# cvxRiskOpt: A Risk-Based Optimization Tool Based on CVXPY

Sleiman Safaoui, *Member, IEEE*, and Tyler H. Summers, *Member, IEEE*

***Abstract*—We introduce cvxRiskOpt (convex Risk-based Optimization): a Python package built on top of CVXPY for the rapid prototyping of convex risk-based optimization problems and generating embeddable C code using CVXPYgen. Our package provides high-level functions to handle several risk-based optimization problems and constraints. These functions reformulate problems and constraints involving random variables and uncertainty into deterministic convex counterparts. The output is either a CVXPY Problem instance or CVXPY constraints that users can directly add to their CVXPY Problem instance. Accordingly, our package can use CVXPYgen to generate C code resulting in custom embeddable risk-based optimization problems. cvxRiskOpt is available at https://tsummerslab.github.io/cvxRiskOpt/.**

***Index Terms*—Computational methods, uncertain systems, optimization.**

## I. Introduction

MODELING languages and parser solvers have become standard tools for the rapid prototyping of optimization problems. These tools, such as CVXPY [1], Yalmip [2], CasADi [3], and many others, allow users to express optimization problems in a natural math-like syntax instead of restrictive standard forms that are usually required by optimization problem solvers such as ECOS [4], CLARABEL [5], SCS [6], or others. This makes coding and solving prototype optimization models significantly faster. In this letter, we focus on tools using the Python programming language such as CVXPY.

Rapid prototyping tools are somewhat less well-developed for stochastic, robust, and distributionally robust optimization problems (SO, RO, and DRO respectively), which explicitly incorporate uncertainties in the problem data. We refer to these problems collectively as risk-based optimization (RBO). Generally, a (parametrized) RBO problem can be written as:

$$\begin{aligned} \min_{x} \quad & \sup_{\mathbb{P}\in\mathcal{P}} \mathcal{R}_0(f_0(x,\theta,\xi)) \\ \text{s.t.} \quad & \sup_{\mathbb{P}\in\mathcal{P}} \mathcal{R}_i(f_i(x,\theta,\xi)) \le b_i, \quad i=1,\dots,p \end{aligned} \tag{1}$$

where $x$ is a vector of optimization variables, $\theta$ is a vector of parameters whose values can change but are constant when solving the problem, and $\xi$ is a random vector. The $b_*$ terms are constants and $\mathcal{R}_*$ represent *risk metrics* [7], [8], [9] that measure the risk associated with the nominal objective and constraint functions $f_*$ that are random variables themselves since they depend on $\xi$. To evaluate the risk associated with $f_*$, their distributions must be known. However, the underlying uncertainty distributions are never known. As such, an *ambiguity set* $\mathcal{P}$ is used to represent and reason about all distributions that have certain characteristics. This definition for (1) based on DRO includes SO and RO as special cases. For example, if $\mathcal{P}=\{\mathbb{P}\}$, i.e., the distribution is known to be $\mathbb{P}$, then a risk metric $\mathcal{R}$ is evaluated with respect to $\mathbb{P}$.

RBO problems are generally challenging to solve and may not be tractable. However, for some choices of ambiguity sets, risk metrics, and functions $f_*$, (1) can be transformed into an equivalent deterministic optimization problem [9], [10], [11], [12]. In many cases, the transformations can be tedious, analogous to when transforming deterministic optimization problems into standard solver forms. Some modeling and parsing tools which have addressed certain classes of RBO problems include cvxstoc [13] and MSPPy [14] for SO, PYomo [15] with its extensions PySP for SO and ROmodel [16] and PyROS [17] for RO, and PICOS [18] and RSOME [19] for RO and DRO problems. Table I summarizes the types of problems considered by these tools.

A useful extension to these tools is the automatic generation of custom C code for embedded system applications and enhanced solve times [20]. This allows users to write optimization problems in mathematical terms and automatically generate optimized C code. Unlike the general-purpose optimization problem modeling tools mentioned earlier, there are much fewer tools that support code generation. Examples include TinyMPC [21], CVXGEN [20], and CVXPYgen [22]. TinyMPC is a recent package for generating embeddable code specifically for deterministic MPC problems. CVXGEN is an older tool for generating C code for linear and quadratic programs that is being superseded by CVXPYgen which supports a wide range of CVXPY Problem instances including

Manuscript received 31 May 2024; revised 13 August 2024; accepted 15 August 2024. Date of publication 30 August 2024; date of current version 16 September 2024. This work was supported in part by the United States Air Force Office of Scientific Research under Grant FA9550-23-1-0424, and in part by the National Science Foundation under Grant ECCS-2047040. Recommended by Senior Editor M. Elena Valcher. *(Corresponding author: Sleiman Safaoui.)*

Sleiman Safaoui was with the Department of Mechanical Engineering, The University of Texas at Dallas, Richardson, TX 75248 USA (e-mail: snsafaoui@gmail.com).

Tyler H. Summers is with the Department of Mechanical Engineering, The University of Texas at Dallas, Richardson, TX 75248 USA (e-mail: tyler.summers@utdallas.edu).

Digital Object Identifier 10.1109/LCSYS.2024.3452194

TABLE I
PYTHON OPTIMIZATION PACKAGES COMPARISON

| Package | SO | RO | DRO | Code gen. |
|---|---|---|---|---|
| | | Pyomo | | |
| PySP | ✓ | ✗ | ✗ | ✗ |
| mpi-sppy | ✓ | ✗ | ✗ | ✗ |
| PyROS | ✗ | ✓ | ✗ | ✗ |
| ROmodel | ✗ | ✓ | ✗ | ✗ |
| | | PICOS | | |
| PICOS | ✗ | ✓ | ✓ | ✗ |
| | | RSOME | | |
| RSOME | ✓ | ✓ | ✓ | ✗ |
| | | CVXPY | | |
| cvxstoch | ✓ | ✗ | ✗ | ✗ |
| CVXPYgen | ✗ | ✗ | ✗ | ✓ |
| **cvxRiskOpt** | ✓ | ✗ | ✓ | ✓(CVXPYgen) |

parameterized second-order cone programs. However, these tools do not have native support for RBO problems.

To streamline prototyping RBO problems, it is desirable to have a single tool that allows users to encode them using high-level functions and easily generate C code.

**Contributions:** We present cvxRiskOpt, a Python package for CVXPY to address this gap. Our package returns CVXPY Problem instances and CVXPY constraints for some RBO problems and risk constraints, making it easier to formulate RBO problems. Since cvxRiskOpt extends the CVXPY ecosystem, we can utilize CVXPYgen to generate C code as well. The main contributions of cvxRiskOpt are:

- supporting rapid prototyping with certain SO and DRO problems, including
  - variations of chance-constrained linear programs with moment-based ambiguity sets
  - variations of worst-case expectation problem with Wasserstein-based ambiguity sets
- integrating the deterministic reformulations of the above problems into CVXPY Problem instances
- utilizing CVXPYgen to generate C code.

To the best of our knowledge, none of the tools that overlap with cvxRiskOpt's utility enable easily encoding RBO problems *and* automatic code generation, simultaneously. RSOME supports many RO and DRO problems; however, it has a few disadvantages and limitations. The modeling of ambiguity sets is based on Event-wise Ambiguity Sets [23]. While [23] provides examples of formulating several ambiguity sets as event-wise ambiguity sets (e.g., Generalized-Moment Ambiguity Sets and Wasserstein Ambiguity Sets), the formulation is complicated, which poses an additional challenge for non-experts who may simply want to experiment with some standard formulations. Compared to RSOME, PICOS provides a more user-friendly interface to a few types of RO and DRO problems that it supports. However, neither RSOME nor PICOS supports automatic code generation. While cvxRiskOpt is currently more limited in its support for RBO problems, it provides a user-friendly and beginner-friendly interface for rapid prototyping with some widely-used risk-based problems and constraints, it extends the features of CVXPY, and it benefits from CVXPYgen for C code generation which can be used for embedded applications or faster solve times [22].

**Notation**: Let $\langle x, y\rangle = x^T y$ denote the inner product, $[a : b]$ denote set of natural numbers from $a$ to $b$ (inclusive), and $\|\cdot\|_\star$ denote the dual norm.

## II. MODELING RBO PROBLEMS

Our package, cvxRiskOpt, builds on top of CVXPY and provides tools to convert some convex RBO problems and constraints into CVXPY Problem classes and constraints. Upon obtaining a CVXPY Problem, its parameters can be set and the class's methods can be used to solve the problem and get an optimal solution using the CVXPY API [24]. For linear, quadratic, and second order cone Problem instances, C code can be generated using CVXPYgen.

We consider some prototypical convex RBO problems. Currently, cvxRiskOpt supports reformulating the worst-case expectation with a Wasserstein-based ambiguity set [25], into a CVXPY Problem instance (Section III) and reformulating some chance constraint linear programs (CCLPs), including the distributionally robust value-at-risk (DR-VaR) with a moment-based ambiguity set, into CVXPY constraints (Section IV).

## III. WASSERSTEIN WORST-CASE EXPECTATION PROBLEMS

Our proposed package, cvxRiskOpt, currently supports two special classes of the general RBO problem (1). We begin with Wasserstein worst-case expectation (WCE) problems.

Consider a random variable $\xi$ for which we have $N$ samples $\hat{\xi}_i,\ i \in [1:N]$. Let the empirical distribution formed by these samples be $\hat{\mathbb{P}}_N$. As the true distribution of $\xi$ is unknown, consider a Wasserstein-based ambiguity set to be robust against, i.e., the set of all distributions in the Wasserstein ball around the empirical distribution $\mathbb{B}_\epsilon(\hat{\mathbb{P}}_N)$ per [25, (6)]. This worst case expectation (WCE) problem [25, (10)] with respect to the Wasserstein-based ambiguity set is given by:

$$\text{WCE} \quad \sup_{\mathbb{P}\in\mathbb{B}_\epsilon\left(\hat{\mathbb{P}}_N\right)} \quad \mathbb{E}^{\mathbb{P}}[\ell(\xi)] \tag{2}$$

where $\ell(\xi)$ is a decision-independent loss function. Note that (2) may appear in (1) in the objective function (e.g., Section V-A) or as the left-hand-side of the inequality constraints (e.g., [26, Def. 3]). For a piecewise affine loss function $\ell(\xi)$, (2) can be reformulated into a computationally tractable convex optimization problem [25]. In particular, here we focus on the case where $\ell(\xi)$ is a piecewise affine loss function consisting of a max of affine terms.

Suppose that the uncertainty set is a polytope characterized by $C\xi \le d$ where $C, d$ are a matrix and vector of appropriate dimensions. Let $\ell(\xi, x, y) = \max_{k\le K} \ell_k(\xi, x, y)$ with

$$\ell_k(\xi, x, y) := \langle a_k(x), \xi\rangle + b_k(y)$$

where $a_k(x)$ is a vector that may depend linearly on $x$, and $b_k(y) = \langle b_k, y\rangle + c_k$ is a scalar and may be linear in some decision variable $y$. Note that (2) is not evaluated with respect to $x, y$. Instead, $x, y$ can be optimized for by another optimization problem (e.g., $\min_{x,y}$ WCE).

From [25, (15a)], the WCE problem (2) where $\ell$ is a max of affine terms is equivalent to:

$$\begin{aligned} \inf_{\lambda, s_i, \gamma_{i,k}} \quad & \lambda\epsilon + \frac{1}{N}\sum_{i=1}^{N} s_i \\ \text{s.t.} \quad & \langle a_k(x), \hat{\xi}_i\rangle + \langle b_k, y\rangle + c_k + \langle \gamma_{i,k}, d - C\hat{\xi}_i\rangle \le s_i \\ & \left\| C^\top \gamma_{i,k} - a_k(x)\right\|_\star \le \lambda, \qquad \gamma_{i,k} \ge 0 \end{aligned} \tag{3}$$

where the constraints are $\forall i \in [1:N],\ \forall k \in [1:K]$.

To encode the reformulation (3), cvxRiskOpt defines a `WassWCEMaxAffine` class, which inherits from CVXPY Problem. Then, the WCE term (2) can be encoded as:

`prob = WassWCEMaxAffine(N, a_k, b_k, C, d, used_norm).`

Users only need to specify the number of samples `N` $= N$ and lists containing $a_k(x), b_k(y)$ terms (`a_k, b_k`). The support set parameters `C` $=C$, `d` $=d$ and the norm (`used_norm` $\in \{1, 2, \infty\}$) are optional parameters. The `WassWCEMaxAffine` class automatically creates CVXPY parameters `eps`, `samp` for the Wasserstein ball radius $\epsilon$ and the data samples $\xi$, respectively. The parameter values can be set by the user with the desired radius and the data either using the CVXPY syntax (i.e., param.value = val) or through the cvxRiskOpt function:

`update_wass_wce_params(prob, eps, samp).`

We also implement some special cases of (3) that we also provide cvxRiskOpt classes for.

### A. Wasserstein Distributionally Robust Expectation

The Distributionally Robust Expectation (DR-$\mathbb{E}$) is implemented as a special case of (3) where the loss function only has a single term: $\ell(\xi, x, y) = \ell_1(\xi, x, y)$. In cvxRiskOpt, we provide the following class that inherits from `WassWCEMaxAffine`:

`WassDRExpectation(N, a, b, C, d, used_norm)`

which implements $\sup_{\mathbb{P}\in\mathbb{B}_\epsilon(\hat{\mathbb{P}}_N)} \mathbb{E}^{\mathbb{P}}[\langle a(x), \xi\rangle + b(y)]$.

The class also supports some problem arithmetic between problems that use the same Wasserstein ball radius $\epsilon$ and same samples $\xi$.

Adding a `WassDRExpectation` and a `WassWCEMax Affine` problem returns an instance of `WassWCEMaxAffine`.

**Subtraction.** Subtracting `WassDRExpectation` problems is similar to their addition by the linearity of expectation.

**Multiplication.** Scalar multiplication is allowed with a `WassDRExpectation` problem.

**Division.** Scalar division of `WassDRExpectation` by a non-negative scalar $\rho$ is defined as a multiplication with $1/\rho$.

### B. WassersteinDistributionally Robust Conditional Value at Risk

The Distributionally Robust Conditional Value at Risk (DR-CVaR) of a scalar random variable $\xi$ is a special case of (3) as seen when using the formal definition of CVaR [27]: $\text{CVaR}^{\mathbb{P}}_\alpha(\xi) := \inf_{\tau\in\mathbb{R}} \mathbb{E}^{\mathbb{P}}[\xi + \frac{1}{\alpha}\max\{\xi - \tau, 0\}]$. In cvxRiskOpt, we provide the class `WassDRCVaR` which implements DR-$\text{CVaR}^{\epsilon}_{\alpha}[\langle a(x), \xi\rangle + b(y)] = \sup_{\mathbb{P}\in\mathbb{B}_\epsilon(\hat{\mathbb{P}}_N)} \text{CVaR}^{\mathbb{P}}_\alpha[\langle a(x), \xi\rangle + b(y)]$ where $\alpha$ represents the $\alpha \times 100\%$ worst cases:

`WassDRCVaR(N, m, a, b, a_k, b_k,alpha, C, d, used_norm)`

which can be used in two ways. The first way is:

`WassDRCVaR(N, m, a, b)`

where users provide the number of samples $N$, sample dimension $m$, $a(x)$, $b(y)$ terms, and optionally the support, norm used, and CVaR level $(C, d, \text{used\_norm}, \alpha)$ to set up an instance of this DR-CVaR problem. The second way is:

`WassDRCVaR(N, m, a_k=a_k, b_k=b_k)` (4)

where users provide the lists `a_k`$= a_k(x)$, `b_k`$= b_k(y)$ (here $k = 2$) analogously to the `WassDRExpectation` function.

The class also supports some problem arithmetic, particularly multiplication and division with a positive scalar.

## IV. Chance Constrained Linear Programs

In this section we consider two forms of the generalized linear chance constraint, using a value-at-risk (VaR) risk metric, based on [28, (6)]:

$$\mathbb{P}\big(a(x)^T\xi_1 + b(y) + \xi_2 \le 0\big) \ge 1 - \epsilon \tag{5a}$$

$$\mathbb{P}\big(a^T\xi_1(z) + b(y) + \xi_2 \le 0\big) \ge 1 - \epsilon \tag{5b}$$

where vector $a(x)$ is linear in a decision variable $x$, scalar $b(y)$ is affine in the decision variable $y$ and $\xi_1(z)$, $\xi_2$ are a random vector and random variable respectively and $z$ is a decision variable. Note that the two forms are similar with the distinction being that only $a$ or $\xi_1$ may depend on a decision variable, but not both since otherwise the problem would be bilinear in $x, z$ and would no longer be convex. For ease of exposition, below we will use the common formulation $\mathbb{P}(a(x)^T\xi_1(z) + b(y) + \xi_2 \le 0) \ge 1 - \epsilon$ while assuming that $\varphi(x, y, z) := a(x)^T\xi_1(z) + b(y) + \xi_2$ is linear. Here, $\epsilon \in (0, 0.5]$. Problems of the form (5) and

$$\inf_{[\xi_1^T, \xi_2]\sim\mathbb{P}\in\mathcal{P}} \mathbb{P}(\varphi(x, y, z) \le 0) \ge 1 - \epsilon$$

can appear as inequality constraints in (1). To aid with incorporating them, cvxRiskOpt provides functions that return deterministic linear constraints based on the reformulations in [28]. The resulting constraints can be used as constraints in a CVXPY problem. The CVXPY expression can be extracted from the CVXPY Inequality constraint (cstr.expr) to use in the objective function.

Let $\xi(z) = \begin{bmatrix}\xi_1(z)^T & \xi_2\end{bmatrix}^T$ and $\tilde{a}(x) = \begin{bmatrix}a(x)^T & 1\end{bmatrix}^T$ so that $\varphi(x, y, z) = \tilde{a}(x)^T\xi(z) + b(y)$. Let

$$\hat{\xi}(z) := \mathbb{E}[\xi(z)] := \begin{bmatrix}\hat{\xi}_1(z)\\ \hat{\xi}_2\end{bmatrix}$$

$$\Gamma := \text{Cov}(\xi) := \begin{bmatrix}\Gamma_{11} & \gamma_{12}\\ \gamma_{12}^T & \gamma_{22}\end{bmatrix} \succcurlyeq 0$$

where $\Gamma$ is independent of $z$. We have:

$$\hat{\varphi}(x, y, z) := \mathbb{E}\big[\varphi(x, y, z)\big] = \tilde{a}(x)^T\hat{\xi}(z) + b(y)$$

$$\sigma^2(x) := \text{Cov}(\varphi(x, y, z)) = \tilde{a}(x)^T\Gamma\tilde{a}(x).$$

### A. CCLP Under Gaussian Distribution

We implement the reformulation from [28, Sec. 2] where it is shown that

$$\text{CCLP (5)} \iff \kappa\sigma(x) + \hat{\varphi}(x, y, z) \le 0 \tag{6}$$

where $\kappa$ is a variable that depends on the distribution. Particularly, for Gaussian distributions, $\kappa = \Psi^{-1}_{\mathcal{N}}(1-\epsilon)$ where $\Psi^{-1}_{\mathcal{N}}$ is the inverse cumulative distribution function of the

standard Gaussian random variable [28, Sec. 2.1]. To generate the equivalent deterministic constraint, we define the function:

```
cclp_gauss(eps, a, b, xi1_hat, xi2_hat, gam11, gam12, gam22)
```

where `xi1_hat` $= \hat{\xi}_1$, `xi2_hat` $= \hat{\xi}_2$, `gam11` $= \Gamma_{11}$, `gam12` $= \gamma_{12}$, `gam22` $= \gamma_{22}$.

### B. Distributionally Robust (DR) CCLP

We implement the DR-CCLP reformulation from [28, Sec. 3] where it is shown that:

$$\inf_{[\xi_1^T,\xi_2]\sim\mathbb{P}\in\mathcal{P}} \mathbb{P}(\varphi(x,y,z)\le 0) \ge 1-\epsilon$$
$$\iff \kappa\sigma(x)+\hat{\varphi}(x,y,z)\le 0$$

for some choices of $\mathcal{P}$, the ambiguity set, and $\kappa$ is a variable that depends on the characteristics of $\mathcal{P}$.

In particular, consider a moment-based ambiguity set with known mean and covariance $\mathcal{P} := (\hat{\xi}, \Gamma)$; i.e., the set of all distributions with a given mean and covariance. In this case, $\kappa = \sqrt{\frac{1-\epsilon}{\epsilon}}$ [28, Sec. 3.1]. If $\mathcal{P}$ only includes distributions that are centrally symmetric [28, Definition 3.1], then $\kappa = \sqrt{\frac{1}{2\epsilon}}$. Both of these reformulations are implemented through:

```
cclp_dro_mean_cov(eps, a, b, xi1_hat, xi2_hat,
        gam11, gam12, gam22, cent_symm)
```

where `cent_symm` $= 1$ indicates that $\mathcal{P}$ only includes centrally symmetric distributions.

## V. Examples

Here, we provide some examples of using cvxRiskOpt with application to portfolio optimization, model predictive control (MPC), and moving horizon estimation (MHE). Additional examples, can be found under the "Examples" section in the package documentation: https://tsummerslab.github.io/cvxRiskOpt/. The package source code can be found at: https://github.com/TSummersLab/cvxRiskOpt. Numerical results were obtained using a MacBook Pro with an M3 Pro 12-core CPU and 18GB of RAM.

### A. Distributionally Robust Portfolio Optimization

Consider the distributionally robust portfolio optimization problem from [25, Section 7.1] with yearly returns $\xi$ of $m$ assets and portfolio percentage weights vector $x$ with $x_i \ge 0$, $\sum_i x_i = 1$:

$$\sup_{\mathbb{P}\in\mathbb{B}_\epsilon\left(\hat{\mathbb{P}}_N\right)} \inf_{x\in\mathbb{X}} \left\{\mathbb{E}^{\mathbb{P}}[-\langle x,\xi\rangle] + \rho\,\mathrm{CVaR}_\alpha^{\mathbb{P}}(-\langle x,\xi\rangle)\right\}. \tag{7}$$

This problem can be encoded as a worst-case expectation optimization problem where the objective is a max of affine terms as shown in [25, (27)] where $i \in [1:N]$ and $k \in \{1,2\}$:

$$\inf_{\substack{x,\tau,\lambda,\\ s_i,\gamma_{i,k}}} \lambda\epsilon + \frac{1}{N}\sum_{i=1}^{N} s_i \tag{8}$$
$$\text{s.t. } a_k\langle x,\hat{\xi}_i\rangle + b_k\tau + \langle\gamma_{i,k}, d - C\hat{\xi}_i\rangle \le s_i, \quad x_i \ge 0$$
$$\left\|C^\top\gamma_{i,k} - a_k x\right\|_\star \le \lambda, \quad \gamma_{i,k}\ge 0, \qquad \sum_i x_i = 1$$

```
# Input: N, m, alpha, rho
import cvxpy as cp
from cvxRiskOpt.wass_risk_opt_pb import
    WassDRExpectation, WassDRCVaR
x = cp.Variable(m, name='x')
expect_part = WassDRExpectation(N, -1 * x, 0,
     used_norm=1)
cvar_part = WassDRCVaR(N, m, -1 * x, 0, alpha
    =alpha, used_norm=1)
portfolio_constr = cp.Problem(cp.Minimize(0),
     [cp.sum(x) == 1, x >= 0])
prob = expect_part + rho * cvar_part +
    portfolio_constr
```

Listing 1. Way 1 - cvxRiskOpt implementation of (8).

```
# Input: N, m, alpha, rho
import cvxpy as cp
from cvxRiskOpt.wass_risk_opt_pb import
    WassWCEMaxAffine
x = cp.Variable(m, name='x')
tau = cp.Variable(name='tau')
prob = WassWCEMaxAffine(N, [-x, (-1-rho/alpha
    )*x], [rho*tau, rho*(1-1/alpha) * tau],
    used_norm=1) + cp.Problem(cp.Minimize(0),
     [cp.sum(x) == 1, x >= 0])
```

Listing 2. Way 2 - cvxRiskOpt implementation of (8).

```
# Input: prob, eps, samp, solver
from cvxRiskOpt.wass_risk_opt_pb import
    update_wass_wce_params
update_wass_wce_params(prob, eps, samp)
prob.solve(solver=solver)
```

Listing 3. Solving the problem.

with $a_1 = -1$, $a_2 = -1-\rho/\alpha$, $b_1 = \rho$, $b_2 = \rho(1-1/\alpha)$, $C = 0$, $d = 0$ (of appropriate dimensions), and the 1-norm so that $\|\cdot\|_*$ is the $\infty$-norm.

The portfolio optimization problem (8) is convex and can be encoded using CVXPY. With cvxRiskOpt, (7) can be encoded in one of two ways.

The first uses `WassDRExpectation` and `WassDRCVaR` and the cvxRiskOpt problem arithmetic allowing users to encode the problem as shown in Listing 1 (notice how line 8 is similar to (7)). The second uses `WassWCEMaxAffine` to encode the reformulation (8) as in Listing 2. In both cases, $\sum_i x_i = 1$ and $x_i \ge 0$ need to be added to the problem. This can be done by creating a CVXPY Problem with no objective and the two constraints, as in Listing 1 line 7, then adding that to the cvxRiskOpt problem (line 8). After setting up the problem, the parameters can be updated and the problem is solved using the chosen solver (e.g., CLARABEL) as shown in Listing 3.

Furthermore, the resulting Problem can easily be compiled into C code using CVXPYgen. Fig. 1 compares the compute times for solving (7) 1) using CVXPY only, 2) using the cvxRiskOpt functions through Listing 1, and 3) using CVXPYgen on the CVXPY Problem in Listing 1 line 8 then calling the generated C code using the Python wrapper. We refer the readers to the "Distributionally Robust Mean-CVaR Portfolio Optimization" example of the package documentation for implementation details of solving the problem with

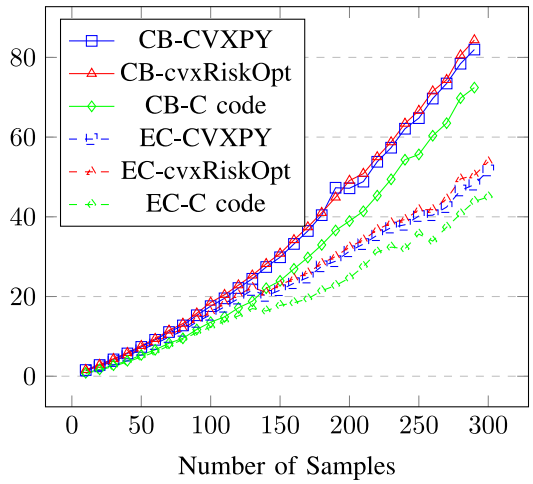


Fig. 1. Compute time (ms) vs number of samples. Solvers: CB: Clarabel, EC: ECOS.

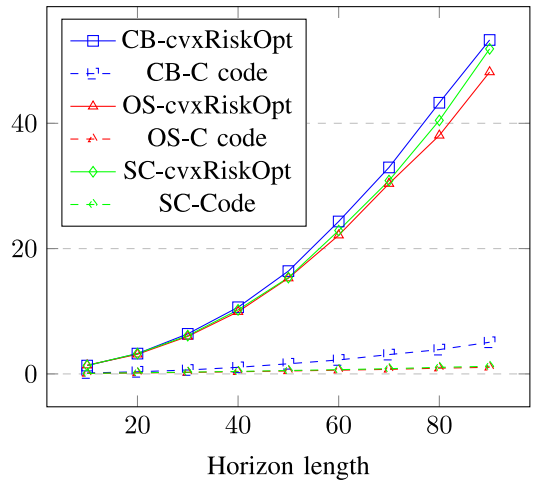


Fig. 2. Compute time (ms) vs horizon length. Solvers: CB: Clarabel. OS: OSQP. SC: SCS.

CVXPY only and compling C code. The reported time are found by aggregating the results over 100 solves. Using cvxRiskOpt adds minimal overhead and the solve times are very similar to the using CVXPY only. Compiling the code and running it using the python wrapper provides a modest speed-up of around 10%.

### B. Temperature Regulator MPC With Time Varying Constraints

Consider a simple MPC-based regulator for a 1D system under process noise. The dynamics are given by:

$$x_{t+1} = Ax_t + Bu_t + Fw_t$$

with $A = 0.9$, $B = 0.1$, $F = 0.1$ where $w_t$ is a Gaussian random variable $w_t \sim \mathcal{N}(\overline{w}_t, \sigma_w^2)$ where $\overline{w}_t$ is a time-varying mean value and $\sigma_w^2$ is the variance.

The stochastic MPC regulation problem starting at time $t$ with a horizon $H$ is given by:

$$\begin{aligned}
\min_{u,x}\ & \mathbb{E}\left[\sum_{\tau=1}^{H}\left(x_{t+\tau}^T Q x_{t+\tau}\right)\right] + \sum_{\tau=1}^{H} u_{t+\tau-1}^T R u_{t+\tau-1} \\
\text{s.t. } & x_{t+\tau} = Ax_{t+\tau-1} + Bu_{t+\tau-1} + Fw_{t+\tau-1} \\
& x_t = \underline{x}_t, \quad w_{t+\tau-1} \sim \mathcal{N}\left(\overline{w}_{t+\tau-1}, \sigma_w^2\right) \\
& \mathbb{P}\left(x_{t+\tau} \geq x_{t+\tau}^{min}\right) \geq 1-\epsilon \\
& \mathbb{P}\left(x_{t+\tau} \leq x_{t+\tau}^{max}\right) \geq 1-\epsilon.
\end{aligned} \tag{9}$$

Here, $\tau \in [1:H]$, $x_{t+\tau}, u_{t+\tau}$ are the state and control decision variables, $\underline{x}_t$ is the current *known* state, and $x_{t+\tau}^{min}, x_{t+\tau}^{max}$ are time varying bounds on the state $x_{t+\tau}$. Notice that the state is a random variable due to the noise. The last two constraints are chance constraints that require the state to remain within the chosen bounds with probability $1-\epsilon$ for some small $\epsilon$ value. The problem (9) can be reformulated into a deterministic risk-tightened problem as follows:

$$\begin{aligned}
\min_{u,x}\ & \sum_{\tau=1}^{H} \mathbb{E}\left[x_{t+\tau}^T Q x_{t+\tau}\right] + \sum_{\tau=1}^{H} u_{t+\tau-1}^T R u_{t+\tau-1} \\
\text{s.t. } & \mathbb{E}\left[x_{t+\tau}\right] = A\mathbb{E}\left[x_{t+\tau-1}\right] + Bu_{t+\tau-1} + F\mathbb{E}\left[w_{t+\tau-1}\right] \\
& \mathbb{E}[x_t] = \underline{x}_t
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}\left[x_{t+\tau}\right] &\geq x_{t+\tau}^{min} + \Psi_{\mathcal{N}}^{-1}(1-\epsilon)\|\mathrm{Cov}(x_{t+\tau})\|_2 \\
\mathbb{E}\left[x_{t+\tau}\right] &\leq x_{t+\tau}^{max} - \Psi_{\mathcal{N}}^{-1}(1-\epsilon)\|\mathrm{Cov}(x_{t+\tau})\|_2
\end{aligned}$$

where

$$\mathbb{E}\left[x_{t+\tau}\right] = A^\tau \underline{x}_t + \sum_{i=1}^{\tau} A^{\tau-i} B u_{t+i-1} + A^{\tau-i} F \overline{w}_{t+i-1},$$

$$\mathrm{Cov}(x_{t+\tau}) = \sum_{i=1}^{\tau} \left(A^{\tau-i}F\right)\ \sigma_w^2 \left(A^{\tau-i}F\right)^T,$$

$$\mathbb{E}\left[x_{t+\tau}^T Q x_{t+\tau}\right] = \mathrm{Tr}(Q\mathrm{Cov}(x_{t+\tau})) + \mathbb{E}\left[x_{t+\tau}\right]^T Q \mathbb{E}\left[x_{t+\tau}\right].$$

Finding the expected value and covariance expressions ($\mathbb{E}[x_{t+\tau}]$, $\mathrm{Cov}(x_{t+\tau})$, $\mathbb{E}[x_{t+\tau}^T Q x_{t+\tau}]$) is done using cvxRiskOpt helper functions, and the `cclp_gauss` function is used to encode the chance constraints. This simplifies the process of encoding the optimization problem while still obtaining a CVXPY Problem.

The solve times for the problem using the cvxRiskOpt functions and using the Python wrapper that calls the C code generated by CVXPYgen are reported in Fig. 2 aggregated over more than 850 solves. The compiled code results in significant speed-up by over an order of magnitude. Using cvxRiskOpt makes encoding the problem easier and enables changing the assumption about the noise, e.g., to a moment-based ambiguity set, as simple as changing the function call, while still utilizing the advantages of CVXPYgen.

### C. State Estimator

Consider a moving horizon estimator (MHE) applied to:

$$\begin{aligned}
x_{t+1} &= Ax_t + Bu_t + w_t =: f(x_t, u_t, w_t), \quad w_t \sim (0, \Sigma_w) \\
y_t &= Cx_t + v_t := h(x_t, v_t), \quad v_t \sim (0, \Sigma_v)
\end{aligned}$$

where $w_t, v_t$ are white noises with known covariances whose distribution is unknown. Denote by $\hat{x}_t$ the state estimate.

MHE considers the measures $y_t$ over a previous time horizon $N$ and the current time. Let the MHE objective at time $t$ be: $J := \sum_{t'=t-N}^{t} \left\|y_{t'} - h(\underline{x}_{t'}, 0)\right\|_{R^{-1}}^2 + \sum_{t'=t-N}^{t-1} \left\|\underline{x}_{t'+1} - f(\underline{x}_{t'}, u_{t'}, 0)\right\|_{Q^{-1}}^2$ where $\underline{x}$ is the optimization variable representing the estimated state. The constraints

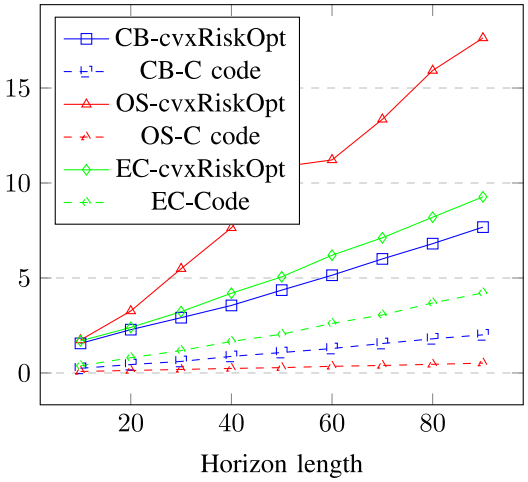


Fig. 3. Compute time (ms) vs horizon length. Solvers: CB: Clarabel. OS: OSQP. EC: ECOS.

include $\underline{x}_{t-N} = \hat{x}_{t-N}$, i.e., starting from the previous state estimate $\hat{x}_{t-N}$, and any additional constraints based on knowledge of the problem data. Here, we consider two distributionally robust chance constraints on the state:

$$\inf_{\mathbb{P}\in\mathcal{P}_{x_{t'}}} \mathbb{P}\big(\underline{x}_{t'} \le x_{max}\big) \ge 1-\epsilon, \quad \inf_{\mathbb{P}\in\mathcal{P}_{x_{t'}}} \mathbb{P}\big(\underline{x}_{t'} \ge x_{min}\big) \ge 1-\epsilon$$

where $x_{max}, x_{min}$ are known state limits. The chance constraints can easily be formulated using cvxRiskOpt functions similarly to the MPC example.

For the double integrator with $A = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 0.5dt^2 \\ dt \end{bmatrix}$, $C = \begin{bmatrix} 1 & 0 \end{bmatrix}$, and discrete time-step $dt$, the MHE problem solve times using the cvxRiskOpt functions and using the Python wrapper that calls the C code generated by CVXPYgen are reported in Fig. 3 aggregated over 200 solves for various $N$ values. Similarly to the MPC example, cvxRiskOpt simply makes encoding the problem easier while integrating directly into CVXPY. While OSQP performs the worst, the OSQP compiled code performs the best with over a 10x speed-up for $N = 90$. Clarabel and ECOS compiled code provide around 3.8x and 2.2x speed-ups for $N = 90$, respectively.

## VI. Conclusion

We introduced cvxRiskOpt, a package built on top of CVXPY that provides a high-level interface for encoding some DRO problems and stochastic and distributionally robust chance constraints. Our solution is easy to use and reduces the tedious manual reformulations required with some of these problems. Since cvxRiskOpt results in and integrates with CVXPY Problem's, we can also utilize CVXPYgen to produce C code for embedded system applications. Future directions include supporting a larger range of RBO problems.

## References

[1] S. Diamond and S. Boyd, "CVXPY: A python-embedded modeling language for convex optimization," *J. Mach. Learn. Res.*, vol. 17, no. 83, pp. 1–5, 2016.

[2] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *Proc. IEEE Int. Conf. Robot. Automat. (CAT)*, 2004, pp. 284–289.

[3] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Math. Program. Comput*, vol. 11, pp. 1–36, Mar. 2019.

[4] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *Proc. Eur. Control Conf. (ECC)*, 2013, pp. 3071–3076.

[5] P. J. Goulart and Y. Chen, "Clarabel: An interior-point solver for conic programs with quadratic objectives," 2024, *arXiv:2405.12762*.

[6] B. O'donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *J. Optim. Theory Appl.*, vol. 169, pp. 1042–1068, Jun. 2016.

[7] H. Föllmer and A. Schied, *Stochastic Finance: An Introduction in Discrete Time*. Berlin, Germany: Walter de Gruyter, 2011.

[8] P. Artzner, F. Delbaen, J.-M. Eber, and D. Heath, "Coherent measures of risk," *Math. Finan.*, vol. 9, no. 3, pp. 203–228, 1999.

[9] A. Shapiro, D. Dentcheva, and A. Ruszczynski, *Lectures on Stochastic Programming: Modeling and Theory*. Philadelphia, PA, USA: Soc. Ind. Appl. Math., 2021.

[10] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*. New York, NY, USA: Springer, 2011.

[11] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust Optimization*, vol. 28. Princeton, NJ, USA: Princeton Univ. Press, 2009.

[12] W. Wiesemann, D. Kuhn, and M. Sim, "Distributionally robust convex optimization," *Oper. Res.*, vol. 62, no. 6, pp. 1358–1376, 2014.

[13] A. Ali, J. Z. Kolter, S. Diamond, and S. P. Boyd, "Disciplined convex stochastic programming: A new framework for stochastic optimization," in *Proc. 31st UAI*, 2015, pp. 62–71.

[14] L. Ding, S. Ahmed, and A. Shapiro, *A Python Package for Multi-Stage Stochastic Programming*, Optim. Online, Omaha, NE, USA, 2019, pp. 1–42.

[15] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: Modeling and solving mathematical programs in python," *Math. Program. Comput*, vol. 3, pp. 219–260, Sep. 2011.

[16] J. Wiebe and R. Misener, "ROmodel: Modeling robust optimization problems in Pyomo," *Optim. Eng.*, vol. 23, no. 4, pp. 1873–1894, 2022.

[17] J. Sherman, N. M. Isenberg, J. D. Siirola, and C. E. Gounaris, "Recent advances in PyROS: The Pyomo solver for two-stage nonconvex robust optimization," presented at the Carbon Capture Simul. Industry Impact, Orlando, FL, USA, 2023.

[18] G. Sagnol and M. Stahlberg, "PICOS: A python interface to conic optimization solvers," *J. Open Source Softw.*, vol. 7, no. 70, p. 3915, 2022.

[19] Z. Chen and P. Xiong, "RSOME in python: An open-source package for robust stochastic optimization made easy," *INFORMS J. Comput.*, vol. 35, no. 4, pp. 717–724, 2023.

[20] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optim. Eng.*, vol. 13, pp. 1–27, Mar. 2012.

[21] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, "TinyMPC: Model-predictive control on resource-constrained microcontrollers," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2024, pp. 1–7.

[22] M. Schaller, G. Banjac, S. Diamond, A. Agrawal, B. Stellato, and S. Boyd, "Embedded code generation with CVXPY," *IEEE Control Syst. Lett.*, vol. 6, pp. 2653–2658, 2022.

[23] Z. Chen, M. Sim, and P. Xiong, "Robust stochastic optimization made easy with RSOME," *Manag. Sci.*, vol. 66, no. 8, pp. 3329–3339, 2020.

[24] "Welcome to CVXPY 1.5." 2024. Accessed: Aug 3, 2024. [Online]. Available: https://www.cvxpy.org/

[25] P. M. Esfahani and D. Kuhn, "Data-driven distributionally robust optimization using the Wasserstein metric: Performance guarantees and tractable reformulations," *Math. Program.*, vol. 171, no. 1, pp. 115–166, 2018.

[26] S. Safaoui and T. H. Summers, "Distributionally robust CVaR-based safety filtering for motion planning in uncertain environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024, pp. 103–109.

[27] R. T. Rockafellar and S. Uryasev, "Optimization of conditional value-at-risk," *J. Risk*, vol. 2, pp. 21–42, Apr. 2000.

[28] G. C. Calafiore and L. E. Ghaoui, "On distributionally robust chance-constrained linear programs," *J. Optim. Theory Appl.*, vol. 130, pp. 1–22, Dec. 2006.