# PyROS: The Pyomo Robust Optimization Solver

Jason A. F. Sherman[1], Natalie M. Isenberg[1,2], John D. Siirola[3], and Chrysanthos E. Gounaris[1,*]

[1]Department of Chemical Engineering and Center for Advanced Process Decision-making, Carnegie Mellon University, Pittsburgh, PA, USA
[2]Data Science and Machine Intelligence, Pacific Northwest National Laboratory, Richland, WA, USA
[3]Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, USA
[*]Corresponding author: gounaris@cmu.edu

June 18, 2026

**Abstract**

We present PyROS, a Python-based meta-solver that automates a generalized cutting-set algorithm for the solution of nonconvex two-stage robust optimization (RO) problems with uncertain equality constraints. Freely available through the open-source optimization software package Pyomo, PyROS is designed to operate on a user-provided deterministic model and uncertainty set, such that a solution to the RO counterpart can be obtained without any reformulation requirements on the part of the modeler. Second-stage degree-of-freedom variables are supported through the automatic construction of static, affine, or quadratic decision rules. PyROS includes a suite of pre-implemented classes to facilitate the representation of uncertainty sets that are commonly used in the RO literature; custom-written, nonlinear programming representable uncertainty sets are also allowed. Thus, PyROS is designed to enable users to seamlessly transition their deterministic optimization models to robust optimization workflows. We demonstrate the capabilities, reliability, and performance of PyROS with the results of a computational benchmarking study.

**Keywords:** robust optimization, nonlinear programming, algebraic modeling language, mathematical modeling, nonconvex optimization

## 1 Introduction

Optimization problems are often subject to some degree of uncertainty in the problem parameters. This uncertainty may be aleatoric (due to, for example, stochastic temporal variations in a system's operating conditions) or epistemic (due to, for example, effects neglected by a model or measurement errors). In many cases, an optimal solution to a problem subject to the nominal realization of the uncertain parameters may become suboptimal or even infeasible when the uncertain parameters deviate even only slightly from the nominal realization. If the parametric uncertainty is quantifiable, then an optimization under uncertainty methodology may be applied to identify a solution that is immunized against the uncertainty.

Robust optimization (RO) [1, 2] is one such methodology. In an RO approach, the uncertain parameters are constrained in value to an *a priori* defined set, and the goal is to obtain a solution that is guaranteed to remain feasible subject to any uncertain parameter realization within that set. RO problems can be classified according to whether (some of) the decision variables are adjustable to the uncertain parameters. In a single-stage, or *nonadjustable*, RO problem, all decision variables must be determined "here-and-now", that is, before the uncertain parameters are observed. In contrast, multi-stage, or *adjustable*, RO problems accommodate "wait-and-see" decision variables, that is, variables that are determined after (some of) the uncertain parameters have been observed [3]. For tractability purposes, the adjustability of the wait-and-see variables is often approximated with decision rules [2, 3] or finite adaptability [4, 5]. We are particularly interested in adjustable RO problems in which every wait-and-see decision variable is adjustable to all of the uncertain parameters; such problems are commonly referred to as *two-stage*.

There are many approaches for solving RO problems; comprehensive surveys are presented in [2, 6–8]. Reformulation approaches involve casting an RO problem to an equivalent single-level program, usually through duality-based arguments [2, 3]; the equivalent program can then solved with an appropriate deterministic optimization algorithm, provided that it remains tractable despite its generally larger size. Although usage of reformulation approaches is common, their applicability is limited to specific problem types and uncertainty set geometries; often, linearity or convexity assumptions must be met [7, 9]. Adjustable RO problems are also commonly solved with cutting-plane methods, such as modified Benders decomposition approaches [10]; in particular, two-stage robust mixed-integer linear programs (MILPs) can also be solved with column-and-constraint generation [11].

RO solution approaches that are applicable to nonlinear RO problems or mixed-integer nonlinear RO problems have more recently attracted interest [7]. Cutting-set methods based on that of [12] involve relaxing an RO problem by replacing the uncertainty set with a finitely sampled subset, and then iteratively tightening the relaxation by adaptively refining this subset until solving the relaxed problem yields a desirable solution to the original RO problem [13, 14]. The cutting-set approach of [14], in particular, is applicable to nonconvex two-stage RO problems with uncertain equality constraints. Alternatively, linearization-based approaches can also be applied to nonlinear adjustable RO problems with uncertain equality constraints [15, 16]. Adaptive bundle methods have been proposed for the solution of general nonconvex min-max RO problems [17] and were recently integrated into an outer approximation approach for mixed-integer nonlinear nonadjustable RO problems [18].

RO is closely related to the field of semi-infinite programming, for which there is also a rich theory and broad range of solution approaches [19–21]. As shown in [3, 21], a single-stage RO problem can be written as a semi-infinite program (SIP), so that SIP solution approaches are also applicable to RO problems. The previously mentioned cutting-set approaches for solving RO problems are essentially variants of the so-called discretization methods for the solution of SIPs [21]. Applicable to nonconvex mixed-integer nonlinear SIPs, adaptive discretization methods were first proposed in [22] and subsequently developed in works such as [23–26]. Furthermore, as discussed or shown in [3, 21, 27], adjustable RO problems, including those with uncertain equality constraints, can be cast to existence-constrained SIPs, which can be solved with the adaptive discretization approach of [27].

RO has been successfully applied to complex problems in many fields [2, 6–8]. In medical logistics applications, RO formulations have been devised and solved for allocating medical evac-

uation assets [28] and ambulance deployment [29]. In other logistics applications, RO has been applied to uncertain vehicle routing, scheduling, network design, and network flow problems [13, 30–38]. RO has also been used to solve problems in finance [39], particularly portfolio selection problems [40]. Machine learning problems have also been solved with RO-based approaches [41–43]. More recently, RO has been applied to various power systems optimization problems under renewable generation and/or load demand uncertainty [44]. Nonlinear RO solution approaches have been used in several chemical engineering applications [14, 16, 45–50].

The wide arrays of solution methodologies and successful applications of RO have led to significant interest in software tools for automatically solving uncertain optimization problems using RO in computational settings [7, 8]. In light of the many available solution methods, and absent any domain expertise in RO, transitioning from a deterministic optimization workflow (for which many established software tools are available) to an RO-based workflow (for which relatively few established software tools are available) may be challenging for practitioners. In response, the number of general-purpose RO software tools has increased gradually over the past two decades. These RO tools are, in general, designed so that the user codes, along with an uncertainty set, a deterministic or uncertain optimization problem, in a manner similar to that in which deterministic optimization problems are coded in other established optimization software tools. The corresponding RO problem is then automatically inferred and solved using an established RO solution approach. We survey the literature on existing RO tools in the following paragraphs.

The AIMMS Robust Optimization Add-on is an extension of the the proprietary algebraic modeling language (AML) AIMMS[1] for the formulation and robust solution of uncertain LPs and MILPs. Adjustable variables are supported via linear decision rules, but all adjustable variables must be continuous. The uncertain models are solved robustly via automatic dual reformulation; interfaces to many state-of-the-art deterministic optimizers are provided for solution of the reformulated problem. There are three explicitly supported uncertainty set types: box, ellipsoidal, and a convex hull of a finite user-supplied set of scenarios.

JuMPeR[2] is an open-source extension of the Julia-based AML JuMP [51]. Designed to facilitate the formulation and robust solution of uncertain LPs and MILPs, JuMPeR supports adjustable variables; adjustability is approximated with automatically constructed static and/or affine decision rules. Robust solution can be performed with built-in reformulation or cutting-plane algorithms; JuMP interfaces to state-of-the-art deterministic optimizers are used to solve the reformulated problem or cutting-plane subproblems. The JuMPeR package provides pre-implemented polyhedral and ellipsoidal uncertainty set classes, along with an interface for users to define custom uncertainty sets. Though JuMPeR does not support uncertain objective functions, this absence of support is mathematically without loss of generality, as the objective can be manually recast via an epigraph reformulation.

The open-source C++ software package libDIPS[3] [52], built on top of the libALE AML, is comprised of a library of discretization-based solvers for the solution of nonconvex hierarchical programs, including SIPs, generalized SIPs, and existence-constrained SIPs. Hence, libDIPS is applicable to the solution of nonconvex mixed-integer nonlinear RO problems with uncertain inequality constraints. Note that libDIPS, as of version 0.3.1, does not support the solution of hierarchical programs with coupling equality constraints, but can be extended to support such models in the

[1] AIMMS. See https://www.aimms.com
[2] JuMPeR. See https://github.com/IainNZ/JuMPeR.jl
[3] libDIPS. See https://git.rwth-aachen.de/avt-svt/public/libdips

same sense as that presented in, for example, [27, 53, 54]. The libDIPS solvers are designed so that the user explicitly and fully codes and provides, in lieu of a syntactically coded SIP, all the subproblems of their chosen discretization algorithm. The subproblems can be solved with state-of-the-art deterministic optimizers. We note that libDIPS effectively supports any mixed integer nonlinear programming representable uncertainty set.

Robustimizer[4] [55] is a graphical-user-interface-based application for surrogate-model-based uncertainty quantification and single-stage robust optimization. Nonlinear optimization models and nonlinear, data-driven uncertainty quantifications are supported. Written using MATLAB, Robustimizer can be integrated with other software tools, and its robust optimization capabilities are rooted in methodologies from the related area of robust design optimization, such as that of [56].

ROC++[5] [57] is an open-source C++ package for the coding and robust solution of uncertain LPs and MILPs. Adjustable variables, including binary adjustable variables, are supported via automatically constructed constant, piecewise-constant, or affine decision rules; finite adaptability approximations are also available. The uncertain models are automatically solved robustly using reformulation approaches; interfaces to Gurobi and SCIP are provided for solution of the reformulated problems. Cone representable uncertainty sets, including polyhedral and ellipsoidal sets, are supported. To specify the uncertainty set for an RO model in ROC++, the user must specify the explicit constraints representing the set. Furthermore, ROC++ supports endogenous (i.e., decision-dependent) uncertainty. The ROC++ package also includes a Python interface called ROPy.

ROME[6] [58] is an open-source, MATLAB-based toolbox for coding and robustly solving uncertain linear programs (LPs). Adjustable variables are supported through linear and deflected linear decision rules. Robust solution of the uncertain programs is performed via reformulation; interfaces to the SDPT3, CPLEX, and MOSEK optimizers are provided for solution of reformulated problems. Polyhedral and ellipsoidal uncertainty sets are supported. Note that ROME also provides capabilities for distributionally robust optimization (DRO). A related tool, RSOME[7] [59], to which a Python interface[8] was more recently made available [60], uses robust stochastic optimization to solve the same class of problems and also supports DRO.

ROModel[9] [61, 62] is an open-source extension of the open-source, Python-based AML Pyomo [63, 64] for the coding and solution of nonconvex mixed-integer nonlinear RO problems. Adjustable variables are supported via linear decision rules. RO problems can be solved automatically with reformulation or cutting-set approaches; Pyomo interfaces to state-of-the-art deterministic optimizers are used to solve the reformulated problem or cutting-set subproblems. ROModel provides pre-implemented interfaces for instantiating polyhedral or ellipsoidal sets and for defining custom uncertainty sets that can be represented by mixed-integer nonlinear constraints.

ROPI[10] [65] is an open-source C++ library for coding and robustly solving uncertain single-stage LPs and MILPs. The library contains an automatic reformulation capability and interfaces to the commercial deterministic solvers CPLEX, Gurobi, and Xpress for the solution of the reformulated problems. As of version 0.1.0, only finite scenario-based (i.e., discrete) uncertainty sets are

[4] Robustimizer. See https://robustimizer.com/
[5] ROC++. See https://sites.google.com/usc.edu/robust-opt-cpp/
[6] ROME. See https://robustopt.com/resources.html
[7] RSOME. See https://www.rsomerso.com/
[8] The Python interface to RSOME is documented at https://xiongpengnus.github.io/rsome/
[9] ROModel. See https://github.com/cog-imperial/romodel
[10] ROPI. https://num.math.uni-goettingen.de/~m.goerigk/ropi/

supported.

SIPAMPL[11] [66] is an open-source software package for coding general nonlinear semi-infinite programs, and thus nonlinear single-stage RO problems, in AMPL, a proprietary AML. SIPs coded with SIPAMPL can be solved automatically with the optimizer NSIPS [67], which offers discretization-based, interior point, sequential quadratic programming, and penalty methods for the solution of SIPs. Problems can also be solved quasi-manually with the MATLAB function `fseminf` [68]. Only box uncertainty sets are (explicitly) supported.

YALMIP[12] [69] is an open-source, MATLAB-based optimization toolbox containing a module that facilitates coding and robust solution of uncertain single-stage linear, quadratic, second-order cone, and semidefinite programs [70]. The module contains automatic RO reformulation capabilities; the reformulated problems can be solved with built-in interfaces to many state-of-the-art deterministic optimization solvers. Cone representable uncertainty sets, including polyhedral and ellipsoidal sets, are supported.

In this work, we present the **Py**omo **R**obust **O**ptimization **S**olver (**PyROS**), a Python-based meta-solver for nonconvex two-stage RO problems. PyROS is included in the open-source optimization software package Pyomo [63, 64] and is designed to operate directly on a user-provided deterministic nonlinear program (NLP) and accompanying uncertainty set. We further characterize PyROS as follows:

1. PyROS automates an extended form of the cutting-set algorithm of [14]. Thus, PyROS is, to our knowledge, the first Python/Pyomo-based optimization software tool that can automatically solve nonconvex two-stage RO problems with uncertain nonlinear equality constraints. The cutting-set subproblems of PyROS can be solved using virtually any deterministic NLP optimizer to which Pyomo provides an interface.

2. PyROS is applicable to RO problems with *non-fixed recourse*. Existing RO software tools that depend on reformulation approaches do not necessarily support models with non-fixed recourse, that is, models in which the coefficients of the adjustable variables in the constraints of the model are potentially uncertain.

3. PyROS supports adjustable variables via constant, affine, and quadratic decision rules. In particular, existing RO software tools do not explicitly support quadratic decision rules.

4. PyROS provides a collection of pre-implemented classes for facile representation of polyhedral, ellipsoidal, and finite scenario-based (i.e., discrete) uncertainty sets. An abstract base class is provided to facilitate the implementation of custom-written (nonconvex) NLP representable uncertainty sets, which are within the scope of the underlying cutting-set algorithm.

Additionally, we compile a novel library of over 8500 benchmark nonlinear RO model instances, derived from a collection of ten small deterministic models that were previously presented in the optimization literature and a broad range of uncertainty set types that are familiar to the RO literature. To demonstrate the utility of PyROS for solving RO model instances, we present the results of a comprehensive numerical study based on this library.

The remainder of this document is organized as follows. In Section 2, we present our formulation of the general nonconvex two-stage RO problems that can be solved with PyROS. In Section 3,

[11] SIPAMPL. See http://www.norg.uminho.pt/aivaz/sipampl.html

[12] YALMIP. See https://yalmip.github.io/

we formally present and discuss the details of the PyROS cutting-set algorithm. In Section 4, we discuss the PyROS solver interface and present a few computational examples. Our comprehensive numerical study is presented in Section 5. Finally, our conclusions are stated in Section 6. We remark that the contents of this work pertain to PyROS specifically as of version 1.3.13.

## 2 Model Formulation

In this section, we present the general forms of the deterministic model on which the PyROS algorithm is designed to operate and the robust optimization counterpart that the algorithm is designed to solve.

### 2.1 Deterministic Problem

We are interested in a deterministic NLP of the form

$$\begin{aligned} (\mathrm{D}): \quad & \min_{\substack{x\in\mathscr{X}\\ z\in\mathbb{R}^{n_z}\\ y\in\mathbb{R}^{n_y}}} \quad f_1(x) + f_2(x,z,y;q^{\mathrm{nom}}) \\ & \text{s.t.} \quad g_i(x,z,y;q^{\mathrm{nom}}) \le 0 \qquad \forall\, i\in\mathscr{I} \\ & \qquad\quad h_j(x,z,y;q^{\mathrm{nom}}) = 0 \qquad \forall\, j\in\mathscr{J}, \end{aligned}$$

where $x\in\mathbb{R}^{n_x}, z\in\mathbb{R}^{n_z}, y\in\mathbb{R}^{n_y}$, denote the first-stage variables (or *design variables*), second-stage degree-of-freedom variables (or *control variables*), and (second-stage) state variables, respectively; $q^{\mathrm{nom}}\in\mathbb{R}^{n_q}$ denotes the nominal realization of the uncertain model parameters; $\mathscr{X}\subseteq\mathbb{R}^{n_x}$ denotes the feasible space of the first-stage variables $x$, as defined by nonlinear constraints involving $x$ only; $f_1\colon\mathbb{R}^{n_x}\to\mathbb{R}$ denotes the first-stage objective function, $f_2\colon\mathbb{R}^{n_x+n_y+n_z+n_q}\to\mathbb{R}$ denotes the second-stage objective function, $g_i\colon\mathbb{R}^{n_x+n_y+n_z+n_q}\to\mathbb{R}$ denotes the inequality constraint functions, and $h_j\colon\mathbb{R}^{n_x+n_y+n_z+n_q}\to\mathbb{R}$ denotes the equality constraint functions. The sets $\mathscr{I}$ and $\mathscr{J}$ are index sets for the inequality and equality constraint functions of the model, respectively.

### 2.2 The Two-Stage Robust Counterpart

In the notation introduced previously, the two-stage RO counterpart of the deterministic model (D) is

$$\begin{aligned} (\mathrm{RC}): \quad & \min_{x\in\mathscr{X}} \ \max_{q\in\mathscr{Q}} \ \min_{\substack{z\in\mathbb{R}^{n_z}\\ y\in\mathbb{R}^{n_y}}} \quad f_1(x) + f_2(x,z,y,q) \\ & \qquad\qquad \text{s.t.} \quad g_i(x,z,y,q) \le 0 \qquad \forall\, i\in\mathscr{I} \\ & \qquad\qquad\qquad\ \ h_j(x,z,y,q) = 0 \qquad \forall\, j\in\mathscr{J}, \end{aligned}$$

where $q\in\mathbb{R}^{n_q}$ denotes the uncertain parameters, which are constrained in value to the compact uncertainty set $\mathscr{Q}\subset\mathbb{R}^{n_q}$. The set $\mathscr{Q}$ may be *continuous* (i.e., a bounded infinite set that can be represented by nonlinear programming constraints), or it may be *discrete* (i.e., a finite set of scenarios).

A point $x\in\mathscr{X}$ is feasible for (RC) provided that, for every $q\in\mathscr{Q}$, there exist $z\in\mathbb{R}^{n_z}$ and $y\in\mathbb{R}^{n_y}$ for which $(x,z,y,q)$ satisfies the constraints of (RC). The second-stage degree-of-freedom variables $z$ are fully adaptive. Observe that the objective of (RC) is to minimize the worst-case objective value. Since the feasibility requirements of (RC) are stricter than those of (D), and due to the worst-case objective focus of (RC), the optimal value of (D) is a lower bound for that of (RC).

### 2.3 Decision Rules and the Simplified Robust Counterpart

To simplify problem (RC), we introduce polynomial decision rules to approximate the adjustability of the second-stage degree-of-freedom variables $z$ to the uncertain parameters $q$. Many works in the literature on adjustable RO use polynomial decision rules [8]. Affine decision rules were first proposed by [3]. Quadratic decision rules were applied in [14] for robustly solving uncertain nonlinear process models. In [71], the quality and performance of affine, quadratic, and cubic decision rules were tested and compared on a collection of multistage RO models of linear dynamical systems.

PyROS supports static, affine, and quadratic polynomial decision rules, based on the modeling framework of [14]; the choice of decision rules is made *a priori* by the modeler. If static (i.e., degree 0) decision rules are chosen, then the modeler essentially forgoes the flexibility to adjust the second-stage degree-of-freedom variables after the uncertain parameters realize, and the entries of $z$ are redesignated to be part of the first-stage variables $x$. Otherwise, we introduce the decision rule equations

$$z_\ell = v_\ell(d_\ell, q) \qquad \ell = 1, 2, \ldots, n_z, \tag{1}$$

in which, for each $\ell = 1, 2, \ldots, n_z$, the quantity $v_\ell \colon \mathbb{R}^{n_{d,\ell}} \times \mathcal{Q} \to \mathbb{R}$ is a scalar polynomial function in the parameters $q$ with coefficients $d_\ell \in \mathbb{R}^{n_{d,\ell}}$. Affine (i.e., degree 1) decision rules bear the form

$$v_\ell(d_\ell, q) = d_\ell^0 + \sum_{t=1}^{n_q} d_{\ell,t}^1 q_t \qquad \ell = 1, 2, \ldots, n_z, \tag{2}$$

such that $d_\ell = (d_\ell^0, (d_{\ell,t}^1)_{t \in [n_q]})$, with $[n] = \{1, 2, \ldots, n\}$ for any nonnegative integer $n$. Quadratic (i.e., degree 2) decision rules bear the form

$$v_\ell(d_\ell, q) = d_\ell^0 + \sum_{t=1}^{n_q} d_{\ell,t}^1 q_s + \sum_{t_1=1}^{n_q} \sum_{t_2=t_1}^{n_q} d_{\ell,t_1,t_2}^2 q_{t_1} q_{t_2} \qquad \ell = 1, 2, \ldots, n_z, \tag{3}$$

such that $d_\ell = \left(d_\ell^0, (d_{\ell,t}^1)_{t \in [n_q]}, (d_{\ell,t_1,t_2}^2)_{(t_1,t_2) \in [n_q]^2 : t_2 \geq t_1}\right)$. The degree $\alpha \in \{0, 1, 2\}$ of the polynomial is referred to as the *decision rule order*. The quantities $d_1^0, d_2^0, \ldots, d_{n_z}^0$ are collectively referred to as the *static coefficients*, and the entries of the coefficient vector $\left((d_{\ell,t}^1)_{t \in [n_q]}, (d_{\ell,t_1,t_2}^2)_{(t_1,t_2) \in [n_q]^2 : t_2 \geq t_1}\right)_{\ell \in [n_z]}$ are collectively referred to as the *non-static coefficients*.

To simplify the robust counterpart (RC), we also assume that, for every $x \in \mathcal{X}, z \in \mathbb{R}^{n_z}, q \in \mathcal{Q}$, there exists a unique $y \in \mathbb{R}^{n_y}$ such that $h_j(x, z, y, q) = 0 \, \forall \, j \in \mathcal{J}$. As noted in [14], any entry of $y$ that cannot be uniquely determined via the equality constraints should be redesignated to the second-stage degree-of-freedom variable vector $z$.

Under these conditions, the robust counterpart (RC) is approximated as

$$(\mathrm{RC}') : \min_{\substack{x \in \mathcal{X} \\ d \in \mathbb{R}^{n_d}}} \max_{\substack{q \in \mathcal{Q} \\ z \in \mathbb{R}^{n_z} \\ y \in \mathbb{R}^{n_y}}} \min_{\zeta \in \mathbb{R}} \zeta \tag{4a}$$

$$\text{s.t.} \quad f_1(x) + f_2(x, z, y, q) \leq \zeta \tag{4b}$$

$$g_i(x, z, y, q) \leq 0 \qquad \forall \, i \in \mathcal{I} \tag{4c}$$

$$\text{s.t.} \quad h_j(x, z, y, q) = 0 \qquad \forall \, j \in \mathcal{J} \tag{4d}$$

$$z_\ell = v_\ell(d_\ell, q) \qquad \forall \, \ell \in [n_z], \tag{4e}$$

in which $d = (d_1, d_2, \ldots, d_{n_z})$, such that $n_d = \sum_{\ell \in [n_z]} n_{d,\ell}$. Note that the objective function has been recast to an equivalent epigraph form through the introduction of the variable $\zeta \in \mathbb{R}$ and (4a)–(4b). We say that a point $(\zeta, x, d) \in \mathbb{R} \times \mathcal{X} \times \mathbb{R}^{n_d}$ is feasible for (RC′) provided that, for every $q \in \mathcal{Q}$, there exist $(z, y) \in \mathbb{R}^{n_z} \times \mathbb{R}^{n_y}$, determined uniquely by (4d)–(4e), such that $(\zeta, x, d, z, y, q)$ satisfies (4b)–(4c). Since the decision rule constraints (4e) limit the adjustability of the second-stage variables $z$ to the uncertain parameters $q$, the optimal value of (RC′) is no less than that of (RC). We emphasize that the state variables $y$ remain fully adaptive.

# 3 Solution Methodology

PyROS is an implementation of a generalized robust cutting-set (GRCS) algorithm based on [14]. In this section, we formally present the GRCS algorithm.

## 3.1 Subproblems

At every iteration of the GRCS algorithm: (1) a *sampled problem* is solved to obtain a candidate first-stage variable solution; (2) a sequence of *separation problems* is solved to evaluate the robustness of the solution, or otherwise determine additional constraints to be imposed in the sampled problem for the next iteration.

For clarity and notational convenience, we formally fold the epigraph constraint (4b) into the inequality constraints (4c), as follows. Declare a new index $i^{\text{epi}} \notin \mathcal{I}$, and let $\mathcal{I}^{\text{aug}} = \mathcal{I} \cup \{i^{\text{epi}}\}$. For every $i \in \mathcal{I}^{\text{aug}}$, we define the function $g_i^{\text{aug}} : \mathbb{R} \times \mathcal{X} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_y} \times \mathcal{Q} \to \mathbb{R}$ by

$$g_i^{\text{aug}}(\zeta, x, z, y, q) = \begin{cases} g_i(x, z, y, q) & i \in \mathcal{I} \\ f_1(x) + f_2(x, z, y, q) - \zeta & \text{otherwise.} \end{cases} \tag{5}$$

### 3.1.1 Sampled Problem

Let $k$ be a nonnegative integer denoting the GRCS iteration number. The sampled problem of iteration $k$ is

$$(\text{SMP}_k): \min_{\substack{\zeta \in \mathbb{R} \\ x \in \mathcal{X} \\ d \in \mathbb{R}^{n_d} \\ z^r \in \mathbb{R}^{n_z}\ \forall r \\ y^r \in \mathbb{R}^{n_y}\ \forall r}} \quad \zeta \tag{6a}$$

$$\text{s.t.} \quad g_i^{\text{aug}}(\zeta, x, z^r, y^r; q^r) \le 0 \qquad \forall r \in \mathcal{K}^k, \forall i \in \mathcal{I}^{\text{aug}} \tag{6b}$$

$$h_j(x, z^r, y^r; q^r) = 0 \qquad \forall r \in \mathcal{K}^k, \forall j \in \mathcal{J} \tag{6c}$$

$$z_\ell^r = v_\ell(d_\ell; q^r) \qquad \forall r \in \mathcal{K}^k, \forall \ell \in [n_z], \tag{6d}$$

where $\mathcal{K}^k = \{0, 1, \ldots, k\}$ is an index set for the variables $(z^0, y^0), (z^1, y^1), \ldots, (z^k, y^k)$ and the sequence of uncertain parameter realizations $q^0, q^1, \ldots, q^k$ against which to explicitly impose the model constraints (4b)–(4e), through (6b)–(6d). At the start of the algorithm, we set $q^0 = q^{\text{nom}}$, such that the initial sampled problem ($\text{SMP}_0$) is equivalent to the deterministic model (D).

Problem ($\text{SMP}_k$) is a relaxation of the robust counterpart ($\text{RC}'$) since the feasibility requirements of $(\zeta, x, d)$ discussed in Section 2.3 need only be satisfied subject to the uncertain parameter realizations in the finite subset $\{q^0, q^1, \ldots, q^k\}$ of $\mathcal{Q}$. It follows that the globally optimal value of ($\text{SMP}_k$) is a lower bound for that of ($\text{RC}'$). Further, infeasibility of ($\text{SMP}_k$) implies infeasibility of ($\text{RC}'$). Observe also that ($\text{SMP}_{k+1}$) is a restriction of ($\text{SMP}_k$). Upon solution of ($\text{SMP}_k$), the realization $q^{k+1}$ is adaptively determined later in iteration $k$.

Problem ($\text{SMP}_k$) is written for minimization of the worst-case objective value. In this case, the inequality constraints (6b) include the sampled epigraph constraints

$$f_1(x) + f_2(x, z^r, y^r; q^r) - \zeta \leq 0 \qquad \forall r \in \mathcal{K}^k. \tag{7}$$

If we instead wish to minimize the nominal objective value, then we drop $i^{\text{epi}}$ from $\mathcal{I}^{\text{aug}}$ and separately add to ($\text{SMP}_k$) the single inequality

$$f_1(x) + f_2(x, z^0, y^0; q^0) - \zeta \leq 0. \tag{8}$$

### 3.1.2 Separation Problems

Let $\left(\bar{\zeta}, \bar{x}, \bar{d}, (\bar{z}^r)_{r\in\mathcal{K}^k}, (\bar{y}^r)_{r\in\mathcal{K}^k}\right)$ be an optimal solution to the sampled problem ($\text{SMP}_k$). At iteration $k$ and for each $i \in \mathcal{I}^{\text{aug}}$, a separation problem

$$(\text{SEP}_i): \quad \max_{\substack{q\in\mathcal{Q}\\ z\in\mathbb{R}^{n_z}\\ y\in\mathbb{R}^{n_y}}} \; g_i^{\text{aug}}(\bar{\zeta}, \bar{x}, z, y, q) \tag{9a}$$

$$\text{s.t.} \quad h_j(\bar{x}, z, y, q) = 0 \qquad \forall j \in \mathcal{J} \tag{9b}$$

$$z_\ell = v_\ell(\bar{d}_\ell, q) \qquad \forall \ell \in [n_z] \tag{9c}$$

is solved to assess whether $(\bar{\zeta}, \bar{x}, \bar{d})$ robustly satisfies the inequality constraints (4c) of ($\text{RC}'$). If $(z^{\text{sep},i}, y^{\text{sep},i}, q^{\text{sep},i})$ is an optimal solution to ($\text{SEP}_i$), then under the assumptions discussed in Section 2.3, the point $(\bar{\zeta}, \bar{x}, \bar{d})$ is feasible for ($\text{RC}'$) provided that $g_i^{\text{aug}}(\bar{\zeta}, \bar{x}, z^{\text{sep},i}, y^{\text{sep},i}, q^{\text{sep},i}) \leq 0$ for every $i \in \mathcal{I}^{\text{aug}}$. Otherwise, we select $i^\star \in \mathcal{I}^{\text{aug}}$ for which $g_i^{\text{aug}}(\bar{\zeta}, \bar{x}, z^{\text{sep},i}, y^{\text{sep},i}, q^{\text{sep},i}) > 0$ and set $q^{k+1} = q^{\text{sep},i^\star}$. In practice, we check that the inequality constraints are robustly satisfied up to a pre-specified relative tolerance.

Note that the uncertain parameters $q$ comprise the only true degrees of freedom of ($\text{SEP}_i$), as under the assumptions of Section 2.3, it follows that for any $q \in \mathcal{Q}$, the equality constraints (9b) and (9c) implicitly determine unique values for $z$ and $y$, respectively.

## 3.2 Statement of the Algorithm

The GRCS algorithm implemented in PyROS is formally stated as Algorithm 1. Notice that the algorithm is designed to operate on a user-provided deterministic model and uncertainty set, from which the subproblems are automatically constructed. Unlike in [14], we do not deprioritize the separation problem ($\text{SEP}_{i^{\text{epi}}}$) for the epigraph constraint (4b). The algorithm terminates at iteration $k$ if either the sampled problem ($\text{SMP}_k$) is found to be infeasible (i.e., infeasibility of ($\text{RC}'$) is proven) or ($\text{SMP}_k$) admits a solution for which the constraints of ($\text{RC}'$) are satisfied up to a relative tolerance $\varepsilon \geq 0$. Otherwise, the uncertain parameter realization $q^{k+1}$ is selected from among the separation

problem solutions that result in violations of the robust inequality constraints (4b) and (4c). Our approach for selecting $q^{k+1}$, given in lines 17–21 of Algorithm 1, is similar to that of [14]. If Algorithm 1 has terminated and returned a solution that is robust feasible, up to the provided tolerance $\varepsilon$, then the solution is also considered robust optimal provided that ($\text{SMP}_k$) is solved to global optimality and we have chosen to optimize for the worst-case, rather than the nominal, objective function value.

**Algorithm 1:** Generalized robust cutting-set (GRCS) method.

**Data:** Deterministic model (D), uncertainty set $\mathcal{Q}$, decision rule order $\alpha \in \{0,1,2\}$, relative feasibility tolerance $\varepsilon \geq 0$

1 Construct subproblems ($\text{SMP}_0$) and ($\text{SEP}_i$), for every $i \in \mathcal{I}^{\text{aug}}$, from (D) and $\mathcal{Q}$ according to decision rule order $\alpha$
2 **for** $k = 0, 1, \ldots$ **do**
3  **if** *($\text{SMP}_k$) is infeasible* **then**
4  **return** `None` `// Robust infeasible`
5  **end**
6  Solve ($\text{SMP}_k$) for point $(\bar{\zeta}, \bar{x}, \bar{d}, (\bar{z}^r)_{r\in\mathcal{K}^k}, (\bar{y}^r)_{r\in\mathcal{K}^k})$
7  Set $\mathcal{I}^{\text{viol}} \leftarrow \varnothing$
8  **for** $i \in \mathcal{I}^{aug}$ **do**
9  Solve ($\text{SEP}_i$) for point $(z^{\text{sep},i}, y^{\text{sep},i}, q^{\text{sep},i})$
10  **if** $g_i^{aug}(\bar{\zeta}, \bar{x}, z^{sep,i}, y^{sep,i}, q^{sep,i}) \leq \varepsilon \cdot \max\{1, |g_i^{aug}(\bar{\zeta}, \bar{x}, \bar{z}^0, \bar{y}^0, q^0)|\}$ **then**
11  Set $\mathcal{I}^{\text{viol}} \leftarrow \mathcal{I}^{\text{viol}} \cup \{i\}$
12  **end**
13  **end**
14  **if** $\mathcal{I}^{viol} = \varnothing$ **then**
15  **return** $(\bar{\zeta}, \bar{x})$ `// Robust feasible or optimal`
16  **else**
17  **for** $(i_1, i_2) \in \mathcal{I}^{viol} \times \mathcal{I}^{viol}$ **do**
18  Set $E_{i_1,i_2} = \max\{0, g_{i_2}^{\text{aug}}(\bar{\zeta}, \bar{x}, z^{\text{sep},i_1}, y^{\text{sep},i_1}, q^{\text{sep},i_1})\} / \max\{1, |g_{i_2}^{\text{aug}}(\bar{\zeta}, \bar{x}, \bar{z}^0, \bar{y}^0, \bar{q}^0)|\}$
19  **end**
20  Select $i^\star \in \arg\max_{i_1 \in \mathcal{I}^{\text{viol}}} \left\{ \sum_{i_2 \in \mathcal{I}^{\text{viol}}} E_{i_1,i_2} \right\}$
21  Set $q^{k+1} \leftarrow q^{\text{sep},i^\star}$
22  **end**
23 **end**

## 3.3 Algorithmic Details

We now discuss the key computational considerations in our implementation of the algorithm.

### 3.3.1 Solution of the Separation Problems

Our approach for solving the separation problem ($\text{SEP}_i$) depends on whether $\mathcal{Q}$ is a continuous or discrete set. If $\mathcal{Q}$ is a continuous set, then ($\text{SEP}_i$) is solved as a monolithic NLP. Otherwise, we may write

$$\mathcal{Q} = \{q^0, q^1, \ldots, q^k, p^1, p^2, \ldots, p^{S-(k+1)}\},$$

such that $p^1, p^2, \ldots, p^{S-(k+1)}$ are those points in $\mathcal{Q}$ that are not included among the realizations $q^0, q^1, \ldots, q^k$ of ($\text{SMP}_k$), and then solve ($\text{SEP}_i$) as follows:

1. For $s = 1, 2, \ldots, S-(k+1)$, solve the scenario-based feasibility problem

$$(\text{SEP}_s^{\text{disc}}): \quad \max_{\substack{z \in \mathbb{R}^{n_z} \\ y \in \mathbb{R}^{n_y}}} \; 0 \tag{10a}$$

$$\text{s.t.} \quad h_j(\bar{x}, z, y; p^s) = 0 \qquad \forall\, j \in \mathcal{J} \tag{10b}$$

$$z_\ell = v_\ell(\bar{d}_\ell; p^s) \qquad \forall\, \ell \in [n_z] \tag{10c}$$

   for a point $(\tilde{z}^s, \tilde{y}^s)$. Since $p^s$ is fixed, only a single $z \in \mathbb{R}^{n_z}$ exists for which (10c) is satisfied. Moreover, due to the assumption stated in Section 2.3, there now exists only a single $(z, y) \in \mathbb{R}^{n_z} \times \mathbb{R}^{n_y}$ for which (10b) and (10c) are satisfied. Hence, $(\tilde{z}^s, \tilde{y}^s)$ is the only feasible solution to ($\text{SEP}_s^{\text{disc}}$).

2. For each $i \in \mathcal{J}^{\text{aug}}$, choose

$$s_i^\star \in \underset{s \in \{1,2,\ldots,S-(k+1)\}}{\arg\max} \left\{ g_i^{\text{aug}}(\bar{\zeta}, \bar{x}, \tilde{z}^s, \tilde{y}^s; p^s) \right\},$$

   and set $(z^{\text{sep},i}, y^{\text{sep},i}, q^{\text{sep},i}) = (\tilde{z}^{s_i^\star}, \tilde{y}^{s_i^\star}, p^{s_i^\star})$. Note that, since the solution of ($\text{SMP}_k$) proves feasibility with respect to the scenarios $q^0, q^1, \ldots, q^k$, an approximate solution of ($\text{SEP}_i$) with $q$ restricted to $\{p^1, p^2, \ldots, p^{S-(k+1)}\}$ in iteration $k$ is sufficient for proving that the inequality constraint $g_i^{\text{aug}}(\bar{\zeta}, \bar{x}, z, y, q) \leq 0$ is robustly satisfied or for selecting $q^{k+1}$.

Our approach for solving problems ($\text{SEP}_i$) when $\mathcal{Q}$ is discrete is expected to be computationally efficient, since in each iteration $k$, it involves the solution of exactly $S-(k+1)$ feasibility problems. In contrast, an approach similar to that used when $\mathcal{Q}$ is continuous would have involved the solution of $|\mathcal{J}^{\text{aug}}|$ mixed-integer nonlinear programs (MINLPs) per iteration.

It should be noted that, in order to certify the robust feasibility or optimality of the most recent sampled problem solution $(\bar{\zeta}, \bar{x}, \bar{d})$, the separation problem ($\text{SEP}_i$) must be solved to global optimality for every $i \in \mathcal{J}^{\text{aug}}$. But since ($\text{SEP}_i$) is, in general, nonconvex, solving ($\text{SEP}_i$) to global optimality in every iteration may be computationally expensive. Therefore, our default approach is to solve ($\text{SEP}_i$) to local optimality, for every $i \in \mathcal{J}^{\text{aug}}$, to populate the set $\mathcal{J}^{\text{viol}}$ for eventual selection of $q^{k+1}$. However, if upon local solution of the separation problems the set $\mathcal{J}^{\text{viol}}$ remains empty, we proceed to solving them again, for every $i \in \mathcal{J}^{\text{aug}}$, but this time to global optimality. For many practical problems, global solution of the separation problems is executed in only the terminal GRCS iteration. When $\mathcal{Q}$ is discrete, the problems ($\text{SEP}_s^{\text{disc}}$) can be solved with a subordinate local NLP optimizer, since each admits exactly one feasible solution.

### 3.3.2 Subproblem Initialization

As the subproblems of Section 3.1 are, in general, nonconvex NLPs, the variables of the subproblems must be carefully initialized to minimize the risk of numerical errors encountered during solution of the subproblems. In the present section, we discuss appropriate strategies for initializing the subproblems.

The variables of the sampled problem ($\text{SMP}_0$) are initialized according to user-provided values. For $k = 1, 2, \ldots$, the variables of ($\text{SMP}_k$) are initialized according to the solution of the slack variable problem

$$
\begin{aligned}
(\text{SMP}_k^{\text{slack}}): \quad \min_{\substack{\zeta\in\mathbb{R} \\ x\in\mathscr{X} \\ d\in\mathbb{R}^{n_d} \\ z^r\in\mathbb{R}^{n_z}\,\forall r \\ y^r\in\mathbb{R}^{n_y}\,\forall r \\ \delta_i\in\mathbb{R}_+\,\forall i}} \quad & \sum_{i\in\mathscr{I}^{\text{aug}}} \delta_i && && \text{(11a)} \\
\text{s.t.} \quad & g_i^{\text{aug}}(\zeta, x, z^r, y^r; q^r) \le 0 && \forall r \in \mathscr{K}^{k-1}, \forall i \in \mathscr{I}^{\text{aug}} && \text{(11b)} \\
& g_i'(\zeta, x, z^k, y^k; q^k) \le \delta_i && \forall i \in \mathscr{I}^{\text{aug}} && \text{(11c)} \\
& h_j(x, z^r, y^r; q^r) = 0 && \forall r \in \mathscr{K}^k, \forall j \in \mathscr{J} && \text{(11d)} \\
& z_\ell^r = v_\ell(d_\ell; q^r) && \forall r \in \mathscr{K}^k, \forall \ell \in [n_z]. && \text{(11e)}
\end{aligned}
$$

The objective (11a) of ($\text{SMP}_k^{\text{slack}}$) is to minimize the overall infeasibility for ($\text{SMP}_k$). Constraints (11b)–(11e) are similar to (6b)–(6d), except that nonnegative slack variables $(\delta_i)_{i\in\mathscr{I}^{\text{aug}}}$ have been added to the right-hand sides of the inequality constraints (6b) for $r = k$. Using the solution to the prior sampled problem ($\text{SMP}_{k-1}$) and separation problems solved in iteration $k-1$, we can, without further computation, provide initial values for the variables $(\zeta, x, d, (z^r)_{r\in\mathscr{K}^k}, (y^r)_{r\in\mathscr{K}^k}, (\delta_i)_{i\in\mathscr{I}^{\text{aug}}})$ of ($\text{SMP}_k^{\text{slack}}$) for which the constraints (11b)–(11e) of ($\text{SMP}_k^{\text{slack}}$) are satisfied.

If the uncertainty set $\mathscr{Q}$ is continuous, then at iteration $k$, the variables of the separation problem ($\text{SEP}_i$) are initialized using the solution $(\bar{\zeta}, \bar{x}, \bar{d}, (\bar{z}^r)_{r\in\mathscr{K}^k}, (\bar{y}^r)_{r\in\mathscr{K}^k})$ of ($\text{SMP}_k$) and the added realizations $q^0, q^1, \ldots, q^k$. Choose

$$
r^{\text{init},i,k} \in \arg\max_{r\in\mathscr{K}^k} \{g_i^{\text{aug}}(\bar{\zeta}, \bar{x}, \bar{z}^r, \bar{y}^r, q^r)\}. \tag{12}
$$

Then the variables of ($\text{SEP}_i$) are initialized to $(\bar{z}^{r^{\text{init},i,k}}, \bar{y}^{r^{\text{init},i,k}}, q^{r^{\text{init},i,k}})$, which is, upon solution of ($\text{SMP}_k$), the best known feasible solution to ($\text{SEP}_i$). For cases in which $\mathscr{Q}$ is discrete, the variables $z$ and $y$ of the scenario-based problems ($\text{SEP}_s^{\text{disc}}$) are initialized in a similar fashion.

### 3.3.3 Decision Rule Polishing

The existence of degenerate optimal decision rules in adjustable RO settings is well-known in the RO literature [14, 72]. As discussed in [14], this degeneracy may cause the sampled problems to admit multiple equivalently optimal solutions if we have chosen to optimize over non-static decision rules (i.e., we have chosen a decision rule order $\alpha > 0$). Consequently, there arises an opportunity to exploit this degeneracy by employing a "polishing" approach, similar to that of [14], in an attempt to simplify the decision rules after the sampled problem has been solved through reduction of the number of nonzero coefficients. Though not necessary for correctness of the algorithm, such

simplification reduces the complexity of the separation problems and, in our experience, yields a net benefit in computational performance.

More specifically, upon solution of (SMP$_k$), the variables $(d, (z^r)_{r\in\mathcal{K}^k}, (y^r)_{r\in\mathcal{K}^k})$ are re-optimized, or "polished", via solution of the problem

$$
\begin{aligned}
(\text{DRP}_k): \min_{\substack{d\in\mathbb{R}^{n_d} \\ z^r\in\mathbb{R}^{n_z}\,\forall r \\ y^r\in\mathbb{R}^{n_y}\,\forall r \\ \tau_\ell\in\mathbb{R}^{n_{d,\ell}-1}\,\forall \ell}} \quad & \sum_{\ell\in[n_z]}\sum_{p=1}^{2}\sum_{\mathbf{t}\in\mathscr{P}^p}\tau^p_{\ell,\mathbf{t}} && && \text{(13a)}\\
\text{s.t.} \quad & g_i^{\text{aug}}(\bar{\zeta},\bar{x},z^r,y^r;q^r)\le 0 && \forall r\in\mathcal{K}^k, \forall i\in\mathcal{I}^{\text{aug}} && \text{(13b)}\\
& h_j(\bar{x},z^r,y^r;q^r)=0 && \forall r\in\mathcal{K}^k, \forall j\in\mathcal{J} && \text{(13c)}\\
& z^r_\ell = v_\ell(d_\ell;q^r) && \forall r\in\mathcal{K}^k, \forall \ell\in[n_z] && \text{(13d)}\\
& |d^1_{\ell,t}q_t|\le\tau^1_{\ell,t} && \forall \ell\in[n_z], \forall t\in[n_q] && \text{(13e)}\\
& |d^2_{\ell,t_1,t_2}q_{t_1}q_{t_2}|\le\tau^2_{\ell,t_1,t_2} && \forall \ell\in[n_z], \forall (t_1,t_2)\in\mathscr{P}, && \text{(13f)}
\end{aligned}
$$

in which $\bar{\zeta}$ and $\bar{x}$ are taken from the solution to (SMP$_k$), and $\mathscr{P}=\{(t_1,t_2)\in[n_q]^2 | t_2\ge t_1\}$. Of course, if $\alpha=1$, then the quadratic coefficients $(d^2_{\ell,t_1,t_2})_{(t_1,t_2)\in\mathscr{P}}$, the corresponding entries of $\tau_\ell$, and the auxiliary constraints (13f) are excluded from (DRP$_k$) for $\ell=1,2,\ldots,n_z$. As discussed in Section 2.3, when $\alpha=0$, the variables $z$ are automatically redesignated to the first-stage variable vector $x$, so that (DRP$_k$) need not be constructed or solved.

The objective (13a) of the polishing problem is to minimize the collective magnitude of the non-static decision rule coefficients. The constraints (13e) and (13f) quantify this magnitude as the 1-norm of the non-static terms of (3). Hence, the objective of (DRP$_k$) is similar to the concept of an $L_1$-regularizer in regression analysis, which, when used, often yields a sparse vector of regression parameters [73]. Indeed, in many practical cases, an optimal solution to (DRP$_k$) is such that most of the non-static coefficients are zero. Note that, unlike in the polishing approach of [14], the magnitudes of the static coefficients $d^0_1, d^0_2, \ldots, d^0_{n_z}$ are excluded from the objective (13a).

For computational tractability purposes, the variables of the polishing problem are initialized according to the solution of the sampled problem (SMP$_k$). Further, in most cases, the solution of (SMP$_k$), (SMP$_k^{\text{slack}}$), and (DRP$_k$) is made more efficient with the following heuristic. When $k=0$, the affine coefficients $(d^1_{\ell,t})_{t\in[n_q]}$ and quadratic coefficients $(d^2_{\ell,t_1,t_2})_{(t_1,t_2)\in\mathscr{P}}$ are fixed to zero for $\ell=1,2,\ldots,n_z$; hence, we do not solve (DRP$_0$). For $k=1,2,\ldots,n_q-1$, only the quadratic coefficients $(d^2_{\ell,t_1,t_2})_{(t_1,t_2)\in\mathscr{P}}$ are fixed to zero for $\ell=1,2,\ldots,n_z$. All coefficients remain unfixed for $k\ge n_q$. This heuristic aims to limit the total number of nonzero coefficients to the minimum required for sufficient recourse flexibility.

### 3.3.4 Treatment of State Variable-Independent Equality Constraints

Although rare in practice, there may exist instances of (RC′) with some $j\in\mathcal{J}$ for which the function $h_j(x,z,y,q)$ is independent of $y$. Let us conveniently denote this function by $\tilde{h}_j(x,z,q)$. Although the assumption in Section 2.3 on the existence of state variable solutions to the equality constraints (4d) may not be satisfied in cases when such a $j$ exists, our methodology implements a workaround to admit such cases regardless. This entails a reformulation of the original equality constraint, $\tilde{h}_j(x,z,q)=0$, in advance of constructing the subproblems of Algorithm 1.

More specifically, we attempt to cast the original constraint to an equivalent set of constraints that depend only on $x$ and $d$, as follows. If $\mathcal{Q}$ is discrete, then $\mathcal{Q} = \{\vartheta^1, \vartheta^2, \ldots, \vartheta^S\}$ for some finite value $S$. In this case, we remove the original constraint from model (RC′) and restrict the joint domain $\mathcal{X} \times \mathbb{R}^{n_d}$ of $(x,d)$ by adding the equality constraints

$$\tilde{h}_j\big(x, v(d,\vartheta^s), \vartheta^s\big) = 0 \qquad s = 1,2,\ldots,S, \tag{14}$$

in which $v = (v_1, v_2, \ldots, v_{n_z})$ is the vector of decision rule functions introduced in (1). On the other hand, if $\mathcal{Q}$ is continuous and $\tilde{h}_j$ is a polynomial in $(z,q)$, then we attempt to reformulate the equality constraint by matching of polynomial coefficients. Since the decision rule function $v_\ell$ is also a polynomial in $q$, for $\ell = 1,2,\ldots,n_z$, we may write

$$\tilde{h}_j(x, v(d,q), q) = \sum_{T=0}^{R_j} \sum_{\mathbf{t} \in \mathscr{R}_T} \eta_{\mathbf{t}}^j(x,d) q_1^{t_1} q_2^{t_2} \cdots q_{n_q}^{t_{n_q}}, \tag{15}$$

in which, for every integer $T \geq 0$, we define $\mathscr{R}_T = \{\mathbf{t} \in \{0,1,\ldots,T\}^{n_q} \mid t_1 + t_2 + \cdots + t_{n_q} = T\}$; the number $R_j$ is the degree of the resulting polynomial with respect to the uncertain parameters $q = (q_1, q_2, \ldots, q_{n_q})$; and, for $T = 0,1,\ldots,R_j$ and $\mathbf{t} \in \mathscr{R}_T$, the quantity $\eta_{\mathbf{t}}^j : \mathcal{X} \times \mathbb{R}^{n_d} \to \mathbb{R}$ is a scalar function that yields a polynomial coefficient. Note that the superscripts $t_1, t_2, \ldots, t_{n_q}$ in each summation term denote exponentiation. Since $\tilde{h}_j$ can be rewritten as in (15), we remove $\tilde{h}_j(x,z,q) = 0$ from (RC′), and for every tuple $\mathbf{t}$ in the summation set of (15) such that $\eta_{\mathbf{t}}^j(x,d)$ is not identically zero, the joint domain $\mathcal{X} \times \mathbb{R}^{n_d}$ of $(x,d)$ is restricted by adding to the model the equation $\eta_{\mathbf{t}}^j(x,d) = 0$. If $\eta_{\mathbf{t}}^j(x,d)$ is identically a non-zero constant for any of the tuples $\mathbf{t}$, then (RC′) is infeasible, and we terminate the algorithm at this point.

We remark that our current implementation of the above coefficient matching approach supports only cases where $R_j \leq 2$. If $R_j > 2$, or if $\tilde{h}_j$ is otherwise not compatible for polynomial reformulation, we follow a generic approach to replace the original constraint with two opposing inequalities, $\tilde{h}_j(x,z,q) \leq 0$ and $-\tilde{h}_j(x,z,q) \leq 0$, which we append to (4c). We note, however, that this alternative approach may be somewhat inefficient, as the size of the index set $\mathcal{I}$, and hence the number of separation problems, is increased.

Finally, we remark that the efficiency improvements for the solution of ($\text{SMP}_k$), ($\text{SMP}_k^{\text{slack}}$), and ($\text{DRP}_k$) discussed in the final paragraph of Section 3.3.3 are not enforced whenever the reformulation approach adds equality constraints that depend on $d$, as in such cases, fixing entries of $d$ may be unduly restrictive.

# 4 Solver Interface

## 4.1 Overview

PyROS, a Python implementation of Algorithm 1, is distributed as a contributed sub-package of the open-source optimization software package Pyomo [63, 64], which features a general-purpose AML. The Pyomo AML is highly extensible, and many extensions, like PyROS, are distributed as either sub-packages of Pyomo or as independent software packages. For example, `pyomo.dae` [74], distributed as a sub-package of Pyomo, is a framework for specifying optimization models containing differential and algebraic equations, and `PySP` [75], distributed as an independent software

package, is an extension to the Pyomo modeling language for solving stochastic programming problems. With the introduction of PyROS, Pyomo now features a generic cutting-set-based solver for nonconvex two-stage RO problems.

To allow Pyomo users to seamlessly transition their deterministic models to RO workflows, the PyROS solver is designed such that successful invocation requires only a few arguments that can be straightforwardly constructed upon preparation of the deterministic model. The required arguments are:

- the deterministic model
- a list of the first-stage variables
- a list of the second-stage degree-of-freedom variables
- a list of the parameters to be considered uncertain
- the uncertainty set
- subordinate deterministic local and global NLP solvers

Any variable of the model that is excluded from the lists of first-stage and second-stage degree-of-freedom variables is automatically considered a (second-stage) state variable. All uncertain parameters should be implemented in the Pyomo deterministic model as either mutable `Param` objects or fixed `Var` objects, and by default, the nominal value for each uncertain parameter is taken to be the number assigned to the `value` attribute of the corresponding `Param` object. The subordinate optimizers are required for solving the subproblems, which are, in general, NLPs. As discussed in Section 3, global solution of the subproblems is required to rigorously certify robust feasibility or robust optimality before termination.

Extending a deterministic optimization workflow to a PyROS-based RO workflow is expected to be straightforward for most use cases. The deterministic model and appropriate subordinate NLP solvers will have already been prepared in the deterministic optimization workflow. Among the required arguments, only the categorization of the problem's variables and the uncertainty set constitute additional information for the modeler to define the RO problem. In Section 4.2, we show the ease with which many commonly used uncertainty sets can be constructed.

## 4.2 Uncertainty Sets

Uncertainty sets are represented by instances of the PyROS `UncertaintySet` class, which is implemented as an abstract base class; PyROS then includes several pre-implemented concrete subclasses, each of which represents an uncertainty set type familiar to the RO literature. Instantiation of a pre-implemented subclass requires only that the user provide the data parameterizing the set; in particular, all pre-implemented subclasses already include methods for automatically declaring the constraints that mathematically define the set. While Figure 1 visualizes the pre-implemented `UncertaintySet` subclasses, Table 1 shows their full mathematical descriptions, which include the set definitions and inferred interval enclosures for the uncertain parameter values. For RO problems with continuous uncertainty sets, the interval enclosures are explicitly imposed on the uncertain

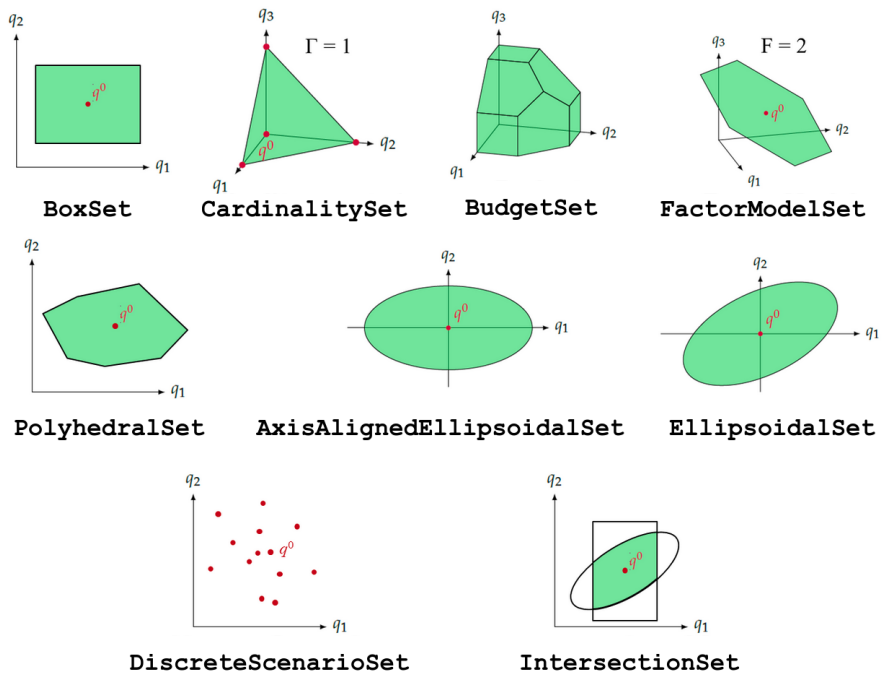


**Figure 1.** Pre-implemented PyROS `UncertaintySet` subclasses.

parameter variables for the PyROS separation subproblems (see Section 3.1.2), as part of the constraints $q \in \mathcal{Q}$. In the remainder of this section, we briefly explain each of the available PyROS uncertainty set classes.

Upon successful invocation with all required arguments, the PyROS solver automatically checks that the uncertainty set object provided by the user is bounded and contains the user-specified nominal point, in advance of initiating the iterative GRCS algorithm. An exception is raised if any of these checks returns a negative result.

**Table 1.** Full descriptions of the $n$-dimensional abstract and pre-implemented PyROS uncertainty set classes.

| `UncertaintySet` Subclass | Input Data | Set Definition | Interval Enclosure for Parameter $q_i$, $i = 1, 2, \ldots, n$ |
|---|---|---|---|
| `BoxSet` | $q^{\mathrm{L}} \in \mathbb{R}^n$, $q^{\mathrm{U}} \in \mathbb{R}^n$ | $\{q \in \mathbb{R}^n \mid q^{\mathrm{L}} \leq q \leq q^{\mathrm{U}}\}$ | $q_i^{\mathrm{L}} \leq q_i \leq q_i^{\mathrm{U}}$ |
| `CardinalitySet` | $q^0 \in \mathbb{R}^n$, $\hat{q} \in \mathbb{R}^n_+$, $\Gamma \in [0, n]$ | $\left\{ q \in \mathbb{R}^n \;\middle\vert\; \begin{array}{l} \exists \xi \in [0,1]^n : \\ q = q^0 + \hat{q} \circ \xi \\ \sum_{i=1}^{n} \xi_i \leq \Gamma \end{array} \right\}$ | $q_i^0 \leq q_i \leq q_i^0 + \min\{1, \Gamma\}\, \hat{q}_i$ |
| `BudgetSet` | $q^0 \in \mathbb{R}^n$, $b \in \mathbb{R}^L_+$, $B \in \{0,1\}^{L \times n}$ | $\left\{ q \in \mathbb{R}^n \;\middle\vert\; \begin{array}{l} B\left(q - q^0\right) \leq b \\ q \geq q^0 \end{array} \right\}$ | $q^0 \leq q_i \leq q^0 + \min\limits_{\substack{\ell \in \{1,2,\ldots,L\}: \\ B_{\ell i} = 1}} \{b_\ell\}$ |
| `FactorModelSet` | $q^0 \in \mathbb{R}^n$, $\Psi \in \mathbb{R}^{n \times F}$, $\beta \in [0,1]$ | $\left\{ q \in \mathbb{R}^n \;\middle\vert\; \begin{array}{l} \exists \xi \in [-1,1]^F : \\ q = q^0 + \Psi \xi \\ \left\lvert \sum_{j=1}^{F} \xi_j \right\rvert \leq \beta F \end{array} \right\}$ | See Section 4.2.4 |
| `PolyhedralSet` | $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ | $\{q \in \mathbb{R}^n \mid Aq \leq b\}$ | Numerically determined by solving LPs |
| `AxisAlignedEllipsoidalSet` | $q^0 \in \mathbb{R}^n$, $a \in \mathbb{R}^n_+$ | $\left\{ q \in \mathbb{R}^n \;\middle\vert\; \begin{array}{l} \sum_{\substack{i=1 \\ a_i \neq 0}}^{n} \left( \frac{q_i - q_i^0}{a_i} \right)^2 \leq 1 \\ q_i = q_i^0,\ \forall i : a_i = 0 \end{array} \right\}$ | $q_i^0 - a_i \leq q_i \leq q_i^0 + a_i$ |
| `EllipsoidalSet` | $q^0 \in \mathbb{R}^n$, $\Sigma \in \mathbb{S}^n_{++}$,[a] $s \in \mathbb{R}_+$ | $\left\{ q \in \mathbb{R}^n \mid (q - q^0)^\top \Sigma^{-1} (q - q^0) \leq s \right\}$ | $q_i^0 - (s\Sigma_{ii})^{1/2} \leq q_i \leq q_i^0 + (s\Sigma_{ii})^{1/2}$ |
| `DiscreteScenarioSet` | $\vartheta^1, \vartheta^2, \ldots, \vartheta^S \in \mathbb{R}^n$ | $\{\vartheta^1, \vartheta^2, \ldots, \vartheta^S\}$ | $\min\limits_{s \in \{1,2,\ldots,S\}} \{\vartheta_i^s\} \leq q_i \leq \max\limits_{s \in \{1,2,\ldots,S\}} \{\vartheta_i^s\}$ |
| `IntersectionSet` | $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_m \subset \mathbb{R}^n$ | $\bigcap_{i=1}^{m} \mathcal{Q}_i$ | Numerically determined by solving LPs/NLPs globally |

[a] $\mathbb{S}^n_{++}$ denotes the set of all symmetric positive definite real matrices of size $n \times n$.

### 4.2.1 Box Sets

The box uncertainty set (`BoxSet`) represents an $n$-dimensional hyperrectangle, fully described by user-provided lower and upper bound vectors $q^{\mathrm{L}} \in \mathbb{R}^n$ and $q^{\mathrm{U}} \in \mathbb{R}^n$, respectively. Listing 1 shows an example instantiation, in which we construct a five-dimensional set with $(-1.0, 2.0)$ as the parameter value bounds in each dimension.

**Listing 1.** An instantiation of the PyROS `BoxSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define uncertainty set
box_set = pyros.BoxSet(bounds=[(-1.0, 2.0)] * 5)
```

### 4.2.2 Cardinality Sets

Parameterized by an origin $q^0 \in \mathbb{R}^n$, a vector of maximal positive deviations $\hat{q} \in \mathbb{R}^n_+$, and a scalar value $\Gamma \in [0, n]$, an $n$-dimensional cardinality set (`CardinalitySet`), otherwise known as a *gamma set*, is such that (i) in each dimension, the uncertain parameter $q_i$ may deviate from the value $q^0_i$ by no more than $\hat{q}_i$; and (ii) the sum of the deviations over all dimensions may not exceed $\Gamma$. If we choose $\Gamma = 0$, then the cardinality set reduces to the singleton set $\{q^0\}$. If we choose $\Gamma = n$, then the cardinality set reduces to an $n$-dimensional hyperrectangle. Listing 2 shows an example instantiation, in which we construct a three-dimensional set with origin $(0, 0, 0)$, maximal positive deviations $(1, 2, 1.5)$, and parameter $\Gamma = 1$. Notice that only $q_1$ may achieve its maximal deviation $\hat{q}_1$.

**Listing 2.** An instantiation of the PyROS `CardinalitySet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define uncertainty set
cardinality_set = pyros.CardinalitySet(
    origin=[0, 0, 0],
    positive_deviation=[1.0, 2.0, 1.5],
    gamma=1,
)
```

### 4.2.3 Budget Sets

The $n$-dimensional budget set (`BudgetSet`) is the intersection of the unbounded box with lower bound vector $q^0$ and a polyhedron defined by a sequence of $L$ *budget constraints*. For each $\ell = 1, 2, \ldots, L$, the $\ell^{\text{th}}$ budget constraint is parameterized by a value $b_\ell \in \mathbb{R}_+$ and an incidence matrix $B_\ell \subseteq \{1, 2, \ldots, n\}$ indicating the entries of $q$ that participate in the constraint, such that the sum of the positive deviations of these entries from the offset $q^0$ may not exceed $b_\ell$. It should be noted that $q^0$ is an optional argument to the `BudgetSet` constructor; by default, $q^0$ is set to the $n$-dimensional zero vector. Listing 3 shows an example instantiation of a three-dimensional set with a single budget constraint in which all three uncertain parameters participate, and under which they cannot cumulatively exceed the origin $(0, 1, 0)$ by more than 2 units.

**Listing 3.** An instantiation of the PyROS `BudgetSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define uncertainty set
budget_set = pyros.BudgetSet(
    budget_membership_mat=[[1, 1, 1]],
    rhs_vec=[2],
    origin=[0, 1, 0],
)
```

### 4.2.4 Factor Model Sets

The $n$-dimensional factor model set (`FactorModelSet`) defines an allowable set of disturbances about an origin point $q^0 \in \mathbb{R}^n$, based on an allowable set of linear combinations of "factors" drawn from an $F$-dimensional set with coefficients specified by an $n \times F$ real matrix $\Psi$ that has full column rank. The quantity $\beta \in [0,1]$ specifies an upper bound for the fraction of the $F$ factors that may realize their maximum allowable values. In most practical applications, we have $F \ll n$, so that the space of factors is of a significantly lower dimension than that of the set itself.

Interval enclosures $[q_i^{\mathrm{L}}, q_i^{\mathrm{U}}]$ for the the uncertain parameters $q_i$ $(i = 1,2,\ldots,n)$ confined to a factor model set are calculated as follows. Define $k = \lfloor F(\beta+1)/2 \rfloor$. For $i = 1,2,\ldots,n$, let $M_i$ denote the number of nonnegative entries in the $i^{\text{th}}$ row of $\Psi$, and let $f_{ij}$ denote the column index of the $j^{\text{th}}$ largest entry of the $i^{\text{th}}$ row of $\Psi$. Then the factor model set satisfies the following properties.

(a) If $M_i > k$, then

$$q_i^{\mathrm{L/U}} = q_i^0 \mp \left( \sum_{j=1}^{k} \Psi_{if_{ij}} + \big(F(\beta+1) - 2k - 1\big)\Psi_{if_{i,k+1}} - \sum_{\ell=k+2}^{F} \Psi_{if_{i\ell}} \right).$$

(b) Else if $M_i < F - k$, then

$$q_i^{\mathrm{L/U}} = q_i^0 \mp \left( \sum_{j=1}^{F-k-1} \Psi_{if_{ij}} - \big(F(\beta+1) - 2k - 1\big)\Psi_{if_{i,F-k}} - \sum_{\ell=F-k+1}^{F} \Psi_{if_{i\ell}} \right).$$

(c) Else,

$$q_i^{\mathrm{L/U}} = q_i^0 \mp \left( \sum_{j=1}^{M_i} \Psi_{if_{ij}} - \sum_{\ell=M_i+1}^{F} \Psi_{if_{i\ell}} \right).$$

Listing 4 shows an example instantiation, in which we construct a four-dimensional set with two factors, no more than one of which may attain their extreme values ($\beta = 1/2$).

**Listing 4.** An instantiation of the PyROS `FactorModelSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define uncertainty set
factor_model_set = pyros.FactorModelSet(
    origin=[0] * 4,
    number_of_factors=2,
    psi_mat=[[0.1, 0], [0.1, 0], [0, 0.1], [0, 0.1]],
    beta=1/2,
)
```

### 4.2.5 Polyhedral Sets

The `PolyhedralSet` object is a representation of a general polyhedron and may be viewed as a generalization of the box, gamma, budget, and factor model sets. An instance of the `PolyhedralSet` object is parameterized by a coefficient matrix $A \in \mathbb{R}^{m \times n}$ and right-hand side vector $b \in \mathbb{R}^m$. Listing 5 shows an example instantiation, in which we construct a two-dimensional triangle bounded by the coordinate axes, the line $q_1 = q_2$, and the line $q_1 = 1$.

**Listing 5.** An instantiation of the PyROS `PolyhedralSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define uncertainty set
polyhedral_set = pyros.PolyhedralSet(
    lhs_coefficients_mat=[[-1, 0], [0, -1], [-1, 1], [1, 0]],
    rhs_vec=[0, 0, 0, 1],
)
```

### 4.2.6 Axis-Aligned Ellipsoidal Sets

The axis-aligned ellipsoidal set (`AxisAlignedEllipsoidalSet`) is an $n$-dimensional hyperellipsoidal region of which the principal axes are parallel to the coordinate axes. The set is parameterized by a center $q^0 \in \mathbb{R}^n$ and a vector $a \in \mathbb{R}^n_+$ of nonnegative half-lengths. Listing 6 shows the construction of the two-dimensional origin-centered axis-aligned ellipsoidal set with semiaxes of length 2.

**Listing 6.** An instantiation of the PyROS `AxisAlignedEllipsoidalSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define axis-aligned ellipsoidal uncertainty set
aligned_ellipsoidal_set = pyros.AxisAlignedEllipsoidalSet(
    center=[0, 0],
    half_lengths=[2, 2],
)
```

### 4.2.7 (General) Ellipsoidal Sets

The (general) ellipsoidal uncertainty set (`EllipsoidalSet`) is an $n$-dimensional hyperellipsoidal region, parameterized by a center $q^0 \in \mathbb{R}^n$ and an $n \times n$ symmetric positive definite real matrix $\Sigma$ defining the region's size and orientation. We may optionally also specify a nonnegative real-valued factor $s$ that scales the squares of the ellipsoidal set axes; by default, PyROS sets $s = 1$. If $\Sigma$ is a diagonal matrix, then the region is axis-aligned, and the length of the $i^{\text{th}}$ semiaxis is given by $\sqrt{s\Sigma_{ii}}$, the square root of the respective scaled diagonal entry, for $i = 1, 2, \ldots, n$. Listing 7 shows the construction of the two-dimensional, origin-centered, axis-aligned ellipsoid with semiaxes of length 2.

**Listing 7.** An instantiation of the PyROS `EllipsoidalSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define ellipsoidal uncertainty set
```

```
ellipsoidal_set = pyros.EllipsoidalSet(
    center=[0, 0],
    shape_matrix=[[4, 0], [0, 4]],
)
```

Ellipsoidal uncertainty sets are often used when the uncertain parameters are assumed to follow a multivariate normal probability distribution with mean $q^0$ and covariance matrix $\Sigma$ [31, 76]. The relationships between $q^0, \Sigma, s$, the geometry of the ellipsoidal region, and the properties of the distribution are conveniently discussed in [77].

### 4.2.8 Discrete Sets

The discrete uncertainty set (`DiscreteScenarioSet`) is a finite set of uncertain parameter realizations (i.e., *scenarios*); in practical applications, these scenarios may be gathered from prior observations of the uncertain parameters. Listing 8 shows the instantiation of a two-dimensional set with three scenarios.

**Listing 8.** An instantiation of the PyROS `DiscreteScenarioSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# Define uncertainty set
discrete_scenario_set = pyros.DiscreteScenarioSet(
    scenarios=[(1.0, 1.0), (2.0, 1.0), (1.0, 2.0)],
)
```

### 4.2.9 Intersection Sets

The `IntersectionSet` class represents an intersection of finitely many uncertainty sets, each represented by an `UncertaintySet` instance. Listing 9 shows the instantiation of the intersection of a box set and an ellipsoidal set of dimension $n = 2$.

**Listing 9.** An instantiation of the PyROS `IntersectionSet`.

```
# Import the PyROS module
import pyomo.contrib.pyros as pyros

# sets for which intersection is to be taken
box_set = pyros.BoxSet(bounds=[[-0.3, 0.3], [-0.3, 0.3]])
ellipsoidal_set = pyros.AxisAlignedEllipsoidalSet(
    center=[0, 0],
    half_lengths=[0.2, 0.2],
)

# Note: sets must be passed by keyword,
#       but keyword argument names are arbitrary
intersection_set = pyros.IntersectionSet(
    set_1=box_set,
    set_2=ellipsoidal_set,
)
```

#### 4.2.10 Custom Uncertainty Sets

Custom uncertainty sets can be implemented by inheriting from the `UncertaintySet` class. Successful implementation of a concrete `UncertaintySet` subclass requires defining methods for:

- declaring the NLP constraints that mathematically define the set
- evaluating the uncertain parameter interval enclosures
- checking whether the set is nonempty
- checking whether the set is bounded
- checking whether the set contains a given point

### 4.3 Optional Arguments to the PyROS Solver

In addition to the mandatory arguments passed to the PyROS solver (see Section 4.1), there are optional settings that a user may specify through a sequence of keyword arguments. All optional arguments are fully described in the official documentation, available at `https://pyomo.readthedocs.io/en/stable/`. Below, we discuss a few select arguments.

#### 4.3.1 Sampled Problem Options

In alignment with the discussion of Section 3.1.1, PyROS provides an option `objective_focus` for selection of the objective focus. Possible values are:

- `ObjectiveType.nominal`: Optimize for the nominal objective value. Selecting this option leads only to a proven robust feasible solution.
- `ObjectiveType.worst_case`: Optimize for the worst-case objective value. Selecting this option leads to a solution that is proven robust optimal (up to the chosen decision rule approximation), provided the sampled problems are solved to global optimality.

In addition, a user may choose to solve the sampled problem to either local or global optimality, through the boolean argument `solve_master_globally`. If this argument is set to `True`, and a worst-case objective focus is chosen, then the model solution found at the end of the PyROS algorithm is proven robust optimal. Otherwise, only robust feasibility is proven, in alignment with the discussion of Section 3.2.

By default, the focus is to optimize for the nominal objective value and the sampled problems are solved to local optimality. This is because, in many practical use cases, we expect the deterministic problem (and even more so the sampled problems at later GRCS iterations) to be large, highly nonconvex, and therefore challenging to solve globally.

#### 4.3.2 Decision Rule Options

As discussed in Section 2.3, PyROS supports polynomial decision rules of degree 0 (static), 1 (affine), and 2 (quadratic) to approximate the second-stage degree-of-freedom variables. The degree may be specified through the `decision_rule_order` keyword argument, for which the default value is 0.

Trivially, for models with no second-stage degree-of-freedom variables, PyROS does not construct a decision rule approximation, and hence, the solution returned by the solver is independent of the argument `decision_rule_order`.

#### 4.3.3 Separation Problem Options

The user may benefit from several efficiencies provided by PyROS for solving the separation problems. These efficiencies may be incorporated into a PyROS solver call through the associated optional arguments.

Firstly, a user may specify the order in which the PyROS separation problems are to be solved, through the keyword argument `separation_priority_order`. This argument takes the form of a Python dictionary, each entry of which maps the full name of a selected model component object to a number specifying the priority of the corresponding separation problem. Objects eligible for being mapped represent inequality constraints from the set $\mathscr{I}$ and model variables; for the latter, priorities apply to the separation problems associated with their bounds. Priorities can also be designated by annotating the deterministic model with Pyomo suffixes. By default, PyROS assigns a priority value of 0 to separation problems that correspond to model components not mapped by the user. In the PyROS implementation, separation problems are grouped by priority, and the groups are addressed in order of decreasing priority. If a violation is found in any member of a group at iteration $k$, then the uncertain parameter realization $q^{k+1}$ for the next sampled problem is selected from among only the solutions to the members of that group, in a manner otherwise similar to that of Algorithm 1.

Secondly, in alignment with the discussion of Section 3.3.1, separation problems are, by default, first solved to local optimality with the subordinate local optimizer provided to the PyROS solver; the global optimizer is then invoked only if no inequality constraint violations are detected. If the separation problems can be solved efficiently to global optimality, then the user may elect to skip the local optimization tasks by setting the boolean argument `bypass_local_separation` to `True`. Conversely, if the separation problems cannot be tractably solved to global optimality, then the user may opt to skip the global optimization tasks by setting the boolean argument `bypass_global_separation` to `True`. In the latter case, however, PyROS can no longer rigorously certify the robustness of a candidate solution, so upon termination, an appropriate warning is issued to the user. By default, both arguments are set to `False`.

#### 4.3.4 Other Optional Arguments

A user may specify upper bounds on the number of GRCS iterations and the total wall-clock time through the arguments `max_iter` and `time_limit`, respectively. By default, PyROS does not impose an upper bound on either quantity. The relative robust feasibility tolerance (parameter $\varepsilon$ in Algorithm 1) can be explicitly specified by passing the argument `robust_feasibility_tolerance`, for which the default value is $1 \times 10^{-4}$.

If a user anticipates that the subordinate NLP solver(s) provided may encounter a numerical issue in attempting to solve at least one PyROS subproblem, then the user may provide a list of additional local NLP solvers and/or global NLP solvers to invoke in the event the original solver passed to PyROS terminates unsuccessfully, through the optional arguments `backup_local_solvers` and `backup_global_solvers`, respectively.

### 4.4 Solver Output

The PyROS solver is callable through an interface object constructed via the Pyomo `SolverFactory` class. When PyROS is invoked on a model, PyROS returns a "results" object containing the total solver wall clock time, total number of iterations, final objective function value, and the termination status. The possible termination statuses are listed in Table 2. If a robust feasible or optimal solution has been found, then the solution is automatically loaded into the user-provided deterministic (Pyomo) model object.

**Table 2.** PyROS solver termination conditions.

| PyROS Termination Condition | Event in Which Termination Condition is Returned |
|---|---|
| `robust_optimal` | `objective_focus=worst_case` AND `solve_master_globally=True` AND no constraint violations found in separation |
| `robust_feasible` | (`objective_focus=nominal` OR `solve_master_globally=False`) AND no constraint violations found in separation |
| `robust_infeasible` | Sampled problem subsolver terminates with `infeasible` status |
| `max_iter` | Number of elapsed GRCS iterations equals the value of (optional) argument `max_iter` |
| `time_out` | Elapsed time (including time spent by subsolvers) in seconds is no less than the value of (optional) argument `time_limit` |
| `subsolver_error` | User-supplied solver(s) did not return acceptable status for a sampled or separation subproblem (see Appendix B) |

### 4.5 Examples

In this section, we demonstrate the application of the PyROS solver to a few simple optimization problems. We start by seeking an optimal solution to the RO problem [78, Problem (4_3)]

$$\min_{x\in[-10^3,10^3]^3} \; x_1 + \frac{x_2}{2} + \frac{x_3}{3} \tag{16a}$$

$$\text{s.t.} \quad \exp(u-1) - x_1 - x_2 u - x_3 u^2 \leq 0 \quad \forall\, u \in [0,1], \tag{16b}$$

in which $x = (x_1, x_2, x_3)$ is the vector of decision variables and $u$ is the sole uncertain parameter. Problem (16) features a linear objective (16a) in $x$ with no potential uncertainty and an inequality constraint (16b) that is nonlinear in $u$. We take $u^{\text{nom}} = 0.5$ to be the nominal uncertain parameter realization.

Listing 10 shows a script used to solve Problem (16) with PyROS; all variables $x_1, x_2, x_3$ are considered first-stage. We instantiate the PyROS solver through the Pyomo `SolverFactory`. The PyROS solver is invoked through the `solve()` method (lines 21–32) of the solver interface object. Since all decision variables of Problem (16) are first-stage variables, we set the argument `first_stage_variables` to `[m.x1, m.x2, m.x3]`, while an empty list is passed for the `second_stage_variables` argument. The uncertain parameter `m.u` is supplied through the `uncertain_params` argument. The uncertainty set $[0,1]$ is implemented as a one-dimensional `BoxSet` object and is passed via the `uncertainty_set` argument. The NLP solver COUENNE 0.5.8 [79], with default settings, is used as both the local and global subordinate optimizer, through

the `local_solver` and `global_solver` arguments, respectively. Since we wish to solve the model to robust optimality, we opt for a worst-case objective focus and to solve the sampled problems globally, through the `objective_focus` and `solve_master_globally` optional arguments, respectively. Moreover, we opt to bypass solution of the separation problems to local optimality by setting the argument `bypass_local_separation` to `True`. With these settings, PyROS successfully converges to the robust optimal solution $x_1 = 0.3769$, $x_2 = 0.3025$, $x_3 = 0.3206$, with objective value 0.6350, after six iterations.

**Listing 10.** Solving Problem (16), with all variables taken to be first-stage, to robust optimality with PyROS.

```
1  # module imports
2  import pyomo.environ as pyo
3  import pyomo.contrib.pyros as pyros
4
5  # construct the deterministic model
6  m = pyo.ConcreteModel()
7  m.u = pyo.Param(initialize=0.5, mutable=True)
8  m.x1 = pyo.Var(bounds=[-1000, 1000])
9  m.x2 = pyo.Var(bounds=[-1000, 1000])
10 m.x3 = pyo.Var(bounds=[-1000, 1000])
11 m.obj = pyo.Objective(expr=m.x1 + m.x2 / 2 + m.x3 / 3)
12 m.con = pyo.Constraint(
13     expr=pyo.exp(m.u - 1) - m.x1 - m.x2 * m.u - m.x3 * m.u ** 2 <= 0,
14 )
15
16 # instantiate NLP subsolver and PyROS solver
17 nlp_subsolver = pyo.SolverFactory("couenne")
18 pyros_solver = pyo.SolverFactory("pyros")
19
20 # solve with PyROS
21 pyros_solver.solve(
22     model=m,
23     first_stage_variables=[m.x1, m.x2, m.x3],
24     second_stage_variables=[],
25     uncertain_params=[m.u],
26     uncertainty_set=pyros.BoxSet(bounds=[[0, 1]]),
27     local_solver=nlp_subsolver,
28     global_solver=nlp_subsolver,
29     objective_focus=pyros.ObjectiveType.worst_case,
30     solve_master_globally=True,
31     bypass_local_separation=True,
32 )
```

To examine the impact of the decision rule approximation on the robust optimal objective value and computational performance of PyROS for two-stage RO problems, we now consider the case in which $x_1$ is considered a first-stage variable and $x_2, x_3$ are considered second-stage variables. The resulting RO problem can be solved with the script of Listing 10, after lines 21–32 are replaced with:

```
pyros_solver.solve(
    model=m,
    first_stage_variables=[m.x1],
    second_stage_variables=[m.x2, m.x3],
    uncertain_params=[m.u],
    uncertainty_set=pyros.BoxSet(bounds=[[0, 1]]),
    local_solver=nlp_subsolver,
    global_solver=nlp_subsolver,
    objective_focus=pyros.ObjectiveType.worst_case,
    solve_master_globally=True,
    bypass_local_separation=True,
```

```
    decision_rule_order=0,  # can be 0, 1, or 2
)
```

Notice that the `first_stage_variables` and `second_stage_variables` arguments have been specified according to our new variable partitioning, and we have passed a custom value (0, 1, or 2) for the `decision_rule_order` argument.

Table 3 shows the robust optimal objective value and PyROS performance statistics under each of the decision rule approximation schemes supported by PyROS. Observe that:

1. When the decision rule order is set to 0, the robust optimal objective value is equal to the value obtained when all three variables were taken to be first-stage. This is expected, since under a static decision rule policy, the second-stage variables are effectively first-stage variables.

2. The robust optimal objective value is reduced as the the decision rule order is increased. This trend agrees with expectations, as selecting a higher decision rule order enables optimization over a broader range of decision rule policies.

3. The total solve time and iteration requirements increase as the iteration number is increased. In tandem with the previous observation, this means that there is a trade-off between the quality of the solution and the computational cost, which has been observed for polynomial decision rules in [14, 71].

**Table 3.** Robust optimal objective values and PyROS performance statistics for Problem (16) with $x_1$ considered a first-stage variable, and $x_2, x_3$ considered second-stage variables, subject to the three decision rule order options available in PyROS.

| Decision Rule Order | Robust Optimal Objective Value | Solver Wall-Clock Time (s) | Number of Iterations |
|---|---|---|---|
| 0 | 0.6350 | 1.4 | 6 |
| 1 | 0.6292 | 11.9 | 11 |
| 2 | 0.6280 | 16.5 | 12 |

To illustrate the capability of PyROS to solve nonconvex RO problems with uncertain equality constraints, we now study the optimization problem

$$\min_{x \in [-10^3, 10^3]^3} \; x_1 + \frac{x_2}{2} + \frac{x_3}{3} \tag{17a}$$

$$\text{s.t.} \quad \exp(u-1) - x_1 - x_2 u - x_3 u^2 \leq 0 \qquad \forall u \in [0,1] \tag{17b}$$

$$u^2(x_2 - 1) + u(x_1^3 + 0.5) - 5u x_1 x_2 + u(x_1 + 2) = 0 \; \forall u \in [0,1] \tag{17c}$$

which is just Problem (16) augmented with the nonlinear equality constraint (17c). All variables are considered first-stage, and as in the previous examples, we take $u^{\text{nom}} = 0.5$. A robust optimal solution can be obtained with the PyROS solver by running the script of Listing 10, with the addition of the equality constraint declaration:

```
model.eq_con = pyo.Constraint(
    expr=(
        model.u ** 2 * (model.x2 - 1)
```

```
        + model.u * (model.x1 ** 3 + 0.5)
        - 5 * model.u * model.x1 * model.x2
        + model.u * (model.x1 + 2)
        == 0
    ),
)
```

Running the resulting script, we find that PyROS successfully converges after just two iterations to the robust optimal solution $x_1 = 0.7172, x_2 = 1, x_3 = -0.7172$, with objective function value 0.9782. Recall from our previous examples that the robust optimal objective value of (16) with all variables considered first-stage was found to be 0.6350. Hence, the introduction of the equality constraint (17c) induces a 54% increase in the robust optimal objective value.

The application of the equality constraint reformulation approach of Section 3.3.4 is the reason only two PyROS iterations are required to solve (17) to robust optimality. Observe that the equality constraint (17c) does not involve state variables and can be rewritten as

$$(x_2 - 1)u^2 + (x_1^3 - 5x_1x_2 + x_1 + 2.5)u = 0, \tag{18}$$

which bears the standard form of (15). That is, (17c) can be written as a polynomial in the uncertain parameter $u$, with coefficients that depend only on the first-stage variables $x$. Using the polynomial coefficient matching approach of Section 3.3.4, observe that (18) is satisfied for all $u \in [0, 1]$ provided that the constraints

$$x_1^3 - 5x_1x_2 + x_1 + 2.5 = 0 \tag{19a}$$

$$x_2 - 1 = 0 \tag{19b}$$

are satisfied. From (19b), we immediately deduce that $x_2 = 1$. Substituting into (19a) yields a cubic equation in $x_1$, of which the roots are approximately equal to $-2.2597$, 0.7172, and 1.5425; that is, at every iteration of the sampled problem, $x_1$ is to be chosen from one of these three values. This significantly restricts the feasible space of the sampled problems from their very first instantiation, leading the PyROS solver to quickly converge to the robust optimal solution.

# 5 Numerical Results

## 5.1 Test Set

A library of 8591 benchmark RO model instances is derived from the ten deterministic models described in Table 4. The base models are chosen to represent an array of nonlinear characteristics, outlined in the rightmost column, and vary by the number of degrees of freedom, model parameters, and state variables. Python scripts used to build the base models are provided in the Electronic Supplement.

Each RO model instance is characterized by a base model, a partitioning of the degrees-of-freedom into first-stage and second-stage variables, a subset of the model parameters to be considered uncertain, an uncertainty set type and parameterization, and a decision rule order specification. Table 5 demonstrates these characterizations for a few example instances. In Appendix A, we articulate the procedure for generating the library. Note that, for all instances, each uncertain parameter appears exclusively as the coefficient of exactly one term of either the objective function or exactly one constraint function.

**Table 4.** Deterministic models from which the benchmark RO model instances are derived. The model `haverly` is taken from the GAMS Model Library [80], and all other models are taken from the PrincetonLib collection of the GAMS World model library [81]. The constraint counts include any variable bounds defined during variable declarations.

| Model Name | Number of Model Components | | | | Nonconvex Model | Notes on Nonlinearities |
|---|---|---|---|---|---|---|
| | Variables | State Variables | Inequality Constraints | Equality Constraints | | |
| `himmelp6` | 2 | 0 | 9 | 0 | ✓ | Exponential and polynomial objective, quadratic constraints |
| `s353` | 5 | 2 | 6 | 2 | ✓ | Linear objective, quadratic equality constraints |
| `lewispol` | 7 | 3 | 20 | 3 | ✓ | Quadratic objective, cubic inequalities |
| `haverly` | 12 | 7 | 12 | 7 | ✓ | Linear objective, bilinear terms in equality constraints |
| `s381` | 13 | 1 | 16 | 1 | | Linear model |
| `s382` | 13 | 1 | 16 | 1 | | Linear objective, square roots of quadratic forms in inequalities |
| `hydro` | 31 | 19 | 54 | 19 | ✓ | Quadratic objective and constraints |
| `optcntrl` | 32 | 23 | 31 | 23 | ✓ | Quadratic objective and constraints |
| `hydrothermal` | 52 | 38 | 92 | 38 | ✓ | Quadratic objective, quartic equality constraints |
| `optmass` | 70 | 48 | 11 | 48 | ✓ | Quadratic objective, quadratic inequality constraints |

**Table 5.** Example instance information for derived benchmark problems from base problem `s381`.

| Base Model Name | Uncertainty Set Type | Uncertainty Set Description | First-Stage Variables | Second-Stage (DOF) Variables | Uncertain Parameters | Decision Rule Order |
|---|---|---|---|---|---|---|
| `s381` | `BoxSet` | ±15% of nominal | $x_1, x_2, x_3$ | $x_4, \ldots, x_{12}$ | $p_0, p_1$ | 0 |
| `s381` | `BoxSet` | ±15% of nominal | $x_1, x_2, x_3$ | $x_4, \ldots, x_{12}$ | $p_0, p_1$ | 1 |
| `s381` | `BoxSet` | ±15% of nominal | $x_1, x_2, x_3$ | $x_4, \ldots, x_{12}$ | $p_0, p_1$ | 2 |
| | | | ⋮ | | | |
| `s381` | `DiscreteScenarioSet` | 20 scenarios | | $x_1, \ldots, x_{12}$ | $p_0, p_1, p_2, p_3$ | 0 |
| `s381` | `DiscreteScenarioSet` | 20 scenarios | | $x_1, \ldots, x_{12}$ | $p_0, p_1, p_2, p_3$ | 1 |
| `s381` | `DiscreteScenarioSet` | 20 scenarios | | $x_1, \ldots, x_{12}$ | $p_0, p_1, p_2, p_3$ | 2 |

## 5.2 Computational Setup

All two-stage RO models are instantiated and solved in a Python 3.12.11 virtual environment that includes Pyomo 6.10.0dev0/PyROS 1.3.13. Each instance is solved on a single core of an Intel® Xeon® Gold 5215 CPU @ 2.50 GHz, running Ubuntu Linux 20.04 with 125 GB RAM. Benchmark instances derived from all but the `hydrothermal` and `optmass` deterministic base models are solved to robust optimality with the subordinate solver BARON 25.11.17/CPLEX 22.1.0 [82, 83]. A robust feasibility tolerance of $1 \times 10^{-4}$ and a wall-clock time limit of 400 s are imposed on the PyROS solver for each instance. The local optimization of the separation problems is bypassed; that is, the option `bypass_local_separation=True` is specified. Benchmark intances derived from the `hydrothermal` and `optmass` deterministic models are solved with a robust feasibility target, by opting to solve the sampled problems to local optimality with the subordinate local solver IPOPT 3.14.6/HSL MA57 [84, 85]; the IPOPT option `constr_viol_tol` is lowered from its default value of $1 \times 10^{-4}$ to a value of $1 \times 10^{-6}$, which is the default constraint feasibility tolerance for the BARON solver. Our decision to solve to local optimality the sampled problems for instances

derived from `hydrothermal` and `optmass` is informed by the relatively high computation time requirements for the global solution of the corresponding deterministic models and sampled problems with BARON 25.11.17 and/or recent preceding releases.

## 5.3 Performance and Reliability Statistics

### 5.3.1 Overview

Our computational results show that most of the instances in the test set are solved successfully, meaning that PyROS within the allotted time limit obtains either a robust optimal solution, a robust feasible solution, or a proof of robust infeasibility. Indeed, of the 8591 instances tested, a total of 7060 (approximately 82.2%) are solved successfully.

More specifically, as the empirical distribution plots of Figure 2 suggest, approximately 79.1% of all instances are solved to robust optimality or robust feasibility. Of those instances, approximately 87.9% are solved in under 10 s of wall-clock time, and approximately 98.3% are solved in under 100 s of wall-clock time. Although, in the worst case, over 100 iterations may be required for solution to robust optimality or feasibility, over 61% of the instances that are solved to robust optimality or feasibility require 3 or fewer iterations, and over 93% require 10 or fewer iterations. A total of 631 instances (approximately 7.3% of the entire test set) are solved after only one iteration. Indeed, all of the 605 instances in the library for which the uncertainty set is the singleton (implemented as a `BoxSet` with equal lower and upper bounds, i.e., $\mathcal{Q} := [q^{\text{nom}}, q^{\text{nom}}] = \{q^{\text{nom}}\}$) are among the 631 instances solved after just one iteration, as expected.

Our computational results also suggest that, for the present set of model instances, the computational requirements for proof of robust infeasibility are relatively small. As Figure 3 shows, robust infeasibility is proven for approximately 3.1% of the tested instances; this comprises a small but noticeable portion of the test set. Of the instances proven to be robust infeasible, over 94% are solved in under 10 s. Moreover, while over 70% of the instances proven robust infeasible are solved after no more than 3 iterations, there exist a few that require over 10 iterations in order to converge to the robust infeasibility result.

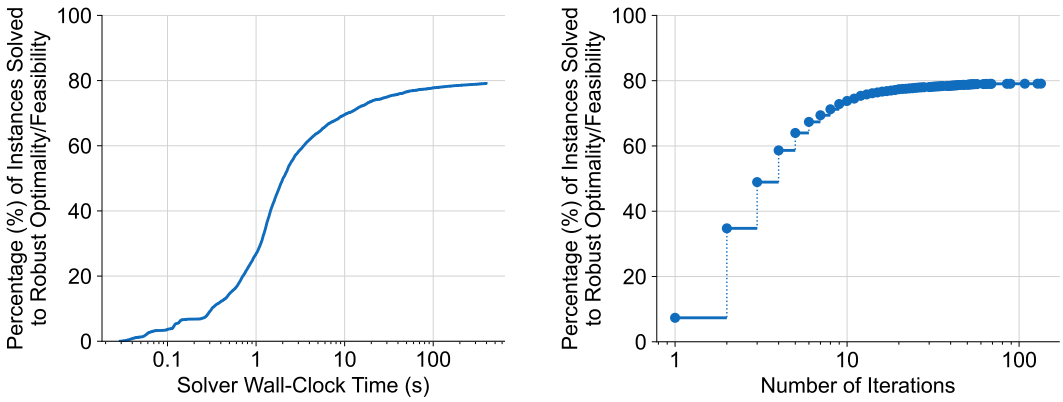


**Figure 2.** Empirical cumulative distribution functions of the solve time and iteration requirements for instances solved to robust optimality or robust feasibility.

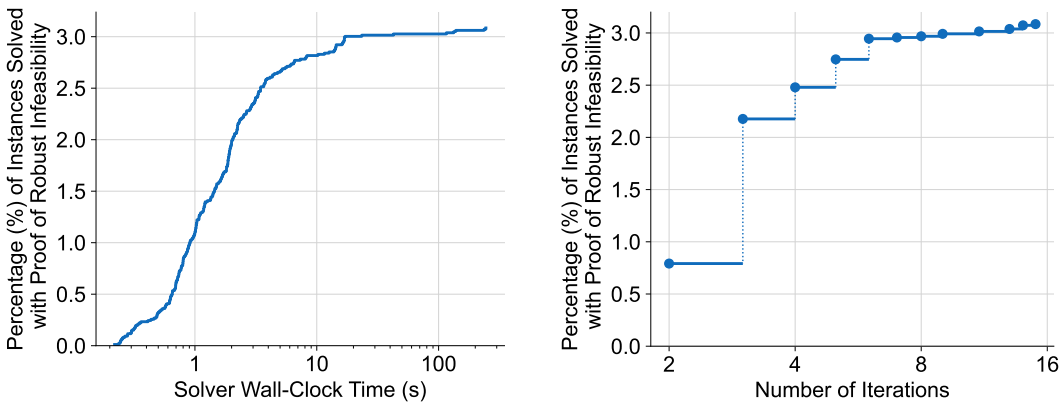


**Figure 3.** Empirical cumulative distribution functions of the solve time and iteration requirements for instances proven robust infeasible.

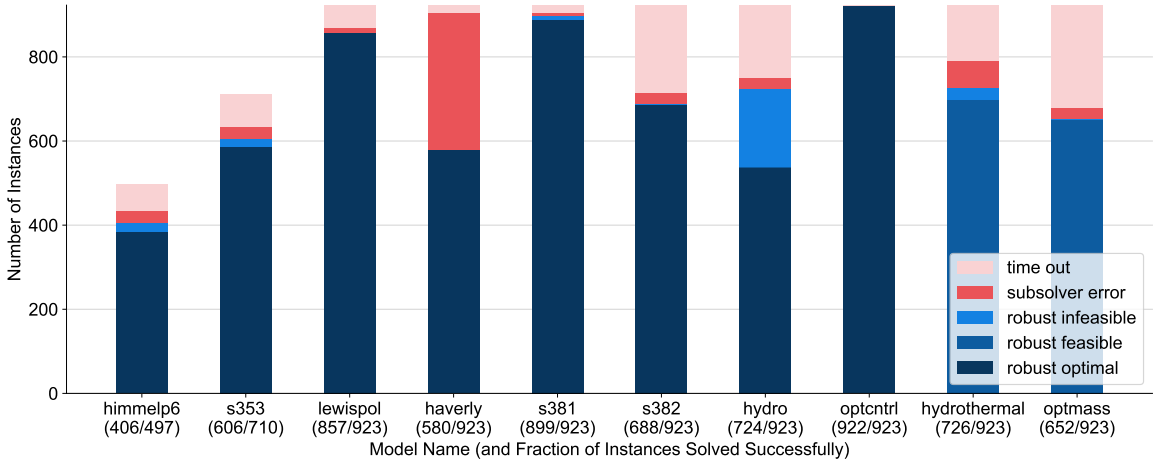


**Figure 4.** Termination statistics for all instances of the computational study, grouped by deterministic model. For each group, the fraction of RO instances that were solved successfully is given in parentheses. Going from left to right, the deterministic models are sorted in order of increasing variable dimension, per Table 4.

### 5.3.2 Response to Deterministic Model

The reliability statistics of Figure 4 show that, for each deterministic model, over 62% of the derived RO model instances are successfully solved to robust optimality, robust feasibility, or proven to be robust infeasible. Nevertheless, across deterministic models, we find considerable variation in the fraction of instances solved successfully. The fraction of successfully solved derived instances is lowest for the model `haverly`, for which only 62.8% of the derived instances are solved successfully, followed by `optmass`, for which only 70.6% of the derived instances are solved successfully. In contrast, 99.9% of the instances derived from `optcntrl` are solved successfully.

For a given RO model instance, a timeout may occur due to the anomalously large computational cost of a single particular subproblem or to a high number of PyROS iterations required to solve the instance to an acceptable status. In our study, we found the causes of timeouts to largely correlate with the deterministic model on which the instances that terminate with a timeout status

are based. In particular, timeouts for instances derived from `s381` and `hydrothermal` are predominantly a result of number of PyROS iterations. In contrast, timeouts for instances derived from `himmelp6` are predominantly a result of the cost of solving a single particular sampled or decision rule polishing problem, whereas timeouts for instances derived from `s353`, `lewispol`, `haverly`, `s382`, `hydro`, and `optmass` are predominantly found to arise from the cost of a single particular separation problem. Interestingly, no timeouts are observed for instances derived from base model `optcntrl`.

For instances that terminate with a subsolver error status, the error more often stems from separation problem solution failures than from sampled problem solution failures; in particular, termination due to separation problem failures occurs for 78.1% of all such instances. The preponderance of separation problem failures possibly stems from the preponderance of variables that are not (explicitly) bounded in the separation problems; see ($\text{SEP}_i$), in which the second-stage and state variables $(z, y)$ are restricted only by the equality constraints (9b)–(9c). There are, nevertheless, some deterministic models for which the derived RO instance subsolver error status stems mainly from sampled problems. For example, a sampled problem failure was found in all of the reported instances derived from `s353` and `optmass`.

The empirical distribution plots of Figures 5 and 6 suggest that the computational requirements vary with the base model, but are not necessarily dependent on the size of the model. Figure 5, in which the deterministic models are sorted in order of increasing variable dimension along the x-axis, shows no clear trend in the median solve time requirement. Similarly, Figure 6 shows no clear trend in the median iteration requirement. Nevertheless, the solve time requirements tend to be greatest and most variable for the instances derived from `hydro` and `hydrothermal`, which (according to Table 4) have the highest numbers of inequality constraints and therefore tend to yield the most separation problems that are to be solved in every PyROS iteration. Central variations in the iteration requirements for instances derived from `s381`, `hydrothermal`, and `optmass` are also considerable.

As Figure 6 reveals, very few iterations are required to solve the instances derived from `haverly` or `optcntrl`. In particular, aside from instances corresponding to singleton uncertainty sets, which always require exactly one iteration to solve, all other instances of `haverly` require no more than three PyROS iterations, since the nonlinear cuts added after the first one or two iterations are sufficient for finding the robust optimal solution through solving the terminal sampled problem. Similar behavior is observed for the instances derived from `optcntrl`, in which the uncertain parameters appear only in the objective function; in particular, all such instances require at most two PyROS iterations to converge.

### 5.3.3 Response to Uncertainty Set Type

The reliability statistics of Figure 7 show that, for each uncertainty set type, over 65% of the derived RO instances are solved successfully to robust optimality, robust feasibility, or proven robust infeasible. However, the success rate depends on the uncertainty set type. Success rates for the discrete and box set instances are the highest, at 98.8% and 91.1%, respectively. In contrast, success rates for the intersection, ellipsoidal, and factor model set instances are the lowest, at 65.7%, 67.8%, and 71.4%, respectively. The weaker performance on the instances with ellipsoidal and intersection sets likely stems from the nonlinear, albeit convex, constraints defining the ellipsoidal sets, and thus, the increased difficulty of the separation problems; note that the intersection sets used in the study

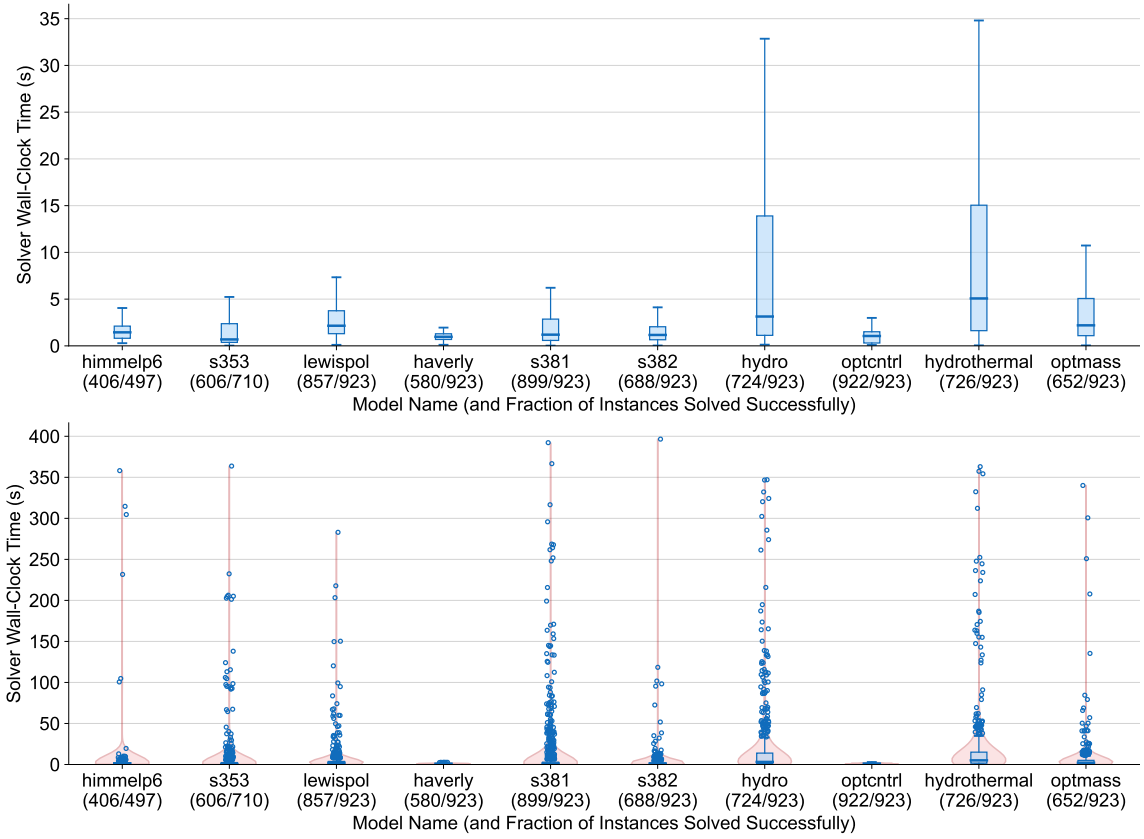


**Figure 5.** Central tendency box plots (top) and full range box-and-violin plots (bottom) of empirical solve time distributions for all instances of the computational study that were solved successfully. Instances are grouped by deterministic model. For each group, the fraction of RO instances that were solved successfully is given in parentheses. Going from left to right, the deterministic models are sorted in order of increasing variable dimension, per Table 4.

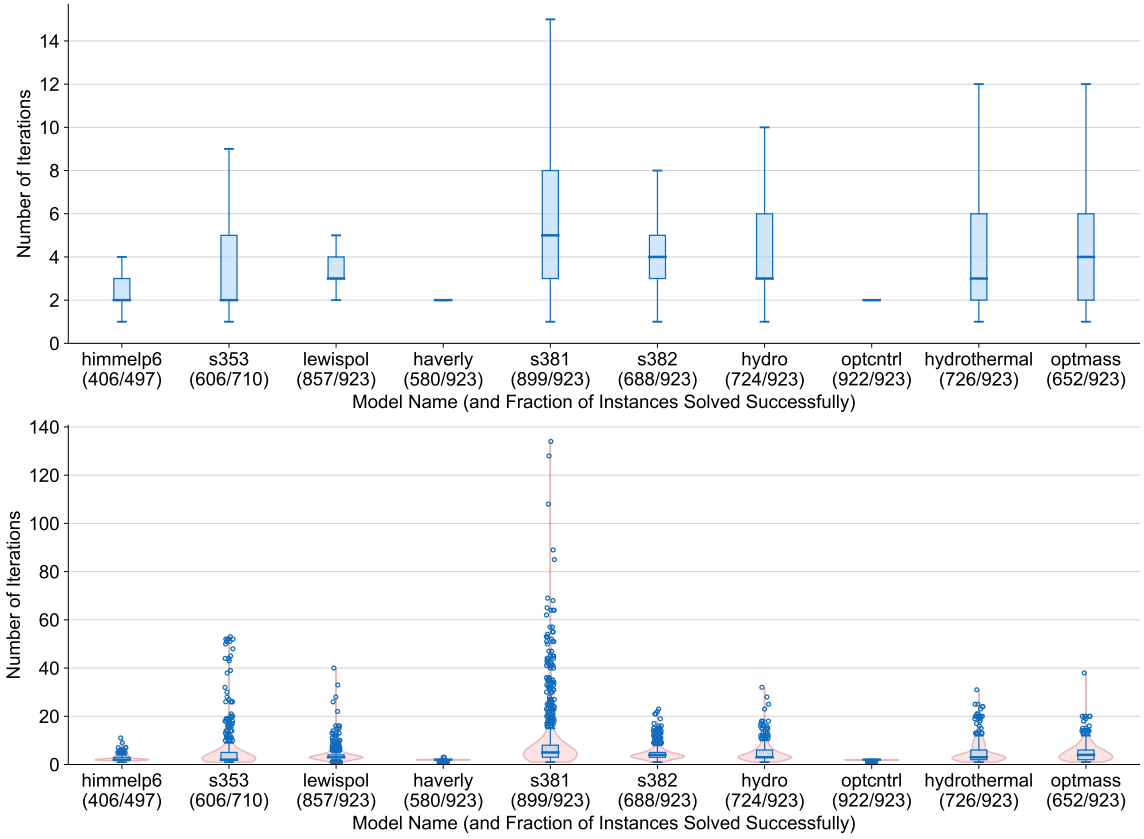


**Figure 6.** Central tendency box plots (top) and full range box-and-violin plots (bottom) of empirical iteration requirement distributions for all instances of the computational study that were solved successfully. Instances are grouped by deterministic model. For each group, the fraction of RO instances that were solved successfully is given in parentheses. Going from left to right, the deterministic models are sorted in order of increasing variable dimension, per Table 4.

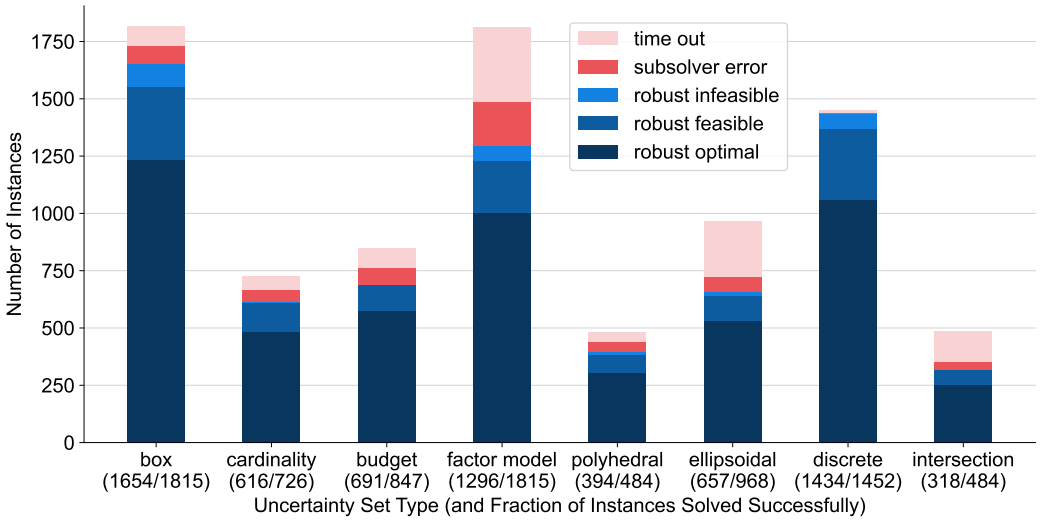


**Figure 7.** Termination statistics for all instances of the computational study, grouped by uncertainty set type. For each group, the fraction of RO instances that were solved successfully is given in parentheses.

also involve ellipsoidal constraints. In contrast, all other uncertainty sets are either discrete or polyhedral. The weaker performance on the RO model instances subject to the factor model set likely stems from the relatively high numbers of variables in the separation problems. Whereas every factor model set instance used in the study is five-dimensional, each of the other set types has instances of dimension ranging from either 1 to 5 or 2 to 5. Moreover, as shown in Table 1, the constraints defining a factor model set involve the introduction of additional auxiliary variables and, as outlined in Appendix A, each factor model set in the present study involves 3–5 such additional variables.

The performance statistics visualized in Figures 8 and 9 show that the (average) solve time and iteration requirements may depend on the uncertainty set type. The solve time and iteration requirements are, on average, the lowest among instances with a box uncertainty set. This may be due to the inclusion of box sets that are 1D or singleton (see Appendix A) and the relatively simple geometry of the box sets. In contrast, the solve time and iteration requirements for instances with an ellipsoidal uncertainty set are, on average, the highest among all uncertainty set types; additionally, there is considerably more variation in the requirements. The higher requirements for the ellipsoidal set-based instances may follow from the nonlinear geometries of the ellipsoidal sets, which increase the complexity of the separation problems, and/or the fact that an ellipsoidal set features an infinite number of extreme points.

### 5.3.4 Response to Decision Rule Order

Figures 10 and 11 show that, although for all decision rules most instances are successfully solved, the reliability and performance levels decrease as the order of the decision rules is increased. Considering only instances for which there is at least one second-stage degree-of-freedom variable, so that our analysis more carefully assesses the impact of the decision rule order, we observe that, of the reported instances with static (order 0) decision rules, 93.6% are solved successfully. In contrast, only 76.6% and 73.1% of instances with affine (order 1) and quadratic (order 2) decision rules, respectively, are solved successfully. Although the average iteration requirements are comparable

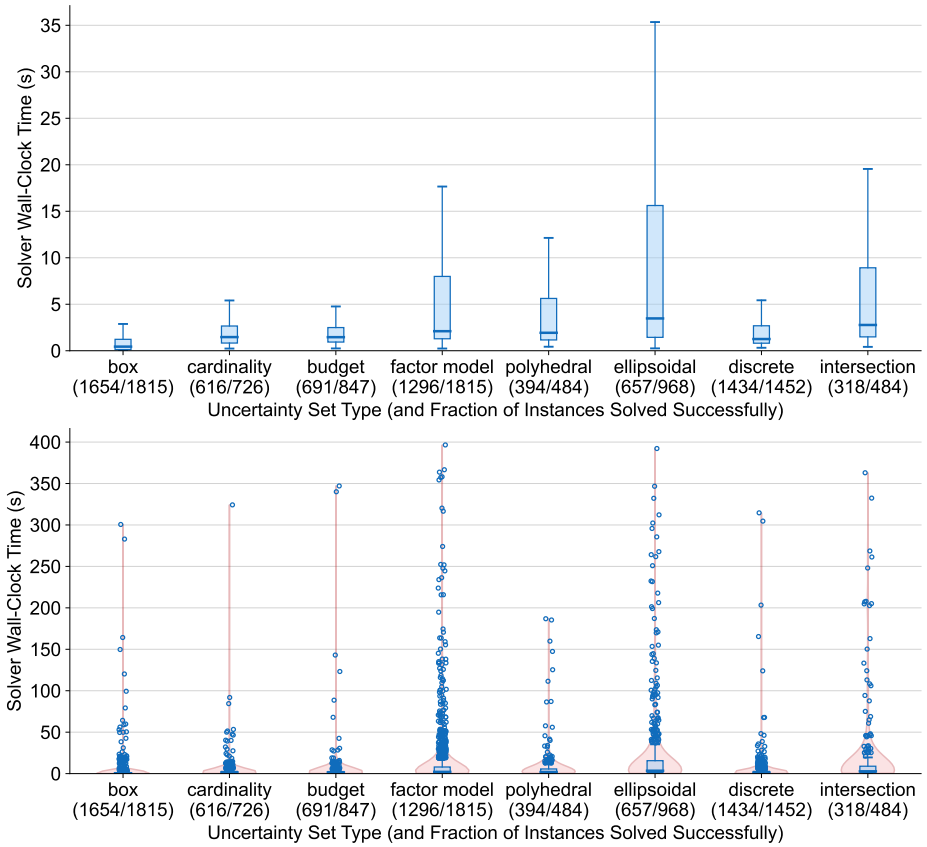


**Figure 8.** Central tendency box plots (top) and full range box-and-violin plots (bottom) of empirical solve time distributions for all instances of the computational study that were solved successfully. Instances are grouped by uncertainty set type. For each group, the fraction of RO instances that were solved successfully is given in parentheses.

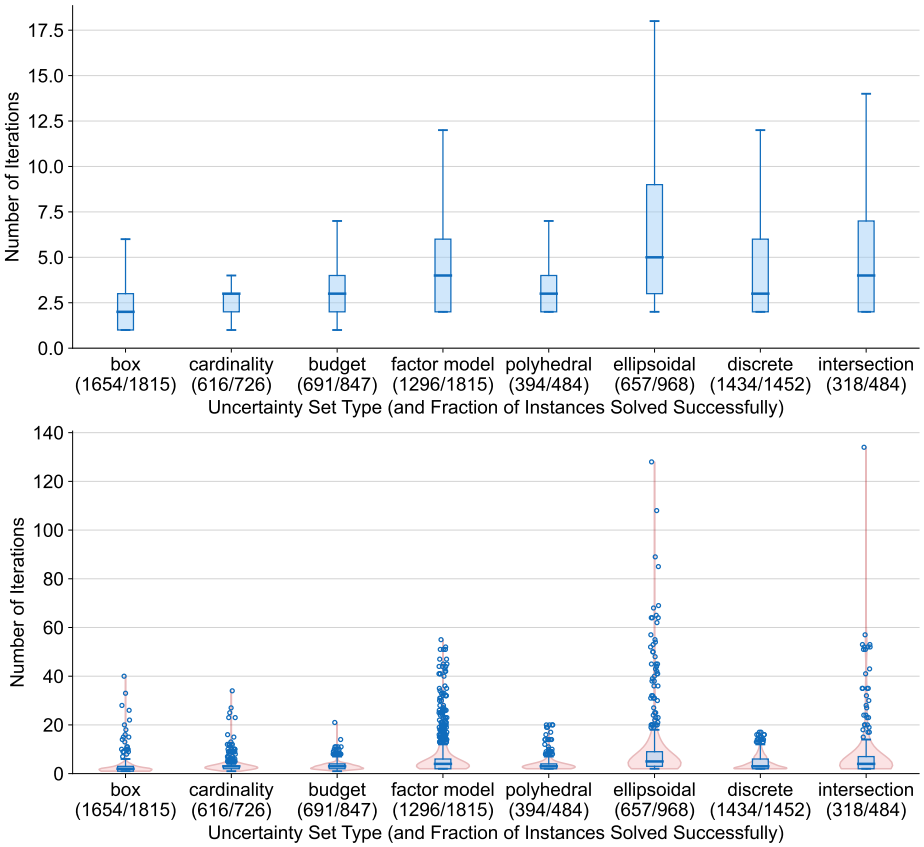


**Figure 9.** Central tendency box plots (top) and full range box-and-violin plots (bottom) of empirical iteration requirement distributions for all instances of the computational study that were solved successfully. Instances are grouped by uncertainty set type. For each group, the fraction of RO instances that were solved successfully is given in parentheses.

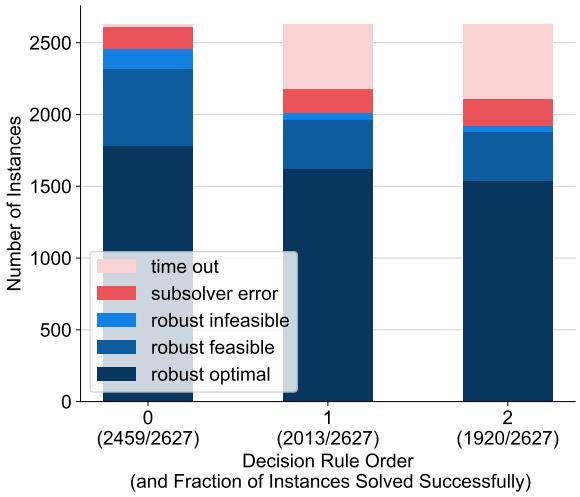


**Figure 10.** Termination statistics for all instances of the computational study, excluding those without second-stage variables. Instances are grouped by decision rule order. For each group, the fraction of RO instances that were solved successfully is given in parentheses.

for instances solved to robust optimality or feasibility, when the decision rules are non-static, the solve time requirements are higher and more diverse. Either way, the trends shown in Figures 10 and 11 are as expected, as higher-order decision rules are more complex; see Section 2.3. Similar performance trends were also observed in Section 4.5 and [14, 71].

## 6 Conclusions

We have presented PyROS, a Python-based meta-solver for nonconvex two-stage RO problems. The meta-solver's interface design and compatibility with the widely used open-source Pyomo AML guarantee its accessibility and ease of use. We have showcased the various options available to the user, including the uncertainty set specification, objective focus, and decision rule approximations for adjustability of the second-stage variables. Through a comprehensive benchmarking study, we have shown that PyROS can be an effective tool for identifying solutions to nonconvex two-stage RO problems across a broad range of uncertainty set types. The automatic RO capabilities of PyROS allow owners of Pyomo models to seamlessly extend their deterministic optimization workflows into RO workflows and insure their solutions against parametric uncertainties inherent in their models. Furthermore, PyROS allows users to readily solve hierarchies of RO problems and investigate the effects of the uncertainty set characterization and the adaptivity specifications on the quality of the robust solutions.

## Acknowledgements

This work was supported the United States Department of Energy's Hydrocarbons and Geothermal Energy Office, through the Institute for Design of Advanced Energy Systems (IDAES) and the Carbon Capture Simulation for Industry Impact (CCSI$^2$) initiative. Natalie M. Isenberg also

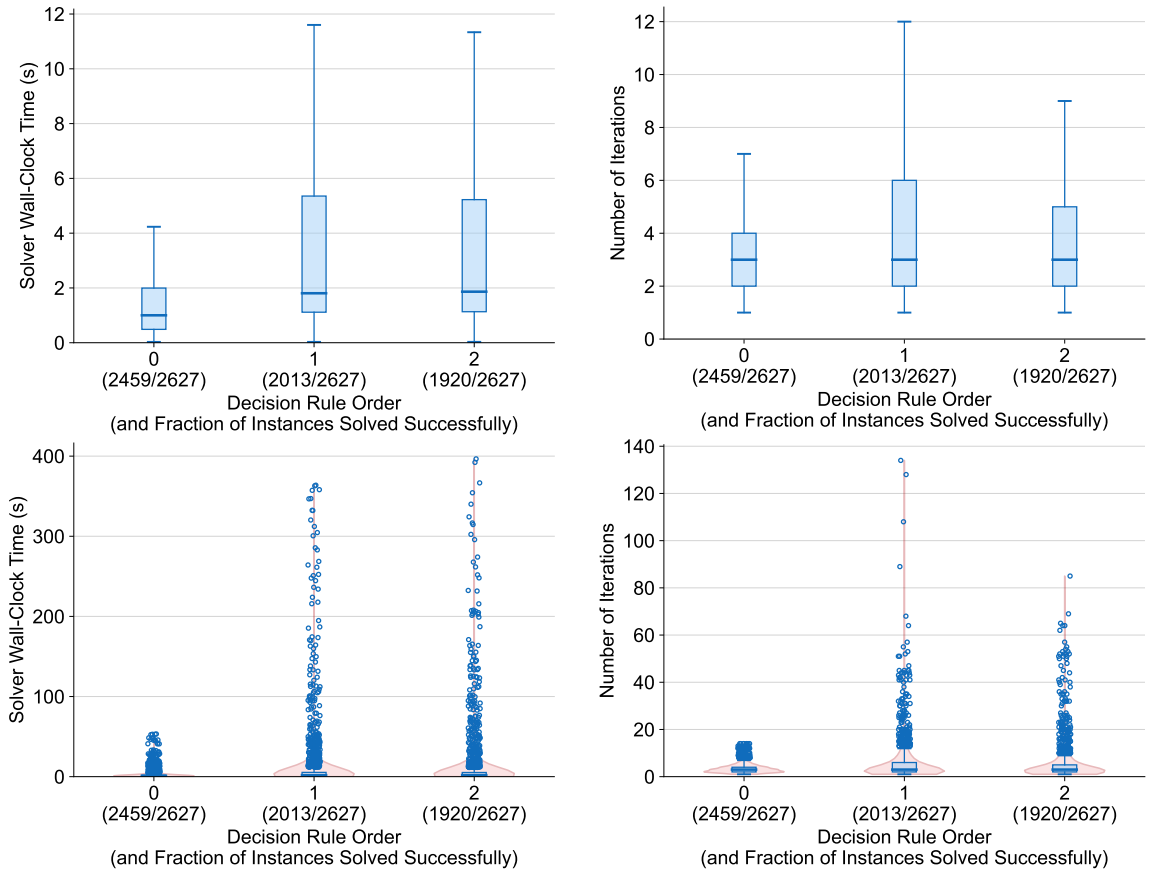


**Figure 11.** Central tendency box plots (top) and full range box-and-violin plots (bottom) of empirical solve time and iteration requirement distributions for all instances of the computational study that were solved successfully, excluding those without second-stage variables. Instances are grouped by decision rule order. For each group, the fraction of RO instances that were solved successfully is given in parentheses.

acknowledges support from a U.S. Department of Energy, Office of Science Graduate Student Research award.

## Disclaimer



## Conflict of interest

The authors declare that they have no conflict of interest.

## A Procedure for Generation of the Two-Stage Model Library

We are unaware of a publicly available library of nonlinear two-stage RO problems for benchmarking RO algorithms. For this reason, we have constructed the library of benchmarks outlined in Section 5, on the basis of the deterministic NLP models listed in Table 4. In this section, we articulate the procedure for extending the deterministic models to a library of two-stage RO model instances.

### A.1 Base Models and Degree-of-Freedom Partitioning

For each base model, we perform a structural analysis of the base model's system of equality constraints to determine the degree-of-freedom variables. Supposing there are $M$ such variables, we consider the cases in which the first $0, \lfloor M/4 \rfloor, \lfloor M/2 \rfloor, \lfloor 3M/4 \rfloor$, and $M$ variables are considered to be first-stage (avoiding repeats), and the remaining variables are considered second-stage. See the

Electronic Supplement for the exact selection and classification of the degrees of freedom of each of the base models.

### A.2 Uncertain Parameters

For each base model, a total of five parameters $(p_0, p_1, p_2, p_3, p_4)$ are chosen to be potentially uncertain. Our parameter selections are based on the results of a local sensitivity analysis study. In particular, we chose the parameters which, upon deviating from their nominal values, induced the greatest combined response of the optimal objective function value and robust infeasibility of the nominally optimal degree-of-freedom variable values. Readers are referred to the Electronic Supplement for our specific choices of uncertain parameters. It should be noted that the uncertain parameters are implemented as normalization factors (of nominal value 1) for the coefficients, rather than the original, hard-coded coefficients themselves.

### A.3 Uncertainty Sets

We allow uncertainty sets represented by all the pre-implemented PyROS `UncertaintySet` subclasses, and we work with a hierarchy of parameterizations of each uncertainty set type. Table 6 defines the hierarchy used for each set type. For a set of dimension $n \in \{1,2,3,4,5\}$, the first $n$ of the five model parameters identified in Section A.2 are considered uncertain.

1. Coefficient matrices used for the `FactorModelSet` instances are sub-matrices of the matrix $\Phi \in \mathbb{R}^{5\times 5}$ defined through

$$\Phi_{ij} = 10^{-3}\left\lfloor 10^3(0.2+0.1v_{i1})\frac{v_{i,j+1}}{\sum_{\ell=1}^{5} v_{i,\ell+1}}\right\rfloor \qquad \forall i \in \{1,2,3,4,5\}, \forall j \in \{1,2,3,4,5\}, \quad (20)$$

   in which the values $v_{i\ell}$ are drawn from the continuous uniform distribution $U(0,1)$. Hence, $\Phi$ is such that all entries $\Phi_{ij}$ are rounded to the nearest thousandth, and the sum of the entries of each row of $\Phi$ is a value between 0.2 and 0.3. For reproducibility, the matrix $\Phi$ has been hard-coded into the scripts included in the Electronic Supplement.

2. The $n$-dimensional `PolyhedralSet` instance used in the study is the regular simplex with centroid $e_n$ and vertices $\bar{q}^1, \bar{q}^2, \ldots, \bar{q}^{n+1} \in \mathbb{R}^n$ given by

$$\bar{q}^i = \frac{0.15}{\sqrt{2(n+1)}} R\bar{v}^i + e_n \qquad i = 1,2,\ldots,n+1, \quad (21)$$

   where $e_n$ is the $n$-dimensional all-ones vector; $R$ is an $n \times n$ real matrix given by

$$R_{ij} = \begin{cases} 1 + \frac{1}{\sqrt{n}(\sqrt{n}+1)} & i = j < n \\ \frac{1}{\sqrt{n}(\sqrt{n}+1)} & i,j < n \text{ and } i \neq j \\ \frac{1}{\sqrt{n}} & j = n \\ -\frac{1}{\sqrt{n}} & \text{otherwise,} \end{cases} \quad (22)$$

for $i, j = 1, 2, \ldots, n$; and the point $\bar{v}^i$ is given by

$$\bar{v}^i_j = \begin{cases} -\frac{1}{\sqrt{2j(j+1)}} & j > i-1 \\ \sqrt{\frac{j}{2(j+1)}} & j = i-1 \\ 0 & \text{otherwise,} \end{cases} \tag{23}$$

for $i = 1, 2, \ldots, n+1$ and $j = 1, 2, \ldots, n$.

3. The shape matrix $\Sigma$ for the $n$-dimensional `EllipsoidalSet` instance is obtained by applying successive 45°-counterclockwise, two-dimensional planar rotations to an axis-aligned ellipsoidal region with center $q^0 = e_n$ and preset half-lengths. For $i, j = 1, 2, \ldots, n$, applying such a rotation in the $q_i q_j$-plane involves a similarity transformation of the ellipsoidal region's shape matrix by the rotation matrix $\Lambda^{ij}$ defined through

$$\Lambda^{ij}_{i_1,i_2} = \begin{cases} \cos 45^\circ & i_1 = i_2 \in \{i, j\} \\ -\sin 45^\circ & i_1 = i \text{ and } i_2 = j \\ \sin 45^\circ & i_1 = j \text{ and } i_2 = i \\ 1 & i_1 = i_2 \notin \{i, j\} \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

The specific half-lengths and rotations used in the present study are as follows:

- For $n = 2$, we apply a rotation in the $q_1 q_2$-plane to the axis-aligned ellipsoidal region with half-lengths 0.3 and 0.1.
- For $n = 3$, we apply a rotation in the $q_1 q_2$-plane then the $q_1 q_3$-plane to the axis-aligned ellipsoidal region with half-lengths 0.3, 0.2, and 0.1.
- For $n = 4$, we apply rotations in the $q_1 q_2$-plane, $q_1 q_3$-plane, then the $q_2 q_3$-plane to the axis-aligned ellipsoidal region with half-lengths 0.3, 0.2, 0.2, and 0.1.
- For $n = 5$, we apply rotations in the $q_1 q_2$-plane, $q_1 q_3$-plane, $q_2 q_3$-plane, then the $q_1 q_4$-plane to the axis-aligned ellipsoidal region with half-lengths 0.3, 0.25, 0.2, 0.15, and 0.1.

4. Scenarios for the `DiscreteScenarioSet` instances used in this study are based on the randomly generated points $\theta^1, \theta^2, \ldots, \theta^{20}$, defined through

$$\theta^s_i \sim U(0.7, 1.3) \qquad \forall i \in \{1, 2, 3, 4, 5\}, \forall s \in \{1, 2, \ldots, 20\}. \tag{25}$$

For reproducibility, the points have been hard-coded into the scripts included in the Electronic Supplement.

**Table 6.** Parameterizations for the uncertainty sets used in the benchmarking study, using the notation of Table 1.[a]

| `UncertaintySet` Subclass | Set Description | Parameterizations for Sets of Dimension $n$ |
|---|---|---|
| `BoxSet` | singleton ($c=0$), small ($c=0.15$), large ($c=0.3$) | $q^{\mathrm{L}} = (1-c)e_n, \quad q^{\mathrm{U}} = (1+c)e_n, \quad n \in \{1,2,3,4,5\}$ |
| `CardinalitySet` | larger $\Gamma$ | $q^0 = e_n, \quad \hat{q} = 0.3\,e_n, \quad \Gamma = \left\lceil \frac{n}{2} \right\rceil, \quad n \in \{2,3,4,5\}$ |
| | smaller $\Gamma$ | $q^0 = e_n, \quad \hat{q} = 0.3\,e_n, \quad \Gamma = \left\lceil \frac{n}{3} \right\rceil, \quad n \in \{3,5\}$ |
| `BudgetSet` | single budget | $q^0 = e_n, \quad L = 1, \quad b_1 = 0.3, \quad B = e_n^{\mathsf{T}}, \quad n \in \{2,3,4,5\}$ |
| | multiple budgets | $q^0 = e_n, \quad L = \binom{n}{2}, \quad b_\ell = 0.3 \text{ and } B_{\ell i} = \begin{cases} 1 & i \in \mathscr{B}^n_{\ell,2} \\ 0 & \text{otherwise} \end{cases} \quad \forall \ell \in \{1,2,\ldots,\binom{n}{2}\}, \quad n \in \{3,4,5\}$ |
| `FactorModelSet` | coefficient matrix $\Phi$ defined in Eq. (20) | $q^0 = e_n, \quad \Psi_{ij} = \Phi_{ij} \quad \forall i \in \{1,2,\ldots,n\} \quad \forall j \in \{1,2,\ldots,F\}, \quad \beta \in \{0, 0.25, 0.5, 0.75, 1\}, \quad F \in \{3,4,5\}, \quad n = 5$ |
| `PolyhedralSet` | simplex | $A, b$ specify the half-space representation of the regular simplex with vertices defined in Eq. (21) |
| `AxisAlignedEllipsoidalSet` | hyperball | $q^0 = e_n, \quad a = 0.2\,e_n, \quad n \in \{2,3,4,5\}$ |
| `EllipsoidalSet` | rotated ellipsoid | $q^0 = e_n, \quad \Sigma$ obtained using method outlined in text, $\quad s = 1, \quad n \in \{2,3,4,5\}$ |
| `DiscreteScenarioSet` | scenarios $\theta^1, \theta^2, \ldots, \theta^{20}$ defined in Eq. (25) | $\vartheta_i^s = \theta_i^s \quad \forall i \in \{1,2,\ldots,n\} \quad \forall s \in \{1,2,\ldots,S\}, \quad S \in \{10, 15, 20\}, \quad n \in \{2,3,4,5\}$ |
| `IntersectionSet` | box and hyperball | $m = 2, \quad \mathscr{Q}_1 :=$ small `BoxSet` and $\mathscr{Q}_2 :=$ `AxisAlignedEllipsoidalSet` (as above), $\quad n \in \{2,3,4,5\}$ |

[a] Note that $e_n$ denotes the $n$-dimensional vector of ones, while $\mathscr{B}^n_{\ell,2}$ denotes the $\ell^{\text{th}}$ member of the set of all 2-element subsets of $\{0,1,\ldots,n\}$.

### A.4 Decision Rule Policies

Polynomial decision rules of order 0, 1, and 2 are used for all choices of base model, degree-of-freedom partitioning, uncertain parameter specification, and uncertainty set parameterization. In the event that all degrees of freedom are taken to be first-stage variables (i.e., there are no second stage variables), the decision rule order is irrelevant, and we include only the instance labeled with order 0.

## B Addressing PyROS Subsolver Termination Statuses

In the event a Pyomo optimizer is invoked to solve a PyROS sampled or separation subproblem, the optimizer returns any of the Pyomo termination condition and solver status combinations listed in the left two columns of Table 7; the action taken by the PyROS meta-solver in each case is specified in the remaining columns.

**Table 7.** Mapping Pyomo subsolver termination conditions to PyROS sampled and separation problem actions.

| Pyomo Termination Condition | Pyomo Solver Status | Action Taken by PyROS After | |
|---|---|---|---|
| | | Sampled Problem | Separation Problem |
| `optimal` | `ok` | Accept solution | |
| `globallyOptimal` | `ok` | Accept solution | |
| `locallyOptimal` | `ok` | Accept solution | |
| `feasible` | `ok` | Try next backup solver* | |
| `maxTimeLimit` | `ok` | Try next backup solver* | |
| `maxIterations` | `ok` | Try next backup solver* | |
| `maxEvaluations` | `ok` | Try next backup solver* | |
| `minStepLength` | `ok` | Try next backup solver* | |
| `minFunctionValue` | `ok` | Try next backup solver* | |
| `other` | `ok` | Try next backup solver* | |
| `unbounded` | `warning` | Try next backup solver* | |
| `infeasible` | `warning` | Return `robust_infeasible` | Try next backup solver* |
| `infeasibleOrUnbounded` | `warning` | Try next backup solver* | |
| `invalidProblem` | `warning` | Try next backup solver* | |
| `intermediateNonInteger` | `warning` | Try next backup solver* | |
| `noSolution` | `warning` | Try next backup solver* | |
| `solverFailure` | `error` | Try next backup solver* | |
| `internalSolverError` | `error` | Try next backup solver* | |
| `error` | `error` | Try next backup solver* | |
| `unknown` | `unknown` | Try next backup solver* | |
| `userInterrupt` | `aborted` | Return `subsolver_error` | |
| `resourceInterrupt` | `aborted` | Return `subsolver_error` | |
| `licensingProblems` | `aborted` | Return `subsolver_error` | |

* If no backup solver available, return `subsolver_error`.

## References

[1] A. L. Soyster. “Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming”. In: *Operations Research* 21.5 (1973), pp. 1154–1157.

[2] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton, New Jersey: Princeton University Press, 2009.

[3] A. Ben-Tal et al. “Adjustable robust solutions of uncertain linear programs”. In: *Mathematical Programming* 99.2 (2004), pp. 351–376.

[4] Dimitris Bertsimas and Constantine Caramanis. “Finite Adaptability in Multistage Linear Optimization”. In: *IEEE Transactions on Automatic Control* 55.12 (2010), pp. 2751–2766.

[5] Anirudh Subramanyam, Chrysanthos E. Gounaris, and Wolfram Wiesemann. “K-adaptability in two-stage mixed-integer robust optimization”. In: *Mathematical Programming Computation* 12.2 (2020), pp. 193–224.

[6] Dimitris Bertsimas, David B. Brown, and Constantine Camaranis. “Theory and Applications of Robust Optimization”. In: *SIAM Review* 53.3 (2011), pp. 454–501.

[7] Sven Leyffer et al. “A survey of nonlinear robust optimization”. In: *INFOR: Information Systems and Operational Research* 58.2 (2020), pp. 342–373.

[8] İhsan Yanıkoğlu, Bram L. Gorissen, and Dick den Hertog. “A Survey of Adjustable Robust Optimization”. In: *European Journal of Operational Research* 277 (2019), pp. 799–813.

[9] Bram L. Gorissen, İhsan Yanıkoğlu, and Dick den Hertog. “A practical guide to robust optimization”. In: *Omega* 53 (2013), pp. 124–137.

[10] Aurélie Thiele, Tara Terry, and Marina A. Epelman. *Robust Linear Optimization With Recourse*. 2010. URL: https://optimization-online.org/?p=10712 (visited on Feb. 7, 2026).

[11] Bo Zeng and Long Zhao. “Solving two-stage robust optimization problems using a column-and-constraint generation method”. In: *Operations Research Letters* 41.5 (2013), pp. 457–461.

[12] Almir Mutapcic and Stephen Boyd. “Cutting-set methods for robust convex optimization with pessimizing oracles”. In: *Optimization Methods and Software* 24.3 (2009), pp. 381–406.

[13] Johannes Wiebe, Inês Cecílio, and Ruth Misener. “Robust Optimization for the Pooling Problem”. In: *Industrial & Engineering Chemistry Research* 58.28 (2019), pp. 12712–12722.

[14] Natalie M. Isenberg et al. “A generalized cutting-set approach for nonlinear robust optimization in process systems engineering”. In: *AIChE Journal* 67.5 (2021), e17175.

[15] Y. Zhang. “General Robust-Optimization Formulation for Nonlinear Programming”. In: *Journal of Optimization Theory and Applications* 132.1 (2007), pp. 111–124.

[16] Yuan Yuan, Zukui Li, and Biao Huang. “Nonlinear robust optimization for process design”. In: *AIChE Journal* 64.2 (2018), pp. 481–494.

[17] Martina Kuchlbauer, Frauke Liers, and Michael Stingl. “Adaptive Bundle Methods for Nonlinear Robust Optimization”. In: *INFORMS Journal on Computing* 34.4 (2022), pp. 2106–2124.

[18] Martina Kuchlbauer. “Outer approximation for generalized convex mixed-integer nonlinear robust optimization problems”. In: *Operations Research Letters* 60 (2025), p. 107243.

[19] R. Hettich and K. O. Kortanek. “Semi-Infinite Programming: Theory, Methods, and Applications”. In: *SIAM Review* 35.3 (2006), pp. 380–429.

[20] Marco López and Georg Still. “Semi-infinite programming”. In: *European Journal of Operational Research* 180.2 (2007), pp. 491–518.

[21] Hatim Djelassi, Alexander Mitsos, and Oliver Stein. "Recent advances in nonconvex semi-infinite programming: Applications and algorithms". In: *EURO Journal on Computational Optimization* 9 (2021), p. 100006.

[22] J. W. Blankenship and J. E. Falk. "Infinitely constrained optimization problems". In: *Journal of Optimization Theory and Applications* 19.2 (1976), pp. 261–281.

[23] Alexander Mitsos. "Global optimization of semi-infinite programs via restriction of the right-hand side". In: *Optimization* 60.10 (2011), pp. 1291–1308.

[24] Angelos Tsoukalas and B. Rustem. "A feasible point adaptation of the Blankenship and Falk algorithm for semi-infinite programming". In: *Optimization Letters* 5.4 (2011), pp. 705–716.

[25] Hatim Djelassi and Alexander Mitsos. "A hybrid discretization algorithm with guaranteed feasibility for the global solution of semi-infinite programs". In: *Journal of Global Optimization* 68.2 (2017), pp. 227–253.

[26] Tobias Seidel and Karl-Heinz Küfer. "An adaptive discretization method solving semi-infinite optimization problems with quadratic rate of convergence". In: *Optimization* 71.8 (2022), pp. 2211–2239.

[27] Hatim Djelassi and Alexander Mitsos. "Global Solution of Semi-infinite Programs with Existence Constraints". In: *Journal of Optimization Theory and Applications* 188.3 (2021), pp. 863–881.

[28] Phillip R. Jenkins, Brian J. Lunday, and Matthew J. Robbins. "Robust, multi-objective optimization for the military medical evacuation location-allocation problem". In: *Omega* 97 (2020), p. 102088.

[29] Dimitris Bertsimas and Yeesian Ng. "Robust and stochastic formulations for ambulance deployment and dispatch". In: *European Journal of Operational Research* 279.2 (2019), pp. 557–571.

[30] Logan R. Matthews, Chrysanthos E. Gounaris, and Ioannis G. Kevrekidis. "Designing networks with resiliency to edge failures using two-stage robust optimization". In: *European Journal of Operational Research* 279.3 (2019), pp. 704–720.

[31] Anirudh Subramanyam, Panagiotis P. Repoussis, and Chrysanthos E. Gounaris. "Robust Optimization of a Broad Class of Heterogeneous Vehicle Routing Problems Under Demand Uncertainty". In: *INFORMS Journal on Computing* 32.3 (2020), pp. 661–681.

[32] Nikos H. Lappas et al. "Adjustable Robust Optimization for multi-tasking scheduling with reprocessing due to imperfect tasks". In: *Optimization and Engineering* 20.4 (2019), pp. 1117–1159.

[33] Sanjula Kammammettu and Zukui Li. "Two-Stage Robust Optimization of Water Treatment Network Design and Operations under Uncertainty". In: *Industrial & Engineering Chemistry Research* 59.3 (2020), pp. 1218–1233.

[34] Nikolaos H. Lappas and Chrysanthos E. Gounaris. "Multi-stage adjustable robust optimization for process scheduling under uncertainty". In: *AIChE Journal* 62.5 (2016), pp. 1646–1667.

[35] Chrysanthos E. Gounaris and Daniel R. Schmidt. "Generalized Hose uncertainty in single-commodity robust network design". In: *Optimization Letters* 14.4 (2020), pp. 925–944.

[36] Anirudh Subramanyam et al. “Robust Multiperiod Vehicle Routing Under Customer Order Uncertainty”. In: *Operations Research* 69.1 (2021), pp. 30–60.

[37] Nikolaos H. Lappas and Chrysanthos E. Gounaris. “Theoretical and computational comparison of continuous-time process scheduling models for adjustable robust optimization”. In: *AIChE Journal* 64.8 (2018), pp. 3055–3070.

[38] Chrysanthos E. Gounaris, Wolfram Wiesemann, and Christodoulos A. Floudas. “The Robust Capacitated Vehicle Routing Problem Under Demand Uncertainty”. In: *Operations Research* 61.3 (2013), pp. 677–693.

[39] Miguel Sousa Lobo. “Robust and convex optimization with applications in finance”. PhD thesis. United States – California: Stanford University, 2000.

[40] D. Goldfarb and G. Iyengar. “Robust Portfolio Selection Problems”. In: *Mathematics of Operations Research* 28.1 (2003), pp. 1–38.

[41] Huan Xu, Constantine Caramanis, and Sujay Sanghavi. “Robust PCA via Outlier Pursuit”. In: *Advances in Neural Information Processing Systems*. Vol. 23. Curran Associates, Inc., 2010.

[42] Constantine Camaranis, Shie Mannor, and Huan Xu. “Robust Optimization in Machine Learning”. In: *Optimization for Machine Learning*. MIT Press, 2011.

[43] Seung-Jean Kim and Stephen Boyd. “A Minimax Theorem with Applications to Machine Learning, Signal Processing, and Finance”. In: *SIAM Journal on Optimization* 19.3 (2008), pp. 1344–1367.

[44] Line A. Roald et al. “Power systems optimization under uncertainty: A review of methods and applications”. In: *Electric Power Systems Research* 214 (2023), p. 108725.

[45] E. T. Hale and Y. Zhang. “Case Studies for a First-Order Robust Nonlinear Programming Formulation”. In: *Journal of Optimization Theory and Applications* 134.1 (2007), pp. 27–45.

[46] Ilse M. Cerrillo-Briones and Luis A. Ricardez-Sandoval. “Robust optimization of a post-combustion CO2 capture absorber column under process uncertainty”. In: *Chemical Engineering Research and Design* 144 (2019), pp. 386–396.

[47] Ignacio E. Grossmann et al. “Recent advances in mathematical programming techniques for the optimization of process systems under uncertainty”. In: *Computers & Chemical Engineering*. 12th International Symposium on Process Systems Engineering & 25th European Symposium of Computer Aided Process Engineering (PSE-2015/ESCAPE-25), 31 May - 4 June 2015, Copenhagen, Denmark 91 (2016), pp. 3–14.

[48] Jason A. F. Sherman et al. “Designing a Robust MEA-Based Post-Combustion Carbon Capture Process with Capture Rate Guarantees”. Paper submitted for publication. 2026.

[49] Ilayda Akkor et al. “Epistemic Uncertainty Analysis and Robust Optimization of a Second-Generation Solvent-Based Post-Combustion Carbon Capture Process”. Forthcoming. 2026.

[50] Manuel Tejeda-Iglesias et al. “Explicit model predictive controller under uncertainty: An adjustable robust optimization approach”. In: *Journal of Process Control* 84 (2019), pp. 115–132.

[51] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2 (2017), pp. 295–320.

[52] Daniel Jungen et al. *libDIPS — Discretization-Based Semi-Infinite and Bilevel Programming Solvers*. 2025. URL: https://optimization-online.org/?p=24914 (visited on Feb. 7, 2026).

[53] Hatim Djelassi. "Discretization-based algorithms for the global solution of hierarchical programs". PhD thesis. Aachen, Germany: RWTH Aachen University, 2020.

[54] Hatim Djelassi, Moll Glass, and Alexander Mitsos. "Discretization-based algorithms for generalized semi-infinite and bilevel programs with coupling equality constraints". In: *Journal of Global Optimization* 75.2 (2019), pp. 341–392.

[55] Omid Nejadseyfi. "Robustimizer: A graphical user interface application for efficient uncertainty quantification, robust optimization, and reliability-based optimization of processes and designs". In: *SoftwareX* 30 (2025), p. 102077.

[56] Gabriella Dellino, Jack P. C. Kleijnen, and Carlo Meloni. "Robust optimization in simulation: Taguchi and Response Surface Methodology". In: *International Journal of Production Economics* 125.1 (2010), pp. 52–59.

[57] Phebe Vayanos, Qing Jin, and George Elissaios. "ROC++: Robust Optimization in C++". In: *INFORMS Journal on Computing* 34.6 (2022), pp. 2873–2888.

[58] Joel Goh and Melvyn Sim. "Robust Optimization Made Easy with ROME". In: *Operations Research* 59.4 (2011), pp. 973–985.

[59] Zhi Chen, Melvyn Sim, and Peng Xiong. "Robust Stochastic Optimization Made Easy with RSOME". In: *Management Science* 66.8 (2020), pp. 3329–3339.

[60] Zhi Chen and Peng Xiong. "RSOME in Python: An Open-Source Package for Robust Stochastic Optimization Made Easy". In: *INFORMS Journal on Computing* 35.4 (2023), pp. 717–724.

[61] Johannes Wiebe and Ruth Misener. "ROmodel: A Python Robust Optimization Modeling Toolbox". In: *Computer Aided Chemical Engineering*. Vol. 50. Elsevier, 2021, pp. 683–688.

[62] Johannes Wiebe and Ruth Misener. "ROmodel: modeling robust optimization problems in Pyomo". In: *Optimization and Engineering* (2021). Publisher: Springer, pp. 1–22.

[63] William E. Hart, Jean-Paul Watson, and David L. Woodruff. "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.

[64] Michael L. Bynum et al. *Pyomo — Optimization Modeling in Python*. 3rd ed. Vol. 67. Springer Optimization and Its Applications. Cham: Springer International Publishing, 2021.

[65] Marc Goerigk. "ROPI—a robust optimization programming interface for C++". In: *Optimization Methods and Software* 29.6 (2014), pp. 1261–1280.

[66] A. Ismael F. Vaz, Edite M. G. P. Fernandes, and M. Paula S. F. Gomes. "SIPAMPL: Semi-infinite programming with AMPL". In: *ACM Trans. Math. Softw.* 30.1 (2004), pp. 47–61.

[67] A. Ismael F. Vaz, Edite M.G.P. Fernandes, and M. Paula S.F. Gomes. *NSIPS Version 2.1: Nonlinear Semi-Infinite Programming Solver*. Technical Report ALG/IV/5-2004. Braga, Portugal: Universidade do Minho, 2004.

[68] The Mathworks, Inc. *fseminf*. Version R2025a. Natick, Massachusetts, United States, 2025. URL: https://www.mathworks.com/help/optim/ug/fseminf.html (visited on Feb. 7, 2026).

[69] Johan Löfberg. "YALMIP : a toolbox for modeling and optimization in MATLAB". In: *2004 IEEE International Conference on Robotics and Automation*. 2004 IEEE International Conference on Robotics and Automation. Institute of Electrical and Electronics Engineers, 2004, pp. 284–289.

[70] Johan Löfberg. "Automatic robust convex programming". In: *Optimization Methods and Software* 27.1 (2012), pp. 115–129.

[71] Dimitris Bertsimas, Dan Andrei Iancu, and Pablo A. Parrilo. "A Hierarchy of Near-Optimal Policies for Multistage Adaptive Optimization". In: *IEEE Transactions on Automatic Control* 56.12 (2011), pp. 2809–2824.

[72] Erick Delage and Dan A. Iancu. "Robust Multistage Decision Making". In: *The Operations Research Revolution*. INFORMS TutORials in Operations Research. INFORMS, 2015, pp. 20–46.

[73] Robert Tibshirani. "Regression Shrinkage and Selection Via the Lasso". In: *Journal of the Royal Statistical Society*. B 58.1 (1996), pp. 267–288.

[74] Bethany Nicholson et al. "pyomo.dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations". In: *Mathematical Programming Computation* 10.2 (2018), pp. 187–223.

[75] Jean-Paul Watson, David L. Woodruff, and William E. Hart. "PySP: modeling and solving stochastic programs in Python". In: *Mathematical Programming Computation* 4.2 (2012), pp. 109–149.

[76] Katherine A. Klise et al. "Parmest: Parameter Estimation Via Pyomo". In: *Proceedings of the 9th International Conference on Foundations of Computer-Aided Process Design*. 9th International Conference on Foundations of Computer-Aided Process Design. Ed. by Salvador García Muñoz, Carl D. Laird, and Matthew J. Realff. Vol. 47. Computer Aided Chemical Engineering. Elsevier, 2019, pp. 41–46.

[77] Bin Wang, Wenzhong Shi, and Zelang Miao. "Confidence Analysis of Standard Deviational Ellipse and Its Extension into Higher Dimensional Euclidean Space". In: *PLoS ONE* 10.3 (2015), e0118537.

[78] Alexander Mitsos and Hatim Djelassi. *A Test Set of Semi-Infinite Programs*. Aachen, Germany, 2016. URL: http://permalink.avt.rwth-aachen.de/?id=472053 (visited on Apr. 7, 2025).

[79] Pietro Belotti. *COUENNE: a user's manual*. 2020. URL: https://www.coin-or.org/svn/Couenne/releases/0.5.8/Couenne/doc/couenne-user-manual.pdf (visited on Feb. 7, 2026).

[80] GAMS Corporation. *The GAMS Model Library*. Version 35. 2021. URL: https://www.gams.com/35/gamslib_ml/libhtml/ (visited on Feb. 7, 2026).

[81] GAMS Corporation. *GAMS World model library*. 2023. URL: https://github.com/GAMS-dev/gamsworld (visited on Feb. 7, 2026).

[82] Yi Zhang and Nikolaos V. Sahinidis. "Solving continuous and discrete nonlinear programs with BARON". In: *Computational Optimization and Applications* 92 (2024), pp. 1123–1161.

[83] IBM. *IBM ILOG CPLEX Optimization Studio*. Version 22.1.0. 2022. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio (visited on Feb. 7, 2026).

[84] Andreas Wächter and Lorenz T. Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical Programming* 106.1 (2006), pp. 25–57.

[85] Iain S. Duff. "MA57—a code for the solution of sparse symmetric definite and indefinite systems". In: *ACM Transactions on Mathematical Software (TOMS)* 30.2 (2004), pp. 118–144.