution.

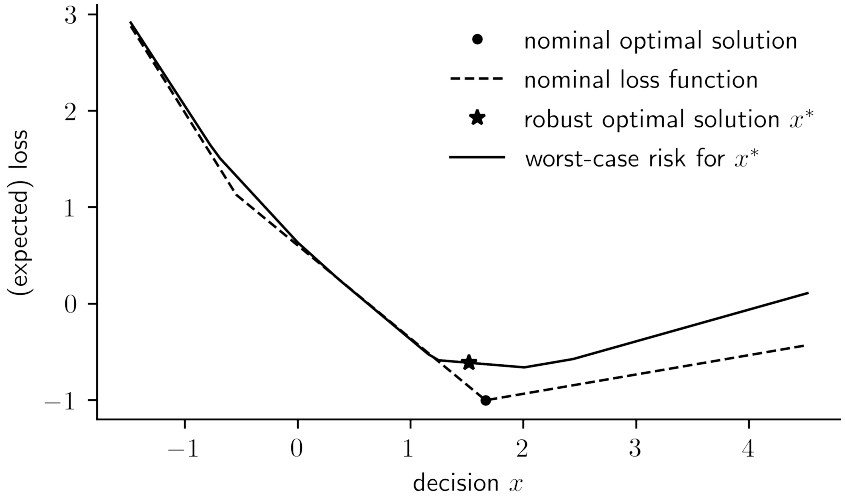

Figure 3: Comparison of a moment-robust stochastic programming solution with a solution to a nominal problem obtained by replacing the random parameter with its nominal mean.

5.2.4. Wasserstein ambiguity: Linear signal estimation[6]

In this section we develop a hypothetical application in linear signal estimation where Wasserstein-robust stochastic programming can be shown to outperform L_2-regularized mean squared error minimization. While vaguely inspired by real world characteristics of wireless communication channels, the focus of our model is to be exemplary as opposed to physically accurate. Portraying both smooth and sharp features, the signals that we want to recover are samples of a sine wave overlayed with a square wave:

```
>>> n = 16  # Signal dimension.
>>> t = numpy.linspace(0, 2*numpy.pi, n)
>>> def make_signal():
...     a, b = 2*numpy.pi*numpy.random.random(2)
...     smooth_part = numpy.sin(t + a)
...     sharp_part = scipy.signal.square(t + b)
...     return 0.4*smooth_part + 0.6*sharp_part
```

We assume that the signal travels through a channel where it is partially reflected off obstacles and reaches the receiver attenuated and at a delay. We model each obstacle as a linear operator on the discrete signal:

```
>>> def make_obstacle(coverage, returned, delay):
...     delay_lower = int(delay)
...     delay_upper = delay_lower + 1
```

[6]To allow for continuous validation in a timely manner, the code listings and figures of this section only are computed using MOSEK [32] instead of CVXOPT as PICOS' backend solver.

```
...     coverage_upper = coverage*(delay - delay_lower)
...     coverage_lower = coverage - coverage_upper
...     A = (1.0 - coverage)*numpy.eye(n)
...     B = coverage_lower*numpy.eye(n, k=-delay_lower)
...     C = coverage_upper*numpy.eye(n, k=-delay_upper)
...     return A + returned*(B + C)
```

The first argument `coverage` determines the fraction of the signal that is affected by the obstacle, the second argument `returned` denotes the fraction of the affected part of the signal that reaches the receiver at a delay and the last argument `delay` defines this delay in discrete time steps. Let us demonstrate this with a trivial signal vector:

```
>>> signal = numpy.array([1.0] + [0.0]*(n - 1))
>>> print(signal)
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
>>> obstacle = make_obstacle(0.8, 0.5, 2.75)
>>> print(obstacle @ signal)
[0.2 0.  0.1 0.3 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
```

Here 80% of the initial signal is affected by the obstacle such that one half is lost and the other half is delayed by 2.75 time steps, which is modeled as delaying 75% of the signal strength by 3 and 25% of it by 2 discrete time steps.

To add an element of uncertainty, the obstacles that describe our transmission channel are not fixed but may slightly vary their properties between transmissions. We model this by adding Gaussian noise to all three creation parameters:

```
>>> def make_noisy_obstacle(coverage, returned, delay):
...     c, r, d = numpy.clip(
...         numpy.random.normal(1.0, 0.25, 3), 0.0, None)
...     return make_obstacle(c*coverage, r*returned, d*delay)
```

Finally, a channel is defined as a composition of uncertain obstacles:

```
>>> def make_channel():
...     A = make_noisy_obstacle(0.2, 0.5, 0.25*n)
...     B = make_noisy_obstacle(0.2, 0.5, 0.50*n)
...     C = make_noisy_obstacle(0.2, 0.5, 0.75*n)
...     return A @ B @ C
```

Note that by definition, channels are lower triangular matrices:

```
>>> channel = make_channel()
>>> numpy.allclose(channel, numpy.tril(channel))
True
```

Given an uncertain communication channel C distributed according to `make_channel`, we would like the receiver to estimate the optimal signal recovery matrix $R_{\text{opt}} := \mathbb{E}\left[C^{-1}\right]$ by observing a small number of known signals that are perturbed by realizations of C. We represent these training samples as $n \times 2$ matrices where the first column denotes the clean and the second column the perturbed signal:

```
>>> def sample():
...     signal = make_signal()
...     channel = make_channel()
...     return numpy.vstack([signal, channel @ signal]).T
```

Whenever we pass a set of scenarios to `ScenarioPerturbationSet` or hand a discrete sample space to `WassersteinAmbiguitySet`, PICOS uses the new `Samples` class to represent these points. We may also instantiate this class directly, which we do here to show some of its features:

```
>>> N, M = 4, 1000  # Number of training and validation samples.
>>> S = picos.Samples([sample() for _ in range(N + M)]); S
<Samples: (1004 32-dimensional samples)>
>>> S.original_shape  # Shape of samples before vectorization.
(16, 2)
>>> T, V = S.partition(N)
>>> T.num, V.num
(4, 1000)
```

Apart from the partition into training and validation samples that we used above, the `Samples` class provides access to its data in different formats and can compute simple statistics such as the sample mean and the sample covariance matrix, which may be useful when working with a `MomentAmbiguitySet`.

Now that we have a set of training samples, let us estimate R_{opt} using different methods. As a baseline approach we compute the minimum sample average mean squared error estimator R_{mse}:

```
>>> X = T.matrix[:n, :]  # Clean signals.
>>> Y = T.matrix[n:, :]  # Perturbed signals.
>>> P_mse = picos.Problem()
>>> R_mse = picos.LowerTriangularVariable("R", (n, n))
>>> P_mse.set_objective("min", abs(R_mse*Y - X)**2 / (N*n))
>>> _ = P_mse.solve()
>>> round(P_mse, 4)
0.0002
```

In the listing above we first split the sample matrix, `T.matrix`, along the first axis to obtain two matrices whose columns are the clean and the perturbed signals, respectively.

This makes use of the fact that the original samples obtained from the `sample` function were vectorized in column-major order when we passed them to `Samples`. Given these $n \times N$ matrices X and Y, we define the estimator as

$$R_{\text{mse}} := \operatorname*{arg\,min}_{R \in \mathbb{R}^{n \times n} \text{ l.t.}} \frac{1}{Nn} \|RY - X\|^2.$$

Note that the optimal recovery matrix R_{opt} is lower triangular by the definition of C so that we also limit our estimator to a lower triangular matrix, which reduces the number of scalar variables passed to the solver. As $X = RY$ is an underdetermined system,[7] it is not surprising that the optimal value of `P_mse` turns out to be small.[8] However, we can expect degraded out-of-sample performance as a result of overfitting. To see this, let us also compute the average mean squared error under the validation set:

```
>>> def amse(recovery_matrix, samples):
...     R = recovery_matrix.value
...     X = samples.matrix[:n, :].value
...     Y = samples.matrix[n:, :].value
...     return sum(abs(R*Y - X)**2) / (len(samples)*n)
>>> round(amse(R_mse, V), 4)
35.5074
```

We can compare this to the average mean squared error of the perturbed signal:

```
>>> R_id = picos.Constant("I", shape=(n, n))
>>> round(amse(R_id, V), 4)
0.1191
```

In this particular example, signal recovery with R_{mse} *increases* the out-of-sample error by a magnitude. A standard approach to prevent such catastrophic overfitting is through regularization. L_2-regularization is particularly attractive as our problem remains unconstrained quadratic. This gives us the estimator R_{reg}:

```
>>> lbd = 0.01
>>> X, Y = T.matrix[:n, :], T.matrix[n:, :]
>>> P_reg = picos.Problem()
>>> R_reg = picos.LowerTriangularVariable("R", (n, n))
>>> P_reg.set_objective(
...     "min", abs(R_reg*Y - X)**2 + lbd*abs(R_reg)**2)
>>> _ = P_reg.solve()
```

[7] The number of scalar constraints $Nn = 64$ is strictly smaller than the number of degrees of freedom of the lower triangular variable R of $\frac{1}{2}(n+1)n = 136$.

[8] The optimal value being nonzero implies that $X = RY$ has no solution. This can be explained by R being lower triangular: In particular N scalar constraints depend only on the scalar variable $R_{1,1}$.

We omit the division by Nn in the objective function as this does not change the set of optimal solutions and because the objective value does not accurately represent the in-sample average mean squared error any more. Let us thus compute both in-sample and out-of-sample errors using the `amse` function:

```
>>> round(amse(R_reg, T), 4)  # In-sample error.
0.0006
>>> round(amse(R_reg, V), 4)  # Out-of-sample error.
0.0669
```

With $\lambda = \frac{1}{100}$, we can already see a significant improvement in the out-of-sample performance. We will later see that this choice is essentially optimal for our model.

Next, let us apply Wasserstein DRO to find a better estimate. Here the empirical distribution on the training samples serves as a nominal distribution that we believe to be close to the real data generating distribution, represented by the `sample` function. To achieve this, we may pass the `Samples` instance `T` directly to the `samples` argument of the class that represents a Wasserstein ambiguity set:

```
>>> D = picos.uncertain.WassersteinAmbiguitySet(
...     parameter_name="[x, y]", p=2, eps=0.6, samples=T)
```

The arguments are given in their default order. The arguments `p` and `eps` represent the hyperparameters p and ϵ of $\mathcal{D}_{W_p}$. There is a fifth argument, `weights`, that controls the relative probability mass of each sample. This can be used to represent arbitrary finite and discrete nominal distributions. Setting a weight to zero would remove the respective sample from the sample space to improve performance. For the empirical nominal distribution we leave this argument at its default value that assigns equal mass to each sample. From the `parameter` attribute of `D`, we may now extract a pair of random variables that represent the clean and the perturbed signal:

```
>>> x = D.parameter[:, 0].renamed("x")
>>> y = D.parameter[:, 1].renamed("y")
>>> x
<16×1 Uncertain Affine Expression: x>
```

With these variables, we can compute a distributionally robust estimator R_{dro} as follows:

```
>>> P_dro = picos.Problem()
>>> R_dro = picos.LowerTriangularVariable("R", (n, n))
>>> P_dro.set_objective("min", abs(R_dro*y - x)**2)
>>> _ = P_dro.solve()
```

Let us evaluate its performance:

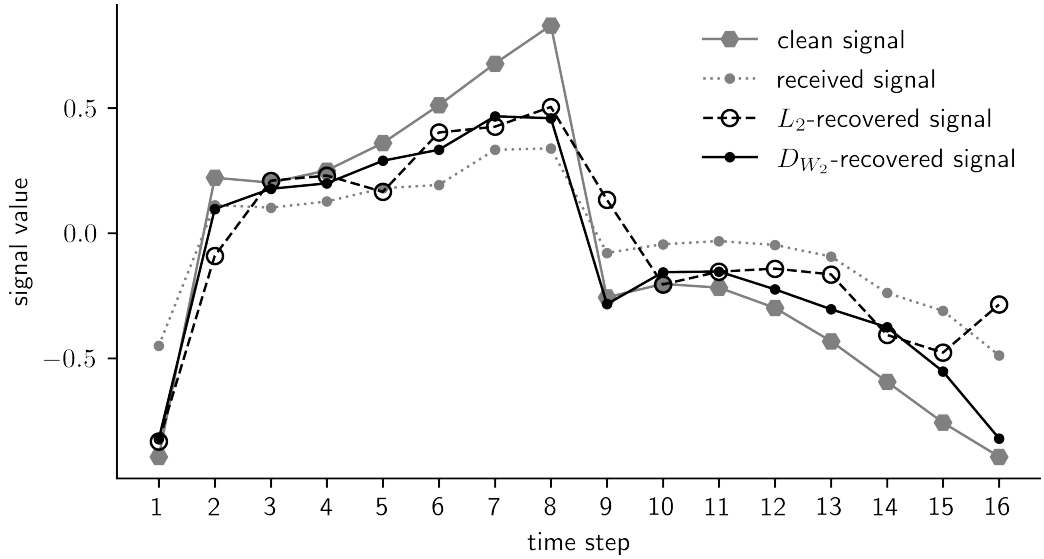

Figure 4: A signal perturbed by transmission through a channel with uncertain properties is reconstructed by minimizing the L_2-regularized sample average MSE and the Wasserstein-ambiguous expected MSE of a small number of training samples.

```
>>> round(amse(R_dro, T), 4)  # In-sample error.
0.0018
>>> round(amse(R_dro, V), 4)  # Out-of-sample error.
0.0236
```

While the quality of a single estimate does not prove much, this significant reduction in the out-of-sample error compared to R_{reg} is an encouraging first impression. Before we present the results of a larger numerical experiment, let us visualize both estimators in action on a selected validation sample:

```
>>> sample = 63
>>> x, y = V[sample][:n].value, V[sample][n:].value
>>> _ = pyplot.plot(t, x, "H-", color="gray", label="clean signal")
>>> _ = pyplot.plot(t, y, ".:", color="gray", label="received signal")
>>> _ = pyplot.plot(t, R_reg.value*y, "ko--", fillstyle="none",
...                 label="$L_2$-recovered signal")
>>> _ = pyplot.plot(t, R_dro.value*y, "k.-",
...                 label="$D_{W_2}$-recovered signal")
>>> _ = pyplot.legend()
>>> pyplot.show()
```

Up to additional formatting, this will show figure 4. Note that the sample we've selected is in fact representative of the out-of-sample performances of both estimates:

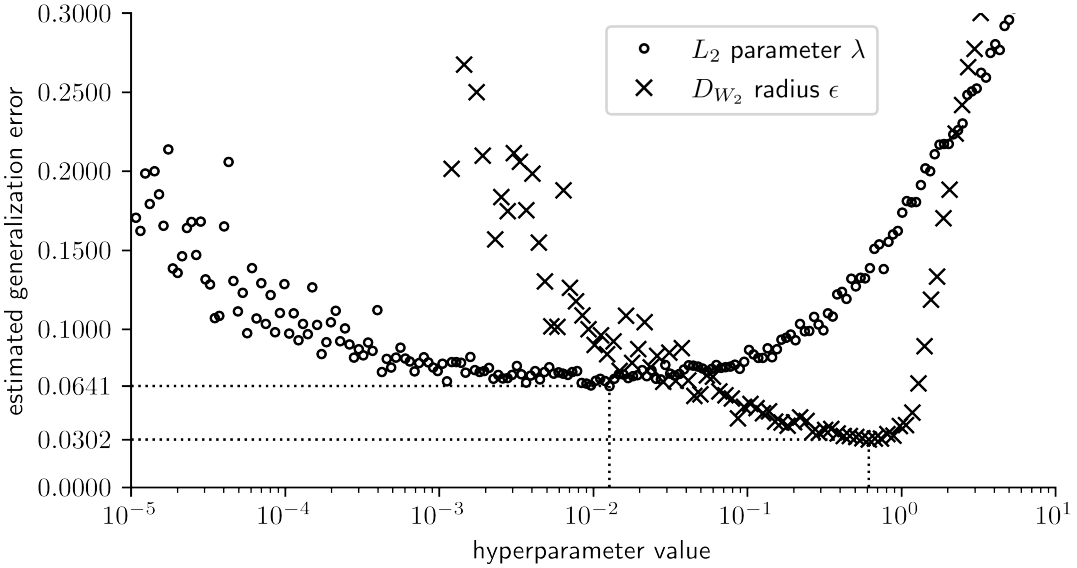

Figure 5: A logarithmic scale hyperparameter grid search that shows superior generalization performance of Wasserstein-ambiguous expected MSE minimization over L_2-regularized sample average MSE minimization for an application in linear signal estimation.

```
>>> W = V.select([sample])  # Create a singleton Samples instance.
>>> round(abs(amse(R_reg, W) - amse(R_reg, V)), 4)
0.0009
>>> round(abs(amse(R_dro, W) - amse(R_dro, V)), 4)
0.0007
```

Last, we provide numerical results that show, for the particular application discussed above, that the Wasserstein estimator generalizes better than the L_2-regularized mean squared error estimator if both are parameterized adequately. To this end we first draft a single validation set V of 1000 samples from the `sample` function. Then, for a selection of hyperparameters λ and ϵ respectively, we train R_{mse} and R_{dro} as defined above 50 times on independently drawn training sets of N samples each. The *average* average mean squared error of the 50 estimator instances under V then serves as an *estimated generalization error* for the particular hyperparameter value. The results are shown in figure 5. We can see that the lowest error estimate for the L_2-regularization approach is found at $\lambda \approx 0.01$ and lets us expect an out-of-sample error of around 0.0641. Meanwhile, Wasserstein-ambiguous DRO appears to give good results for $\epsilon \approx 0.6$ with an estimated generalization error of around 0.0302; less than half that of the L_2 approach.

6. Implementation

This section documents the implementation of the results of sections 3 and 4 within PICOS to an extend that we anticipate to be interesting to a mathematical audience and to authors of similar software. For a more general outline of the code changes that comprise the practical part of this thesis, see appendix A.

6.1. The PICOS backend

We first give a brief overview of the internal structure of the PICOS package. The source code is organized in a number of subpackages (folders), each of which contains a number of modules (Python source files):

- The `picos` root package contains miscellaneous modules that deal with generic tasks such as caching results or printing mathematical expressions to the console,
- `picos.expressions` defines algebraic expression types including constants, variables, affine expressions and norms as well as algebraic set types such as those representing basic cones,
- `picos.constraints` implements the constraint types that are produced when the user compares expressions using the overloaded arithmetic operators,
- `picos.modeling` implements the `Problem` and `Solution` classes as well as classes used internally to manage a solution strategy,
- `picos.reforms` defines the problem reformulation recipes that comprise the first part of a solution strategy and, lastly,
- `picos.solvers` implements interfaces to the low level conic optimization solvers that constitute the backend of a solution strategy.

It should be noted that the `picos` *namespace* (the set of names `x` that are available as `picos.x` following **`import`** *`picos`*) contains a selection of classes and functions from all subpackages that are meant to be used by the user directly, such as the `RealVariable` class from the `picos.expressions.variables` module. This way the user does not need to know about the internal structure and has the most important tools at the ready.

The main job of PICOS is to reformulate the high level optimization problem input by the user to a low level conic problem in a format accepted by the optimization solver in use. Up to the final step where the problem is passed to the solver API, these reformulations coincide with the mathematical concept of a reformulation. For instance, the `EpigraphReformulation` class replaces the objective function with an auxiliary variable and creates a new constraint that bounds the original objective function by that variable. In order to know which reformulations to use and in what order, PICOS first produces the *footprint* of the problem at hand, which describes in detail the types and amount of variables and constraints that are present. Then, PICOS predicts the effect of all applicable reformulations on the footprint, producing new footprints of hypothetical

reformulated problems. This process is repeated as a breadth-first search in the space of reachable footprints until certain criteria are met, in particular until the problem is conceptionally understood by a solver.

Most problem reformulations are *constraint reformulations* that eliminate one type of constraint from the problem. Usually every instance of a "high level" constraint is replaced by a number of "low level" constraints and a collection of auxiliary variables. For instance, the reformulation associated with the `GeometricMeanConstraint` replaces all bounds on the geometric mean over the elements of an affine expressions with a number of rotated second order conic constraints and a selection of fresh real variables. From a purely technical perspective, constraint reformulations are different from other reformulations in the sense that their logic is not implemented in the `picos.reforms` package but within the constraint classes in the `picos.constraints` package. More precisely, every higher level constraint class defines an inner class that inherits from a base class named `ConstraintConversion`. This inner class implements a `predict` method that describes how the resulting reformulation is applied to a problem footprint and a `convert` methods that describes how one instance of the constraint is transformed. This is precisely how the main results of sections 3 and 4 are implemented.

New code. The code implemented as a result of this work enhances the internal structure as follows. First, uncertain expression types, namely

- the basic `UncertainAffineExpression`,
- the conic quadratic `UncertainNorm` and `UncertainSquaredNorm` and
- two classes representing convex and concave piecewise linear functions with random uncertainty in the data, `RandomMaximumAffine` and `RandomMinimumAffine`,

are implemented in a new `picos.expressions.uncertain` subpackage. This package also implements the perturbation universes discussed in section 5.2 as well as the perturbation parameter class `Perturbation` which dual-inherits from `Mutable` (the new base class for variables and parameters) and `UncertainAffineExpression`. Then, a number of new constraint types are implemented in a `picos.constraints.uncertain` subpackage. There is one module and constraint class per supported combination of perturbation universe and expression type, corresponding to theorems 3.1 to 3.3, 4.3, 4.10, 4.15 and 4.16 for a total of seven robust constraint types. As explained above, the constraint classes also contain the logic needed for a finite and conic conversion of that constraint. Lastly, additional modules have been added to existing packages, in particular `picos.expressions.exp_biaffine` contains a new base class `BiaffineExpression` used for both certain and uncertain affine expressions and the module `picos.uncertain` provides a namespace for the user to access the new features.

In the following we will discuss exclusively the new `BiaffineExpression` class that we use to represent functions that are biaffine in the decision variables and in a single perturbation parameter describing the uncertain data. Apart from considerations of space,

the reason for this choice is that most other changes are either technical modifications to the PICOS backend or straightforward transformations of mathematical results to code. Additionally, this class alone makes for roughly 20% of the implementation effort.[9]

6.2. Biaffine expressions

Using PICOS 2.0, it seems that uncertain data could already be represented as an affine expression of a perturbation parameter modeled as a variable. Given a nominal data vector $a \in \mathbb{R}^m$, a matrix $P \in \mathbb{R}^{m \times n}$ and a perturbation parameter $t \in \mathbb{R}^n$, we could try to define the perturbed data $p(t) := a + Pt$ as follows:

```
>>> m, n = 4, 3
>>> a_data, P_data = range(m), range(m*n)
>>> a = picos.Constant("a", a_data, m)
>>> P = picos.Constant("P", P_data, (m, n))
>>> t = picos.RealVariable("t", n)
>>> p = a + P*t
```

However, if a decision variable $x \in \mathbb{R}^m$ is introduced to form an uncertain affine expression involving p, we obtain a nonconvex quadratic expression instead:

```
>>> x = picos.RealVariable("x", m)
>>> p.T*x
<Quadratic Expression: (a + P·t)ᵀ·x>
```

Of course, so far there is no way for PICOS to make any structural distinction between `t` and `x` as both are modeled as regular decision variables. A new expression type, `UncertainAffineExpression`, is needed to distinguish between decision variables that are handed to a solver and perturbation parameters that are eliminated when forming the robust counterpart of a problem. This new expression is biaffine in a collection of decision variables on one side and, by convention, a single perturbation parameter on the other side. To avoid code duplication between `UncertainAffineExpression` and the certain `(Complex)AffineExpression`, we implement a common base class, coined `BiaffineExpression`, which will contain the majority of the logic previously implemented in `ComplexAffineExpression`. In the following we discuss the implementation of this new class.

6.2.1. Representation

First, we recapitulate the data model of `ComplexAffineExpression`, which remains the base class for the real `AffineExpression` and becomes a child class of the new `BiaffineExpression`. Note that since all PICOS expressions are immutable, they are

[9] Its module adds around 2700 out of 13700 new lines of code, measured in git additions.

fully defined by their instantiation parameters. In PICOS 2.0, creating an instance of `ComplexAffineExpression` is done writing

```
ComplexAffineExpression(string, shape, coefficients, constant)
```

where `string` is an algebraic string description, `shape` are the dimensions of a matrix expression, `coefficients` describes the linear part and `constant` describes the constant part of the affine expression. The semantics of the latter two parameters can be formalized as follows: Given an affine expression $A := \sum_{i=1}^{k} A_i(x_i) + A_0 \in \mathbb{C}^{m \times n}$ concerning variables $x_i \in \mathbb{R}^{\alpha_i}$, $i \in [k]$, the `coefficients` argument represents the mapping $\{x_i \mapsto \mathfrak{A}_i \mid i \in [k]\}$ where $\mathfrak{A}_i \in \mathbb{C}^{mn,\alpha_i}$ with $i \in [k]$ is the unique matrix given by $\operatorname{reshape}_{m,n}(\mathfrak{A}_i x_i) = A_i(x_i)$. Similarly, the `constant` argument represents a single vector $\mathfrak{A}_0 \in \mathbb{R}^{mn}$ with $\operatorname{reshape}_{m,n}(\mathfrak{A}_0) = A_0$.

To make the step from affine to biaffine expressions, we enhance the `coefficients` argument. Formally, consider now a biaffine expression

$$A := \sum_{i<j} A_{(i,j)}(x_{(i)}, x_{(j)}) + \sum_{i} A_{(i)}(x_{(i)}) + A_{()} \in \mathbb{C}^{m \times n}$$

concerning again the variables $x_{(i)} \in \mathbb{R}^{\alpha_i}$, $i \in [k]$. To have a unified representation for the constant, linear and bilinear terms of A, the new `coefficients` parameter now maps from tuples of variables to coefficient matrices and can be formalized as

$$\begin{aligned} \operatorname{Coefs}(A) := \; & \big\{(x_{(i)}, x_{(j)}) \mapsto \mathfrak{A}_{(i,j)} \mid 1 \leqslant i < j \leqslant k\big\} \\ & \cup \big\{(A_{(i)}) \mapsto \mathfrak{A}_{(i)} \mid 1 \leqslant i \leqslant k\big\} \\ & \cup \big\{() \mapsto \mathfrak{A}_{()}\big\} \end{aligned}$$

where, for all $i, j \in [k]$ with $i < j$,

- $\mathfrak{A}_{(i,j)} \in \mathbb{C}^{mn,\alpha_i\alpha_j}$ is given by $\operatorname{reshape}_{m,n}\left(\mathfrak{A}_{(i,j)} \operatorname{vec}\left(x_{(i)} x_{(j)}^{T}\right)\right) = A_i(x_{(i)}, x_{(j)})$,
- $\mathfrak{A}_{(i)} \in \mathbb{C}^{mn,\alpha_i}$ is given by $\operatorname{reshape}_{m,n}\left(\mathfrak{A}_{(i)} x_{(i)}\right) = A_{(i)}(x_{(i)})$ and
- $\mathfrak{A}_{()} \in \mathbb{C}^{mn}$ is given by $\operatorname{reshape}_{m,n}\left(\mathfrak{A}_{()}\right) = A_{()}$.

Clearly, this approach allows us to get rid of the additional `constant` argument so that instantiation of any `BiaffineExpression` subclass is performed given only `string`, `shape` and `coefficients`. For subclasses that support no bilinear part, in particular `ComplexAffineExpression` and `AffineExpression`, instantiation is still performed providing tuples of variables as keys for the `coefficients` argument, but tuples of size larger than one are disallowed and raise a `TypeError`.

This subtle enhancement of the coefficient map allows us to lift most operations defined on (complex) affine expressions to biaffine expressions with no or little modification. In the following we have a closer look at some of the existing operations and establish new logic to have them operate in the space of biaffine expressions.

6.2.2. Algebraic operations

The basic algebraic operations of addition, subtraction, scalar division and a number of products are now implemented in `BiaffineExpression` for all cases where the outcome is again biaffine. Additional support for products with quadratic outcome remains within `ComplexAffineExpression`. For instance, the scalar product

```
ComplexAffineExpression.__or__(self, other)
```

is now implemented as follows:

```
if isinstance(other, ComplexAffineExpression) \
and not self.constant and not other.constant:
    (Allow a quadratic outcome.)
else:
    return BiaffineExpression.__or__(self, other)
```

In the following we discuss operations with biaffine outcome on the two biaffine matrix expressions

$$A := \sum_{i<j} A_{(i,j)}(x_{(i)}, x_{(j)}) + \sum_{i} A_{(i)}(x_{(i)}) + A_{()} \in \mathbb{C}^{m\times n} \text{ and}$$
$$B := \sum_{i<j} B_{(i,j)}(y_{(i)}, y_{(j)}) + \sum_{i} B_{(i)}(y_{(i)}) + B_{()} \in \mathbb{C}^{p\times q}$$

concerning variables $X := \{x_{(i)} \in \mathbb{R}^{\alpha_i} \mid i \in [a]\}$ and $Y := \{y_{(i)} \in \mathbb{R}^{\beta_i} \mid i \in [b]\}$ with $|X| = a$ and $|Y| = b$. We do not require that X and Y are disjoint but we will assume in the following that any two variables that appear in a bilinear term in both A and B address their associated coefficient matrices in the same order, formally

$$\nexists x, y : \exists \mathfrak{A}, \mathfrak{B} : (x, y) \mapsto \mathfrak{A} \in \mathrm{Coefs}(A) \wedge (y, x) \mapsto \mathfrak{B} \in \mathrm{Coefs}(B).$$

This property is established in PICOS by reordering, for every new `BiaffineExpression` instance as necessary, the variables that comprise the keys of the coefficient map according to a unique identification number assigned to each variable at creation. If the order of a pair of variables needs to be changed, then the associated coefficient matrix is adjusted by multiplication with a sparse commutation matrix, which is produced by a centralized function and stored in a cache. This function and its cache are also used to compute the regular transposition of a biaffine expression and in the context of matrix multiplication (see below), providing a mutual benefit between all three use cases.

Addition & Subtraction. For $m = p$ and $n = q$, it is easy to see that the sum $A + B$ is again biaffine: In the case of $X \cap Y = \emptyset$, the coefficient map of $A + B$ is just

$$\begin{aligned}\mathrm{Coefs}(A + B) = \; & \{(x_{(i)}, x_{(j)}) \mapsto \mathfrak{A}_{(i,j)},\ (x_{(i)}) \mapsto \mathfrak{A}_{(i)} \mid 1 \leqslant i < j \leqslant a\} \\ \cup \; & \{(y_{(i)}, y_{(j)}) \mapsto \mathfrak{B}_{(i,j)},\ (y_{(i)}) \mapsto \mathfrak{B}_{(i)} \mid 1 \leqslant i < j \leqslant b\} \\ \cup \; & \{() \mapsto \mathfrak{A}_{()} + \mathfrak{B}_{()}\}.\end{aligned}$$

If some variable appears in both A and B, say $x_{(i)} = y_{(j)}$ for some i and j, then we can rewrite the sum

$$A_{(i)}(x_{(i)}) + B_{(j)}(y_{(j)}) = (A_{(i)} + B_{(j)})(x_{(i)})$$

and PICOS would store only one coefficient matrix $\mathfrak{C} := \mathfrak{A}_{(i)} + \mathfrak{B}_{(j)}$ with

$$\mathfrak{C}x_{(i)} = \operatorname{vec}\left((A_{(i)} + B_{(j)})(x_{(i)})\right)$$

to represent the partial sum over the linear forms $A_{(i)}$ and $B_{(j)}$. Likewise, if there is a second pair of variables $x_{(k)} = y_{(l)}$ for some $k > i$ and $l > j$, then the bilinear forms $A_{(i,k)}(x_{(i)}, x_{(k)})$ and $B_{(j,l)}(y_{(j)}, y_{(l)})$ concern the same variables and sum to $(A_{(i,k)} + B_{(j,l)})(x_{(i)}, x_{(k)})$, so that a single coefficient matrix $\mathfrak{D} := \mathfrak{A}_{(i,k)} + \mathfrak{B}_{(j,l)}$ with

$$\mathfrak{D} \operatorname{vec}\left(x_{(i)} x_{(k)}^T\right) = \operatorname{vec}\left((A_{(i,k)} + B_{(j,l)})(x_{(i)}, x_{(k)})\right)$$

is sufficient to represent this part of the sum of A and B.

The enhanced `coefficients` argument, stored after validation as the `_coefs` attribute of the expression, enables a compact implementation of the summation that does not require special handling of the bilinear terms. Given A as ***`self`*** and B as `other`, the following excerpt from `BiaffineExpression.__add__` computes Coefs $(A + B)$ as `coefs`:

```
coefs = {}
for vars, coef in self._coefs.items():
    coefs[vars] = coef + other._coefs[vars] \
        if vars in other._coefs else coef
for vars, coef in other._coefs.items():
    coefs.setdefault(vars, coef)
```

Subtraction is implemented analogously.

Products. The situation is not as simple when we consider products of A and B. On the one hand, new logic is needed to handle the creation of bilinear terms from two linear terms that interact in the product. On the other hand, it is possible that quadratic and higher order terms appear as a result of an interaction between nonconstant terms, so that the resulting expression is not necessarily biaffine. Making a prediction in this matter requires a look at the sparsity patterns of the coefficient matrices involved. For example, the operands of the inner product $\langle \begin{bmatrix} xy & 1 \end{bmatrix}, \begin{bmatrix} 1 & xy \end{bmatrix} \rangle = 2xy$ concerning scalar variables x and y are both properly biaffine, yet the result is also biaffine.

To keep code complexity at a reasonable level, we do not detect compliant special cases like the one in the example. If one of the operands of some matrix product is biaffine and the other one is nonconstant, then we invoke that the result is not biaffine in general and raise a `NotImplementedError`. However, we always compute the product of two properly affine expressions regardless of whether the operands have variables

in common as detecting a nonzero quadratic term in the result does not make the implementation any more involved. Here the outcome depends on the precise types of A and B. If they are both instances of `ComplexAffineExpression`, then the relevant method of that class will compute the product and a quadratic result is possible, returned in the form of a `QuadraticExpression`. Otherwise the computation is performed in `BiaffineExpression` and any quadratic outcome results in a `TypeError`.

Since all products supported by PICOS are distributive over addition, an affine by affine product can be expanded to a sum of products whose operands are limited to linear and constant terms only. Likewise, a biaffine by constant product can be expanded to a sum of products whose operands are limited to bilinear, linear, and constant terms. Products in the expansion that involve a constant operand all have a straightforward implementation that carries over from the affine expressions of PICOS 2.0 with little modification, so that in the following we limit the discussion to products of linear terms. To this end, we assume w.l.o.g. that $A = A_{(i)}(x_{(i)})$ and $B = B_{(j)}(y_{(j)})$ for some $i \in [a]$ and $j \in [b]$. We simplify notation by identifying A with $A_{(i)}$, B with $B_{(j)}$ and defining $x := x_{(i)}$, $y := y_{(j)}$, $\alpha := \alpha_i$, and $\beta := \beta_i$.

Our goal for each product $C(x, y) := A(x) \circ B(y)$ remains to compute from the coefficient matrices $\mathfrak{A}$ and $\mathfrak{B}$, defined through $\operatorname{vec}(A(x)) = \mathfrak{A}x$ and $\operatorname{vec}(B(y)) = \mathfrak{B}y$, a product coefficient matrix $\mathfrak{C}$ such that

$$\operatorname{vec}(A(x) \circ B(y)) = \mathfrak{C} \operatorname{vec}\left(xy^T\right).$$

Inner product. The (complex) inner product between two linear terms can be implemented according to the equation

$$\begin{aligned}
\operatorname{vec}(\langle A(x),\ B(y)\rangle) &= \langle \operatorname{vec}(A(x)),\ \operatorname{vec}(B(y))\rangle \\
&= \langle \mathfrak{A}x,\ \mathfrak{B}y\rangle \\
&= \operatorname{tr}\left(\mathfrak{A}x(\mathfrak{B}y)^H\right) \\
&= \operatorname{tr}\left((\mathfrak{A}^H\mathfrak{B})^H xy^H\right) \\
&= \operatorname{vec}\left(\mathfrak{A}^H\mathfrak{B}\right)^H \operatorname{vec}\left(xy^T\right)
\end{aligned}$$

which gives us $\mathfrak{C} = \operatorname{vec}\left(\mathfrak{A}^H\mathfrak{B}\right)^H$. However, we opt to reduce code duplication by reducing from the inner product to the row vector by column vector matrix multiplication by virtue of the identity

$$\langle A(x),\ B(y)\rangle = \operatorname{vec}(B(y))^H \operatorname{vec}(A(x)).$$

Note that the overhead in the form of vectorization and conjugate transposition is small: Vectorization is both cached and fast initially as no numeric computation is required and no data is copied while conjugate transposition makes use of the commutation matrix cache discussed earlier and is further realized as a *cached self-inverse property*, meaning that the outcome of $A.H$ is cached and that $(A.H).H$ returns A immediately.

To implement matrix by matrix multiplication with scalar outcome efficiently, we will need the following corollary of the inner product result:

Corollary 6.1. *The identity*

$$(\mathfrak{A}x)^{\mathsf{T}}(\mathfrak{B}y) = \operatorname{vec}\left(\mathfrak{A}^{\mathsf{T}}\mathfrak{B}\right)^{\mathsf{T}} \operatorname{vec}\left(xy^{\mathsf{T}}\right)$$

holds for all $\mathfrak{A} \in \mathbb{C}^{m\times a}$, $\mathfrak{B} \in \mathbb{C}^{m\times b}$, $x \in \mathbb{R}^a$, *and* $y \in \mathbb{R}^b$.

Proof. $(\mathfrak{A}x)^{\mathsf{T}}(\mathfrak{B}y) = \langle \mathfrak{A}x,\ \overline{\mathfrak{B}}y\rangle = \operatorname{vec}\left(\mathfrak{A}^{\mathsf{H}}\overline{\mathfrak{B}}\right)^{\mathsf{H}} \operatorname{vec}\left(xy^{\mathsf{T}}\right) = \operatorname{vec}\left(\mathfrak{A}^{\mathsf{T}}\mathfrak{B}\right)^{\mathsf{T}} \operatorname{vec}\left(xy^{\mathsf{T}}\right).$ □

Matrix multiplication. For matrix multiplication we require $n = p$ so that

$$C(x,y) = A(x)B(y) \in \mathbb{C}^{m\times q}.$$

As suggested above, the special case of $m = 1$ and $q = 1$ is handled by virtue of corollary 6.1 according to

$$\begin{aligned}\operatorname{vec}(A(x)B(y)) = &\ \operatorname{vec}\left(\operatorname{vec}(A(x))^{\mathsf{T}} \operatorname{vec}(B(y))\right)\\ = &\ (\mathfrak{A}x)^{\mathsf{T}}(\mathfrak{B}y)\\ = &\ \operatorname{vec}\left(\mathfrak{A}^{\mathsf{T}}\mathfrak{B}\right)^{\mathsf{T}} \operatorname{vec}\left(xy^{\mathsf{T}}\right)\end{aligned}$$

which gives us $\mathfrak{C} = \operatorname{vec}\left(\mathfrak{A}^{\mathsf{T}}\mathfrak{B}\right)^{\mathsf{T}}$.

For the general case, let $A_i x \in \mathbb{C}^n$ with $A_i \in \mathbb{C}^{n\times\alpha}$ be the transposed rows of $A(x)$ and $B_j y \in \mathbb{C}^p$ with $B_j \in \mathbb{C}^{p\times\beta}$ the columns of $B(y)$ for $i \in [m]$ and $j \in [q]$, so that

$$A(x)^{\mathsf{T}} = \begin{bmatrix} A_1 x & \cdots & A_m x\end{bmatrix}, \qquad B(y) = \begin{bmatrix} B_1 y & \cdots & B_q y\end{bmatrix}.$$

Then, by corollary 6.1,

$$C(x,y)_{(i,j)} = (A_i x)^{\mathsf{T}}(B_j y) = \operatorname{vec}\left(A_i^{\mathsf{T}} B_j\right)^{\mathsf{T}} \operatorname{vec}\left(xy^{\mathsf{T}}\right)$$

for all $(i,j) \in [(m,q)]$. Thus, the k-th row of $\mathfrak{C}$ is

$$\mathfrak{C}_{k,:} = \operatorname{vec}\left(A^{\mathsf{T}}_{k \,\%\, m} B_{k \div m}\right)^{\mathsf{T}} \in \mathbb{C}^{1\times\alpha\beta}.$$

Let further $\mathfrak{A}' := K_{(m,n)}\mathfrak{A}$ so that $\mathfrak{A}'x = \operatorname{vec}\left(A(x)^{\mathsf{T}}\right)$. This allows us to write $\mathfrak{A}'$ and $\mathfrak{B}$ as the block matrices

$$\mathfrak{A}' = \begin{bmatrix} A_1\\ \vdots\\ A_m\end{bmatrix} \in \mathbb{C}^{mn\times\alpha}, \qquad \mathfrak{B} = \begin{bmatrix} B_1\\ \vdots\\ B_q\end{bmatrix} \in \mathbb{C}^{pq\times\beta}.$$

Transposing and vectorizing both, we obtain the stacked vectors

$$\operatorname{vec}\left(\mathfrak{A}'^{\mathsf{T}}\right) = \begin{pmatrix} \operatorname{vec}\left(A_1^{\mathsf{T}}\right) \\ \vdots \\ \operatorname{vec}\left(A_m^{\mathsf{T}}\right) \end{pmatrix} \in \mathbb{C}^{mn\alpha}, \qquad \operatorname{vec}\left(\mathfrak{B}^{\mathsf{T}}\right) = \begin{pmatrix} \operatorname{vec}\left(B_1^{\mathsf{T}}\right) \\ \vdots \\ \operatorname{vec}\left(B_q^{\mathsf{T}}\right) \end{pmatrix} \in \mathbb{C}^{pq\beta}.$$

This lets us obtain two distinct partial transpositions of $\mathfrak{A}'$ and $\mathfrak{B}$, respectively, computing first the sparse matrix by vector multiplications

$$(I_m \otimes K_{(\alpha,n)}) \operatorname{vec}\left(\mathfrak{A}'^{\mathsf{T}}\right) = \begin{pmatrix} \operatorname{vec}(A_1) \\ \vdots \\ \operatorname{vec}(A_m) \end{pmatrix}, \quad (I_q \otimes K_{(\beta,p)}) \operatorname{vec}\left(\mathfrak{B}^{\mathsf{T}}\right) = \begin{pmatrix} \operatorname{vec}(B_1) \\ \vdots \\ \operatorname{vec}(B_q) \end{pmatrix}$$

followed by reshaping and, for the left hand side, transposition:

$$\mathfrak{A}'' := \operatorname{reshape}_{n,m\alpha}\left((I_m \otimes K_{(n,\alpha)}) \operatorname{vec}\left(\mathfrak{A}'^{\mathsf{T}}\right)\right)^{\mathsf{T}} = \begin{bmatrix} A_1^{\mathsf{T}} \\ \vdots \\ A_m^{\mathsf{T}} \end{bmatrix} \in \mathbb{C}^{m\alpha \times n},$$

$$\mathfrak{B}' := \operatorname{reshape}_{p,q\beta}\left((I_q \otimes K_{(p,\beta)}) \operatorname{vec}\left(\mathfrak{B}^{\mathsf{T}}\right)\right) = \begin{bmatrix} B_1 & \cdots & B_q \end{bmatrix} \in \mathbb{C}^{p \times q\beta}.$$

From here on, given $n = p$, we form the block matrix outer product

$$\mathfrak{A}''\mathfrak{B}' = \begin{bmatrix} A_1^{\mathsf{T}}B_1 & \cdots & A_1^{\mathsf{T}}B_q \\ \vdots & \ddots & \vdots \\ A_m^{\mathsf{T}}B_1 & \cdots & A_m^{\mathsf{T}}B_q \end{bmatrix} \in \mathbb{C}^{m\alpha, q\beta}.$$

and we arrive at $\mathfrak{C}$ by first performing a blockwise vectorization

$$\operatorname{vecb}_{(m,q),(\alpha,\beta)}\left(\mathfrak{A}''\mathfrak{B}'\right) = \begin{pmatrix} \operatorname{vec}\left(A_1^{\mathsf{T}}B_1\right) \\ \vdots \\ \operatorname{vec}\left(A_m^{\mathsf{T}}B_1\right) \\ \vdots \\ \operatorname{vec}\left(A_1^{\mathsf{T}}B_q\right) \\ \vdots \\ \operatorname{vec}\left(A_m^{\mathsf{T}}B_q\right) \end{pmatrix} \in \mathbb{C}^{mq\alpha\beta}$$

and lastly the partial transposition

$$\operatorname{reshape}_{\alpha\beta,mq}\left(\operatorname{vecb}_{(m,q),(\alpha,\beta)}\left(\mathfrak{A}''\mathfrak{B}'\right)\right)^T = \begin{bmatrix} \operatorname{vec}\left(A_1^T B_1\right)^T \\ \vdots \\ \operatorname{vec}\left(A_m^T B_1\right)^T \\ \vdots \\ \operatorname{vec}\left(A_1^T B_q\right)^T \\ \vdots \\ \operatorname{vec}\left(A_m^T B_q\right)^T \end{bmatrix} = \mathfrak{C}.$$

To obtain good performance, the computation steps for the general case are chosen such that they can be implemented using only sparse matrix operations provided by the CVXOPT library and our cached factory function for sparse commutation matrices. To see this, let `A` and `B` be the CVXOPT sparse representations of two matrices A and B, respectively, and let `K(m, n)` return a sparse representation of $K_{(m,n)}$. Then,

- A^T is denoted by `A.T` and AB can be written as `A*B`,
- $\operatorname{vec}(A)$ is performed in-place via `A.size = (A.size[0]*A.size[1], 1)`,
- $\operatorname{reshape}_{m,n}(A)$ is performed in-place via `A.size = (m, n)`,
- the matrix $I_k \otimes K_{(m,n)}$ is constructed by `cvxopt.spdiag([K(m, n)]*k)` and
- the blockwise vectorization

$$\begin{aligned} & \operatorname{vecb}_{(m,q),(\alpha,\beta)}(A) \\ = \; & \left(I_q \otimes K_{(m,\beta)} \otimes I_\alpha\right) \operatorname{vec}(A) \\ = \; & \left(\left(I_q \otimes K_{(m,\beta)}\right) \otimes I_\alpha\right) \operatorname{vec}\left(\operatorname{reshape}_{\alpha,mq\beta}(A)\right) \\ = \; & \operatorname{vec}\left(I_\alpha \operatorname{reshape}_{\alpha,mq\beta}(A) \left(I_q \otimes K_{(m,\beta)}\right)^T\right) \\ = \; & \operatorname{vec}\left(\operatorname{reshape}_{\alpha,mq\beta}(A) \left(I_q \otimes K_{(\beta,m)}\right)\right) \end{aligned}$$

  is obtained through a combination of the above.

Hadamard Product. While a local implementation handles the case were one of the operands is constant, the general case of the Hadamard product $S \odot T$ between biaffine expressions $S, T \in \mathbb{C}^{s,t}$ is reduced to a matrix multiplication according to

$$\begin{aligned} S \odot T &= \operatorname{reshape}_{s,t}(\operatorname{vec}(S \odot T)) \\ &= \operatorname{reshape}_{s,t}(\operatorname{vec}(S) \odot \operatorname{vec}(T)) \\ &= \operatorname{reshape}_{s,t}(\operatorname{Diag}(\operatorname{vec}(S)) \operatorname{vec}(T)). \end{aligned}$$

This is expressed using existing methods of `BiaffineExpression` as

```
(S.diag*T.vec).reshaped((s, t)).renamed(string).
```

The reduction is justified by the fact that the runtime of `vec`, `reshaped` and `renamed` does not depend on the shape of the operands[10] while `diag` produces sparse coefficient matrices that can be exploited by the subsequent multiplication.

Kronecker Product. The Kronecker product $A(x) \otimes B(y)$ is a block matrix of the form

$$A(x) \otimes B(y) = \begin{bmatrix} A(x)_{1,1}B(y) & \cdots & A(x)_{1,n}B(y) \\ \vdots & \ddots & \vdots \\ A(x)_{m,1}B(y) & \cdots & A(x)_{m,n}B(y) \end{bmatrix} \in \mathbb{C}^{mp \times nq}$$

whose blockwise vectorization can itself be expressed as a Kronecker product:

$$\begin{aligned} & \operatorname{vecb}_{(m,n),(p,q)}(A(x) \otimes B(y)) \\ = \; & (I_n \otimes K_{(m,q)} \otimes I_p) \operatorname{vec}(A(x) \otimes B(y)) \\ = \; & \begin{bmatrix} \operatorname{vec}\left(A(x)_{1,1}B(y)\right) \\ \vdots \\ \operatorname{vec}\left(A(x)_{m,1}B(y)\right) \\ \vdots \\ \operatorname{vec}\left(A(x)_{1,n}B(y)\right) \\ \vdots \\ \operatorname{vec}\left(A(x)_{m,n}B(y)\right) \end{bmatrix} = \begin{bmatrix} A(x)_{1,1} \operatorname{vec}(B(y)) \\ \vdots \\ A(x)_{m,1} \operatorname{vec}(B(y)) \\ \vdots \\ A(x)_{1,n} \operatorname{vec}(B(y)) \\ \vdots \\ A(x)_{m,n} \operatorname{vec}(B(y)) \end{bmatrix} \\ = \; & \operatorname{vec}(A(x)) \otimes \operatorname{vec}(B(y)) \in \mathbb{C}^{mnpq \times 1}. \end{aligned}$$

As $K_{(m,q)}$ is a permutation matrix, it is easy to see that also $I_n \otimes K_{(m,q)}$ and further $I_n \otimes K_{(m,q)} \otimes I_p$ must be permutation matrices. Therefore, the inverse

$$\begin{aligned} P &:= (I_n \otimes K_{(m,q)} \otimes I_p)^{-1} \\ &= (I_n \otimes K_{(m,q)} \otimes I_p)^T \\ &= I_n \otimes K_{(q,m)} \otimes I_p \end{aligned}$$

exists and from the equation

$$\begin{aligned} \operatorname{vec}(A(x) \otimes B(y)) &= P\big(\operatorname{vec}(A(x)) \otimes \operatorname{vec}(B(y))\big) \\ &= P(\mathfrak{A}x \otimes \mathfrak{B}y) \\ &= P \operatorname{vec}\left(\mathfrak{B}y(\mathfrak{A}x)^T\right) \\ &= P \operatorname{vec}\left(\mathfrak{B}(yx^T)\mathfrak{A}^T\right) \\ &= P(\mathfrak{A} \otimes \mathfrak{B}) \operatorname{vec}\left(yx^T\right) \\ &= P(\mathfrak{A} \otimes \mathfrak{B})K_{(\alpha,\beta)} \operatorname{vec}\left(xy^T\right) \end{aligned}$$

[10] Used on a biaffine expression A, each of them runs in $\mathcal{O}(|\operatorname{Coefs}(A)|)$ time.

we obtain

$$\mathfrak{C} = (I_n \otimes K_{(q,m)} \otimes I_p)(\mathfrak{A} \otimes \mathfrak{B})K_{(\alpha,\beta)}.$$

Unfortunately, CVXOPT matrices do not support the Kronecker product at the time, while NumPy does support the product but doesn't have a representation for sparse matrices. As adding another library to PICOS' dependencies for this particular operation would be too costly in terms of import time and portability, we work around the issue by converting $\mathfrak{A}$ and $\mathfrak{B}$ to NumPy dense matrices, computing the product, and loading the result as a CVXOPT (sparse) matrix. For the left hand side factor of $I_n \otimes K_{(q,m)} \otimes I_p$, where we know with certainty that every matrix involved is sparse, we use the existing PICOS functions `left_kronecker_I` and `right_kronecker_I`. The former is implemented efficiently using CVXOPT's `spdiag` function while the latter is implemented in Python.[11] Finally, we perform a regular multiplication involving three CVXOPT matrices.

6.2.3. Support operations

Apart from basic algebraic operations, additional operations become necessary to support the use of biaffine expressions to represent uncertain affine ones. The following operations all have a potential use outside the context of optimization under uncertainty, though, and are thus implemented for all biaffine or even for all expression types, with most of the logic living in `BiaffineExpression` in the latter case. More precisely, if one of the first two operations discussed is performed on a non-biaffine expression, usually a function of one or more affine expression, then it is invoked recursively on that function's operands until `BiaffineExpression` instances are found at the leafs of the recursion tree. The original expression is then rebuilt using the modified leaf expressions.

For this section, we consider again the biaffine expression

$$A := \sum_{i<j} A_{(i,j)}(x_{(i)}, x_{(j)}) + \sum_i A_{(i)}(x_{(i)}) + A_{()} \in \mathbb{C}^{m \times n}$$

as defined in detail in section 6.2.2. The operations discussed here all function with respect to a particular variable in A that is either replaced or factored out of the expression. Therefore, observe that terms in A that depend on some variable $x_{(i)}$ fall into three categories:

$$A_{(i)}(x_{(i)}) = \operatorname{reshape}_{m,n}\left(\mathfrak{A}_{(i)} x_{(i)}\right), \tag{6.1}$$

$$A_{(i,j)}(x_{(i)}, x_{(j)}) = \operatorname{reshape}_{m,n}\left(\mathfrak{A}_{(i,j)} \operatorname{vec}\left(x_{(i)} x_{(j)}^T\right)\right) \text{ with } i < j, \tag{6.2}$$

$$A_{(h,i)}(x_{(h)}, x_{(i)}) = \operatorname{reshape}_{m,n}\left(\mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} x_{(i)}^T\right)\right) \text{ with } h < i. \tag{6.3}$$

We will refer to these cases in the context of each of the operations outlined below.

[11] If the operand was a dense matrix, NumPy would be used instead.

Partial freezing. When a robust solution to an optimization problem affected by uncertainty is obtained, then that solution will assign values to the decision variables of the problem but it will not entail a particular realization of the uncertainty. As a result, it is not possible to ask for the value of an uncertain objective function or the slack value of uncertain constraints in the solved problem as these values still depend on the value of the uncertain perturbation parameter, which is not known. However, depending on the type of dependence and the perturbation set or distribution, we may still be able to compute a worst case or expected value of an expression by replacing the decision variables with their current value and performing an analysis of the resulting expression, which now depends only on the perturbation parameter. We call this replacement *partial freezing* as a part of the expression is frozen at its current value while the remainder stays variable. This extends the existing freezing operator, denoted for an expression `A` by `~A`, which returns a constant PICOS expression that is equal to the current value of `A`, given that that value is defined.

Assume that we want to freeze each occurrence of $x_{(i)} \in \mathbb{R}^{\alpha_i}$ in A at its current value of $\widetilde{x_{(i)}}$. We write $A[x_{(i)} \mapsto \widetilde{x_{(i)}}]$ to denote the resulting expression. In order to find $\operatorname{Coefs}\left(A[x_{(i)} \mapsto \widetilde{x_{(i)}}]\right)$, we identify $x_{(i)}$ with $\widetilde{x_{(i)}}$ and distinguish the three types of occurrences of $x_{(i)}$ in A listed above. Case (6.1) is trivial; the linear term $A_{(i)}(x_{(i)})$ becomes constant and we obtain the partial coefficient map

$$M_1 := \left\{() \mapsto \mathfrak{A}_{()} + \mathfrak{A}_{(i)}\widetilde{x_{(i)}}\right\}.$$

which represents the constant part of $A[x_{(i)} \mapsto \widetilde{x_{(i)}}]$. For case (6.2), we have

$$\begin{aligned}\mathfrak{A}_{(i,j)} \operatorname{vec}\left(x_{(i)}x_{(j)}^T\right) &= \mathfrak{A}_{(i,j)} \operatorname{vec}\left(\widetilde{x_{(i)}}x_{(j)}^T\right)\\ &= \mathfrak{A}_{(i,j)}(I_{\alpha_j} \otimes \widetilde{x_{(i)}})x_{(j)}\end{aligned}$$

which yields

$$M_2 := \left\{x_{(j)} \mapsto \mathfrak{A}_{(j)} + \mathfrak{A}_{(i,j)}(I_{\alpha_j} \otimes \widetilde{x_{(i)}}) \mid i < j\right\}$$

representing the new coefficients concerning the $x_{(j)}$. Case (6.3) is similar,

$$\begin{aligned}\mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)}x_{(i)}^T\right) &= \mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)}\widetilde{x_{(i)}}^T\right)\\ &= \mathfrak{A}_{(h,i)}(\widetilde{x_{(i)}} \otimes I_{\alpha_h})x_{(h)},\end{aligned}$$

and produces

$$M_3 := \left\{x_{(h)} \mapsto \mathfrak{A}_{(h)} + \mathfrak{A}_{(h,i)}(\widetilde{x_{(i)}} \otimes I_{\alpha_h}) \mid h < i\right\}.$$

We conclude that

$$\begin{aligned}\operatorname{Coefs}\left(A[x_{(i)} \mapsto \widetilde{x_{(i)}}]\right) = \;& \left\{V \mapsto C \in \operatorname{Coefs}(A) \mid x_{(i)} \text{ not in } V\right\}\\ \cup\; & (M_1 \cup M_2 \cup M_3).\end{aligned}$$

To extend partial freezing to multiple variables, we need to consider a fourth case where both variables involved in a bilinear term are frozen. To this end we identify a second variable $x_{(j)}$, $i < j$, with its current value $\widetilde{x_{(j)}}$ and consider the term

$$\mathfrak{A}_{(i,j)} \operatorname{vec}\left(x_{(i)} x_{(j)}^T\right) = \mathfrak{A}_{(i,j)} \operatorname{vec}\left(\widetilde{x_{(i)}}\,\widetilde{x_{(j)}}^T\right)$$

which immediately gives us an additional constant coefficient. We omit a formal definition of the coefficient map for the general case.

Change of variables. When a user defines a Frobenius norm $\|G(x,\theta)\|$ with G biaffine in x and θ and with θ belonging to an ellipsoidal perturbation set $\{\theta \mid \|P\theta + q\| \leqslant r\}$ with P invertible and $r > 0$, then we would like to reformulate this uncertain norm as $\left\|G\left(x, P^{-1}(r\zeta - q)\right)\right\|$ with ζ belonging to the unit ball $\{\zeta \mid \|\zeta\| \leqslant 1\}$ in order to meet the requirements of theorem 3.3. The change of variable $\theta \mapsto P^{-1}(r\zeta - q)$ goes beyond the capabilities of PICOS 2.0 which supports only the replacement of variables by other variables of same dimension, a functionality whose extension to biaffine expressions is trivial as it has no arithmetic aspect to it.

In a general setting, suppose that we want to replace each occurrence of the variable $x_{(i)} \in \mathbb{R}^{\alpha_i}$ in A with the affine vector expression

$$B := \sum_{k \in [l]} \mathfrak{B}_{(k)} y_{(k)} + \mathfrak{B}_{()} \in \mathbb{R}^{\alpha_i}$$

where $\mathfrak{B}_{(k)} \in \mathbb{R}^{\alpha_i \times \beta_k}$ is a coefficient matrix and $y_{(k)} \in \mathbb{R}^{\beta_k}$ is a fresh variable with $\beta_k \in \mathbb{Z}_{\geqslant 1}$ for all $k \in [l]$ with $l \in \mathbb{Z}_{\geqslant 1}$ and where $\mathfrak{B}_{()} \in \mathbb{R}^{\alpha_i}$ is a constant term. We write $A[x_{(i)} \mapsto B]$ short for the expression that is obtained from this replacement and we look to compute $\operatorname{Coefs}\left(A[x_{(i)} \mapsto B]\right)$. For case (6.1), we have

$$\begin{aligned} \mathfrak{A}_{(i)} x_{(i)} &= \mathfrak{A}_{(i)} \left(\sum_{k \in [l]} \mathfrak{B}_{(k)} y_{(k)} + \mathfrak{B}_{()} \right) \\ &= \sum_{k \in [l]} \left(\mathfrak{A}_{(i)} \mathfrak{B}_{(k)}\right) y_{(k)} + \left(\mathfrak{A}_{(i)} \mathfrak{B}_{()}\right) \end{aligned}$$

and obtain the partial coefficient map

$$\begin{aligned} M_1 := \; & \left\{ (y_{(k)}) \mapsto \mathfrak{A}_{(i)} \mathfrak{B}_{(k)} \mid k \in [l] \right\} \\ \cup \; & \left\{ () \mapsto \mathfrak{A}_{()} + \mathfrak{A}_{(i)} \mathfrak{B}_{()} \right\}. \end{aligned}$$

For case (6.2), it is

$$\begin{aligned}
&\mathfrak{A}_{(i,j)} \operatorname{vec}\left(x_{(i)} x_{(j)}^{\mathsf{T}}\right) \\
=\ & \mathfrak{A}_{(i,j)} \operatorname{vec}\left(\left(\sum_{k\in[l]} \mathfrak{B}_{(k)} y_{(k)} + \mathfrak{B}_{()}\right) x_{(j)}^{\mathsf{T}}\right) \\
=\ & \sum_{k\in[l]} \mathfrak{A}_{(i,j)} \operatorname{vec}\left(\mathfrak{B}_{(k)} y_{(k)} x_{(j)}^{\mathsf{T}}\right) + \mathfrak{A}_{(i,j)} \operatorname{vec}\left(\mathfrak{B}_{()} x_{(j)}^{\mathsf{T}}\right) \\
=\ & \sum_{k\in[l]} \left(\mathfrak{A}_{(i,j)} \left(I_{\alpha_j} \otimes \mathfrak{B}_{(k)}\right)\right) \operatorname{vec}\left(y_{(k)} x_{(j)}^{\mathsf{T}}\right) + \left(\mathfrak{A}_{(i,j)} \left(I_{\alpha_j} \otimes \mathfrak{B}_{()}\right)\right) x_{(j)}
\end{aligned}$$

which yields the additional coefficients

$$\begin{aligned}
M_2 := \ & \left\{(y_{(k)}, x_{(j)}) \mapsto \mathfrak{A}_{(i,j)} \left(I_{\alpha_j} \otimes \mathfrak{B}_{(k)}\right) \mid k \in [l], i < j\right\} \\
\cup \ & \left\{(x_{(j)}) \mapsto \mathfrak{A}_{(j)} + \mathfrak{A}_{(i,j)} \left(I_{\alpha_j} \otimes \mathfrak{B}_{()}\right) \mid i < j\right\}.
\end{aligned}$$

Similarly, case (6.3) resolves as

$$\begin{aligned}
&\mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} x_{(i)}^{\mathsf{T}}\right) \\
=\ & \mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} \left(\sum_{k\in[l]} \mathfrak{B}_{(k)} y_{(k)} + \mathfrak{B}_{()}\right)^{\mathsf{T}}\right) \\
=\ & \sum_{k\in[l]} \mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} \left(\mathfrak{B}_{(k)} y_{(k)}\right)^{\mathsf{T}}\right) + \mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} \mathfrak{B}_{()}^{\mathsf{T}}\right) \\
=\ & \sum_{k\in[l]} \mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} y_{(k)}^{\mathsf{T}} \mathfrak{B}_{(k)}^{\mathsf{T}}\right) + \mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} \mathfrak{B}_{()}^{\mathsf{T}}\right) \\
=\ & \sum_{k\in[l]} \left(\mathfrak{A}_{(h,i)} \left(\mathfrak{B}_{(k)} \otimes I_{\alpha_h}\right)\right) \operatorname{vec}\left(x_{(h)} y_{(k)}^{\mathsf{T}}\right) + \left(\mathfrak{A}_{(h,i)} \left(\mathfrak{B}_{()} \otimes I_{\alpha_h}\right)\right) x_{(h)}
\end{aligned}$$

and this gives us

$$\begin{aligned}
M_3 := \ & \left\{(x_{(h)}, y_{(k)}) \mapsto \mathfrak{A}_{(h,i)} \left(\mathfrak{B}_{(k)} \otimes I_{\alpha_h}\right) \mid k \in [l], h < i\right\} \\
\cup \ & \left\{(x_{(h)}) \mapsto \mathfrak{A}_{(h,i)} \left(\mathfrak{B}_{()} \otimes I_{\alpha_h}\right) \mid h < i\right\}.
\end{aligned}$$

Finally, we obtain

$$\begin{aligned}
\operatorname{Coefs}\left(A[x_{(i)} \mapsto B]\right) = \ & \left\{V \mapsto C \in \operatorname{Coefs}(A) \mid x_{(i)} \text{ not in } V\right\} \\
\cup \ & (M_1 \cup M_2 \cup M_3).
\end{aligned}$$

The extension to a replacement of multiple variables at once is straightforward when at most one of the variables of each bilinear term is replaced and when the replacing expressions have a pairwise disjoint set of variables. As we do not require such an extension in the context of optimization under uncertainty, it is implemented for this special case only.

Factoring out variables. Apart from a unit-ball description of the perturbation set, another requirement to apply theorem 3.3 is that the expression under the norm in the uncertain constraint is available in a standard form that separates the decision variables from the perturbation parameter. Assuming the identifiers introduced in section 3.3, we recall this form as $A(x)\eta+a(x)$ where η is the perturbation parameter and where A and a are affine functions in the decision vector x. In the following we describe the operation that recovers the affine expressions $A(x)$ and $a(x)$ from the internal representation of $A(x)\eta + a(x)$. To be consistent with the discussions of other operations, we return to this section's notation where A denotes the biaffine expression to operate on.

For the desired form to exist, we require A to be a vector. Let thus $n = 1$ so that A is an m-dimensional column vector. As trivially $\operatorname{reshape}_{m,1}(v) = v$ holds for all $v \in \mathbb{C}^m$, we can write A as

$$A = \sum_{i<j} \mathfrak{A}_{(i,j)} \operatorname{vec}\left(x_{(i)} x_{(j)}^T\right) + \sum_i \mathfrak{A}_{(i)} x_{(i)} + \mathfrak{A}_{()} \in \mathbb{C}^m.$$

Let now $x_{(i)} \in \mathbb{R}^{\alpha_i}$ be the variable to be "factored out" such that we can rewrite A as one term that is linear in $x_{(i)}$ and one that does not depend on $x_{(i)}$. More precisely, we seek new expressions $B \in \mathbb{C}^{m \times \alpha_i}$ and $C \in \mathbb{C}^m$ such that

$$A = B(x_{(k)} \mid k \neq i)x_{(i)} + C(x_{(k)} \mid k \neq i).$$

It is easy to see that C is given by

$$\begin{aligned} \operatorname{Coefs}(C) = \; & \left\{(x_k, x_l) \mapsto \mathfrak{A}_{(k,l)} \mid i \neq k < l \neq i\right\} \\ \cup \; & \left\{(x_k) \mapsto \mathfrak{A}_{(k)} \mid k \neq i\right\} \cup \left\{() \mapsto \mathfrak{A}_{()}\right\}. \end{aligned}$$

As B is constructed exactly from the terms in A that depend on $x_{(i)}$, we need to distinguish cases (6.1) to (6.3) once more. For case (6.1), factoring $x_{(i)}$ out of the linear term $\mathfrak{A}_{(i)}x_{(i)}$ is trivial and yields $\mathfrak{B}_{()} := \operatorname{vec}\left(\mathfrak{A}_{(i)}\right)$ representing the constant term $B_{()} := \operatorname{reshape}_{m,\alpha_i}\left(\mathfrak{B}_{()}\right)$ of B. We will reduce case (6.2) to case (6.3) and continue with the latter. Here the identity

$$\mathfrak{A}_{(h,i)} \operatorname{vec}\left(x_{(h)} x_{(i)}^T\right) = \mathfrak{A}_{(h,i)}(I_{\alpha_i} \otimes x_{(h)})x_{(i)}$$

gives us the linear term

$$B_{(h)}(x_{(h)}) := \mathfrak{A}_{(h,i)}(I_{\alpha_i} \otimes x_{(h)})$$

of B. To obtain its coefficient matrix $\mathfrak{B}_{(h)} \in \mathbb{C}^{m\alpha_i \times \alpha_h}$, we rearrange

$$\begin{aligned} & B_{(h)}(x_{(h)}) = \mathfrak{A}_{(h,i)}(I_{\alpha_i} \otimes x_{(h)}) \\ \iff \quad & \operatorname{reshape}_{m,\alpha_i}\left(\mathfrak{B}_{(h)}x_{(h)}\right) = \mathfrak{A}_{(h,i)}(I_{\alpha_i} \otimes x_{(h)}) \\ \iff \quad & \mathfrak{B}_{(h)}x_{(h)} = \operatorname{vec}\left(\mathfrak{A}_{(h,i)}(I_{\alpha_i} \otimes x_{(h)})\right) \\ \iff \quad & \mathfrak{B}_{(h)}x_{(h)} = (I_{\alpha_i} \otimes \mathfrak{A}_{(h,i)}) \operatorname{vec}\left(I_{\alpha_i} \otimes x_{(h)}\right) \\ \iff \quad & \mathfrak{B}_{(h)}x_{(h)} = (I_{\alpha_i} \otimes \mathfrak{A}_{(h,i)})Vx_{(h)} \\ \iff \quad & \mathfrak{B}_{(h)} = (I_{\alpha_i} \otimes \mathfrak{A}_{(h,i)})V \end{aligned}$$

where

$$V := \begin{bmatrix} E_{(\alpha_i,1/1,1)} \otimes I_{\alpha_h} \\ \vdots \\ E_{(\alpha_i,1/\alpha_i,1)} \otimes I_{\alpha_h} \end{bmatrix}.$$

Intuitively, V is a nested block column matrix of α_i outer blocks of size $\alpha_h\alpha_i \times \alpha_h$ where each outer block is itself a block column matrix of α_i inner blocks of size $\alpha_h \times \alpha_h$. The k-th inner block of the k-th outer block of V is the identity matrix and V is zero elsewhere. As $I_{\alpha_i} \otimes \mathfrak{A}$ is a block matrix of $\alpha_i \times \alpha_i$ blocks of size $m \times \alpha_h\alpha_i$, we can also write $\mathfrak{B}_{(h)}$ as a column block matrix with blocks

$$\begin{aligned} \left(\mathfrak{B}_{(h)}\right)_q &= \sum_{r=1}^{\alpha_i} \left(I_{\alpha_i} \otimes \mathfrak{A}_{(h,i)}\right)_{qr} V_r \\ &= \mathfrak{A}_{(h,i)} V_q \\ &= \mathfrak{A}_{(h,i)} (E_{(\alpha_i,1/q,1)} \otimes I_{\alpha_h}) \in \mathbb{C}^{m \times \alpha_h} \end{aligned}$$

for $q \in [\alpha_i]$. Observe that if we partition $\mathfrak{A}_{(h,i)}$ into a block row matrix with α_i blocks of size $m \times \alpha_h$, then $\mathfrak{A}_{(h,i)}(E_{(\alpha_i,1/q,1)} \otimes I_{\alpha_h})$ is the q-th block of $\mathfrak{A}_{(h,i)}$. It follows that $\mathfrak{B}_{(h)}$ is just a blockwise (or partial) transpose of $\mathfrak{A}_{(h,i)}$. We can implement this efficiently using slicing notation: If `A` is a CVXOPT block row matrix with `k` blocks of size `(m, n)`, then we can obtain its blockwise transposition writing

```
cvxopt.sparse([A[:, i*n:(i+1)*n] for i in range(k)]).
```

Lastly, we reduce case (6.3) to case (6.2). Multiplying the coefficient matrix $\mathfrak{A}_{(i,j)}$ with a commutation matrix, we can swap the variables that it refers to:

$$\begin{aligned} \mathfrak{A}_{(i,j)} \operatorname{vec}\left(x_{(i)}x_{(j)}^T\right) &= \mathfrak{A}_{(i,j)} K_{(\alpha_j,\alpha_i)} \operatorname{vec}\left(\left(x_{(i)}x_{(j)}^T\right)^T\right) \\ &= \mathfrak{A}_{(i,j)} K_{(\alpha_j,\alpha_i)} \operatorname{vec}\left(x_{(j)}x_{(i)}^T\right). \end{aligned}$$

This lets us obtain the coefficient matrix $\mathfrak{B}_{(j)} \in \mathbb{C}^{m\alpha_i \times \alpha_j}$ defining the linear term

$$B_{(j)}(x_{(j)}) = \operatorname{reshape}_{m,\alpha_i}\left(\mathfrak{B}_{(j)}x_{(j)}\right)$$

of B as the blockwise transposition of the $m \times \alpha_i\alpha_j$ block row matrix $\mathfrak{A}_{(i,j)}K_{(\alpha_j,\alpha_i)}$ partitioned in α_i blocks of size $m \times \alpha_h$, the same partition as in case (6.2).

6.3. Evaluation

The PICOS source code comes with a body of production test cases and an auxiliary script to run them under varying settings. As the implementation of biaffine expressions

is also used for affine expressions that are in term used with virtually every test case, a majority of their implementation is covered by existing production tests. In contrast, PICOS does not come with a proper unit testing framework that validates individual parts of the code.[12] As the implementation of such a framework is outside the scope of this work, all of the additional features that distinguish biaffine from affine expressions were tested locally as follows:

1. Create biaffine expressions of suitable shape, say matrices `A` and `B`,
2. compute the symbolic result `C` of some operation on `A` and `B`,
3. value all variables that occur in `A` and `B`,
4. perform the same operation numerically on `A.value` and `B.value` and
5. compare this numeric result with `C.value`.

For the implementation of the results of sections 3 and 4, a number of new production tests were implemented. These are executed every time a new commit is pushed to the PICOS repository on GitLab [39], so that future regressions can be detected. Unfortunately there is a disparity between robust optimization, where it is relatively easy to come up with toy applications where the exact solution can be found analytically, and distributionally robust optimization, where even small applications are comparably hard to solve exactly. We believe however that the likelihood of an implementation error is low as the extended algebraic modeling language of PICOS allows us to implement the mathematical results almost verbatim. In particular, we have added a new function `picos.block` that allows the definition of block matrices using nested lists that visually match the mathematical notation.

It is worth noting that this document itself serves as an additional test for the correctness of the implementation as all interactive code listings are executed and checked by Python's built-in `doctest` module when the document is compiled.

We do not perform a performance analysis as all time critical operations are outsourced to the lower level numeric libraries CVXOPT and NumPy as well as to the backend optimization solvers. The focus of PICOS is modeling power over speed so that users who want to minimize solution time at the cost of code readability and program portability are advised to use PICOS for prototyping and then port their code to use the low level interfaces to their preferred solver directly, reproducing the results presented herein in the solver's own modeling language.

7. Conclusion and outlook

The results included in sections 3 and 4 were selected from the intersection of mathematical models that are flexible enough to represent a wide range of applications on the

[12] Interactive code listings that appear in the API documentation of PICOS are however validated alongside the production tests. We make use of this where applicable to improve test coverage.

one hand with solution methods that are well suited for an implementation in a high level convex optimization library on the other hand. We believe that this careful choice and the extend of this work allow us to call PICOS not only a *conic* but also a *robust* optimization framework going forward.

As hard as coming to a stop may be, some interesting results and worthwhile features present themselves for future efforts. Concerning the field of robust optimization, it is unfortunate that linear matrix inequalities may only be robustified with respect to scenario uncertainty so far, which is only of limited use when dealing with high-dimensional data. A result found in Ben-Tal et al. [4, Theorem 8.2.3], where the uncertain matrix that is constrained to the positive semidefinite cone is required to admit a particular decomposition, might allow some SDP representable constraints and derived objective functions to be robustified with respect to ellipsoidal uncertainty. Another attractive result found therein [4, Proposition 6.4.1] paves the path from bounds on elliptically uncertain Euclidean norms to uncertain convex quadratic inequalities (CQIs). While the special case of uncertain squared norms should be easy to implement, support for arbitrary uncertain CQIs would require additional work in the spirit of section 6.2.

With distributionally robust optimization, a recent spike in research interest provides an opportunity for PICOS and similar software to spark a dialogue between theory and practice by facilitating the search for suited applications. We hope that our implementation of the recent theorem 4.15 will have such an effect. While we covered two of the most prominent loss functions in squared norms and piecewise linear functions, we could only touch upon the degrees of freedom that open up with respect to modeling the distributional ambiguity set. For Wasserstein DRO a natural extension of the presented formulation would be to generalize the norm used in the definition of the Wasserstein distance, which enters as the associated dual norm in theorem 4.16. For instances of linearly separable norms it is known that Wasserstein DRO is equivalent to established regularization techniques [10, 41], so that one may hope to find novel applications in statistical learning for such an extension. With respect to moment-ambiguity, we withheld a recipe by Delage and Ye [13] for building a reasonable ambiguity set from data samples. An implementation thereof and of other means to determine suitable hyperparameters in PICOS could further help fulfill the library's promise of accessibility.

We note that our source code contribution was made with future endeavors in mind: The new biaffine expressions are not limited to uncertain affine expressions and could be used outside the scope of robust optimization to represent other means of parameterized data and associated methods. Similarly, the code that distinguishes between robust and distributionally robust optimization leaves room for extension towards the classical field of stochastic programming, which we have not paid much attention to in this thesis but which would complement the provided toolbox nicely.

Finally, we want to express our hope that the contributed software will be useful to the reader and help the advance of free open source software in research and teaching.

A. Formal scope

Commits. The practical part of this thesis comprises over 100 commits to PICOS' public git repository [39]. A clone of the repository created shortly after the release of PICOS 2.1, more precisely at version 2.1.1 that contains an installation related fix, is found on the compact disc attached to all hard copies of the thesis. An up-to-date repository can be obtained from GitLab with the following shell command:

```
git clone https://gitlab.com/picos-api/picos.git
```

To list the commits that constitute the practical part, one may execute the following command from within the repository:

```
git log --no-merges last-before-robust..v2.1
```

The topic of this thesis was proposed by its author in accordance with the examination regulations. Early programming experiments that led to this proposal have contributed code changes to a small number of commits in the above range. The following commits are affected to some extend by work that predates the writing period:

Commit	Description
`b3d9f07b`	Augment conic sets and constraints.
`3f056f68`	Implement a BiaffineExpression base class for ComplexAffineExpression.
`42a8b960`	Add a DecisionVariable base class.
`fa2032d3`	Start implementing a representation for uncertain data.

We remark that `3f056f68` does not implement any of the products or support operations discussed in sections 6.2.2 and 6.2.3. The commit merely moves code out of the `ComplexAffineExpression` class into a new source file and establishes the coefficient notation of section 6.2.1.

New files. This work adds close to 40 new source files containing more than 10,000 lines of code to PICOS. Below we list only a selection of important new files along with the functionality that they implement (an asterisk is a placeholder for multiple files):

New file / Implements

`picos/constraints/uncertain/ucon_*.py`
: The results of sections 3 and 4. Each theorem corresponds to a robust constraint class that implements a finite conic conversion recipe.

`picos/expressions/cone.py`
: A new base class for all cones that enables scenario-robust optimization.

`picos/expressions/cone_product.py`
: A product cone class used with the conic uncertainty set of section 3.

New file / Implements
`picos/expressions/exp_biaffine.py` The biaffine expressions discussed in section 6.2.
`picos/expressions/exp_extremum.py` A new base class for pointwise extremums, used in particular for the random convex or concave piecewise linear functions of sections 4.1.2 and 4.2.2.
`picos/expressions/mutable.py` A new base class common to decision variables and perturbation parameters.
`picos/expressions/samples.py` The auxiliary `Samples` class introduced in section 5.2.4.
`picos/expressions/set_ellipsoid.py` An `Ellipsoid` class used internally to represent ellipsoidal uncertainty sets and by applications for setting the sample space of a moment ambiguity set.
`picos/expressions/uncertain/pert_*.py` Representations of the perturbation universes discussed in section 5.2.
`picos/expressions/uncertain/uexp_*.py` The uncertain expression types discussed throughout this document, that are affine expressions, norms, squared norms and piecewise linear functions.
`tests/ptest_robust_*.py` A collection of production test cases for the new features.

Modified files. In addition to new files, more than 60 existing source files have been modified for a total of around 3,000 added and 3,500 deleted lines of code (as reported by `git diff`). Again we list only the most relevant changes:

Modified file / Changed
`picos/constraints/con_soc.py` Added a method to pose the constraint as membership in a unit ball. This is used for converting from a `ConicPerturbationSet` to a `UnitBallPerturabtionSet`.
`picos/expressions/algebra.py` Added `picos.block` (used internally for theorems 3.3, 4.3, 4.10, 4.15 and 4.16) and augmented `picos.min` and `picos.max` for defining pointwise extremums.
`picos/expressions/exp_affine.py` Affine expressions are now a special case of biaffine ones.
`picos/expressions/expression.py` Partially extended the results of section 6.2.3 to all expression types.
`picos/modeling/problem.py` Added a method to put problems into conic form that is used internally by the method `ConicPerturbationSet.compile` discussed in section 5.2.2.

References

[1] M. Abadi et al. *TensorFlow: Large-scale machine learning on heterogeneous distributed systems*. 2016. arXiv: 1603.04467.

[2] M. S. Andersen, J. Dahl, and L. Vandenberghe. *CVXOPT: A Python package for convex optimization*. 2020. URL: https://cvxopt.org (visited on 2020-05-07).

[3] I. Averbakh and Y.-B. Zhao. "Explicit reformulations for robust optimization problems with general uncertainty sets". In: *SIAM Journal on Optimization* 18.4 (2008), pp. 1436–1466.

[4] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009.

[5] A. Ben-Tal and A. Nemirovski. "Robust convex optimization". In: *Mathematics of Operations Research* 23.4 (1998), pp. 769–805.

[6] A. Ben-Tal and A. Nemirovski. "Robust solutions of linear programming problems contaminated with uncertain data". In: *Mathematical Programming* 88.3 (2000), pp. 411–424.

[7] A. Ben-Tal and A. Nemirovski. "Robust solutions of uncertain linear programs". In: *Operations Research Letters* 25.1 (1999), pp. 1–13.

[8] A. Ben-Tal and A. Nemirovski. "Robust truss topology design via semidefinite programming". In: *SIAM Journal on Optimization* 7.4 (1997), pp. 991–1016.

[9] A. Ben-Tal and A. Nemirovski. "Selected topics in robust convex optimization". In: *Mathematical Programming* 112.1 (2008), pp. 125–158.

[10] J. Blanchet, Y. Kang, and K. Murthy. "Robust Wasserstein profile inference and applications to machine learning". In: *Journal of Applied Probability* 56.3 (2019), pp. 830–857.

[11] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[12] R. Chen and I. C. Paschalidis. "A robust learning approach for regression models based on distributionally robust optimization". In: *Journal of Machine Learning Research* 19 (2018), 13:1–13:48.

[13] E. Delage and Y. Ye. "Distributionally robust optimization under moment uncertainty with application to data-driven problems". In: *Operations Research* 58.3 (2010), pp. 595–612.

[14] K. Derinkuyu and M. Ç. Pınar. "On the S-procedure and some variants". In: *Mathematical Methods of Operations Research* 64.1 (2006), pp. 55–77.

[15] L. L. Dines. "On the mapping of quadratic forms". In: *Bulletin of the American Mathematical Society* 47.6 (1941), pp. 494–498.

[16] L. Ding, S. Ahmed, and A. Shapiro. "A Python package for multi-stage stochastic programming". In: *Optimization Online* (2019).

[17] I. Dunning, J. Huchette, and M. Lubin. "JuMP: A modeling language for mathematical optimization". In: *SIAM Review* 59.2 (2017), pp. 295–320.

[18] I. R. Dunning. "Advances in robust and adaptive optimization: Algorithms, software, and insights". PhD thesis. Massachusetts Institute of Technology, 2016.

[19] L. El Ghaoui and H. Lebret. "Robust solutions to least-squares problems with uncertain data". In: *SIAM Journal on Matrix Analysis and Applications* 18.4 (1997), pp. 1035–1064.

[20] P. Esfahani Mohajerin and D. Kuhn. "Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations." In: *Mathematical Programming* 171.1-2 (2018), pp. 115–166.

[21] J. Goh and M. Sim. "Robust optimization made easy with ROME". In: *Operations Research* 59.4 (2011), pp. 973–985.

[22] W. E. Hart, J.-P. Watson, and D. L. Woodruff. "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.

[23] JetBrains s.r.o. *Python developers survey 2018 results*. 2019. URL: https://www.jetbrains.com/research/python-developers-survey-2018/ (visited on 2020-05-06).

[24] L. V. Kantorovich and G. S. Rubinstein. "On a space of completely additive functions". In: *Vestnik Leningradskogo Universiteta* 13.7 (1958). In Russian, pp. 52–59.

[25] S. Karabuk. *APLEpy: An open source algebraic programming language extension in Python*. 2005.

[26] R. H. Koning, H. Neudecker, and T. Wansbeek. "Block Kronecker products and the vecb operator". In: *Linear Algebra and its Applications* 149 (1991), pp. 165–184.

[27] H. Konno. "Piecewise linear risk function and portfolio optimization". In: *Journal of the Operations Research Society of Japan* 33.2 (1990), pp. 139–156.

[28] D. Kuhn, P. Esfahani Mohajerin, V. A. Nguyen, and S. Shafieezadeh-Abadeh. "Wasserstein distributionally robust optimization: Theory and applications in machine learning". In: *Operations Research & Management Science in the Age of Analytics*. INFORMS, 2019, pp. 130–166.

[29] J. Löfberg. "Automatic robust convex programming". In: *Optimization Methods and Software* 27.1 (2012), pp. 115–129.

[30] J. Löfberg. "YALMIP: A toolbox for modeling and optimization in MATLAB". In: *2004 IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers. New Orleans, LA, USA, 2004, pp. 284–289.

[31] S. Mehrotra and H. Zhang. "Models and algorithms for distributionally robust least squares problems." In: *Mathematical Programming* 146.1-2 (2014), pp. 123–141.

[32] MOSEK ApS. *MOSEK Optimizer API for Python 9.2.29*. 2020.

[33] V. A. Nguyen, S. Shafieezadeh-Abadeh, D. Filipovic, and D. Kuhn. "Distributionally robust risk measures with structured Wasserstein ambiguity sets". Working paper. 2020.

[34] F. Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[35] H. Rahimian and S. Mehrotra. *Distributionally robust optimization: A review*. 2019. arXiv: 1908.05659.

[36] M. Roelofs and J. Bisschop. *AIMMS: The Language Reference*. AIMMS B.V., 2020.

[37] W. Römisch and R. Schultz. "Stability analysis for stochastic programs". In: *Annals of Operations Research* 30.1-4 (1991), pp. 241–266.

[38] G. Sagnol and M. Stahlberg. *PICOS: A Python interface to conic optimization solvers*. 2020. URL: https://picos-api.gitlab.io/picos/ (visited on 2020-05-06).

[39] G. Sagnol and M. Stahlberg. *The PICOS repository on GitLab*. 2020. URL: https://gitlab.com/picos-api/picos (visited on 2020-11-27).

[40] H. E. Scarf. "A min-max solution of an inventory problem". In: *Studies in the Mathematical Theory of Inventory and Production* 10 (1958), pp. 201–209.

[41] S. Shafieezadeh-Abadeh, D. Kuhn, and P. Esfahani Mohajerin. "Regularization via mass transportation". In: *Journal of Machine Learning Research* 20.103 (2019), pp. 1–68.

[42] A. Shapiro. "On duality theory of conic linear problems". In: *Semi-Infinite Programming*. Ed. by M. Á. Goberna and M. A. López. Vol. 57. Nonconvex Optimization and Its Applications. Boston, MA, USA: Springer, 2001, pp. 135–165.

[43] C.-K. Sim and G. Zhao. "A note on treating a second order cone program as a special case of a semidefinite program". In: *Mathematical Programming* 102.3 (2005), pp. 609–613.

[44] J. E. Smith and R. L. Winkler. "The optimizer's curse: Skepticism and postdecision surprise in decision analysis". In: *Management Science* 52.3 (2006), pp. 311–322.

[45] A. L. Soyster. “Convex programming with set-inclusive constraints and applications to inexact linear programming”. In: *Operations Research* 21.5 (1973), pp. 1154–1157.

[46] TIOBE Software BV. *The Python programming language*. 2020. URL: https://www.tiobe.com/tiobe-index/python/ (visited on 2020-11-25).

[47] P. Virtanen et al. “SciPy 1.0: Fundamental algorithms for scientific computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272.

[48] J.-P. Watson, D. L. Woodruff, and W. E. Hart. “PySP: Modeling and solving stochastic programs in Python”. In: *Mathematical Programming Computation* 4.2 (2012), pp. 109–149.

[49] J. Zhen, D. Kuhn, and W. Wiesemann. “Distributionally robust nonlinear optimization”. Working paper. 2020.