ROmodel: modeling robust optimization problems in Pyomo

Johannes Wiebe¹ · Ruth Misener¹

Received: 4 May 2021 / Revised: 8 November 2021 / Accepted: 23 November 2021 /
Published online: 21 December 2021
© The Author(s) 2021

Abstract

This paper introduces ROmodel, an open source Python package extending the modeling capabilities of the algebraic modeling language Pyomo to robust optimization problems. ROmodel helps practitioners transition from deterministic to robust optimization through modeling objects which allow formulating robust models in close analogy to their mathematical formulation. ROmodel contains a library of commonly used uncertainty sets which can be generated using their matrix representations, but it also allows users to define custom uncertainty sets using Pyomo constraints. ROmodel supports adjustable variables via linear decision rules. The resulting models can be solved using ROmodels solvers which implement both the robust reformulation and cutting plane approach. ROmodel is a platform to implement and compare custom uncertainty sets and reformulations. We demonstrate ROmodel's capabilities by applying it to six case studies. We implement custom uncertainty sets based on (warped) Gaussian processes to show how ROmodel can integrate data-driven models with optimization.

Keywords Robust optimization · Mathematical programming · Software tools

1 Introduction

Robust optimization is a common way of managing optimization under uncertainty in process systems engineering: applications range from production scheduling to flexible chemical process design (Janak and Floudas 2005; Li and Ierapetritou 2008; Zhang et al. 2015; Ning and You 2017; Shang and You 2018; Grossmann et al. 2016; Pistikopoulos et al. 2021). New developments in robust optimization include

✉ Johannes Wiebe
j.wiebe17@imperial.ac.uk

Ruth Misener
r.misener@imperial.ac.uk

¹ Imperial College London, London, UK

distributionally and adjustable robust optimization to reduce solution conservatism (Grossmann et al. 2016), data-driven robust optimization to model uncertainty sets based on available data (Bertsimas et al. 2018), and approximate robust optimization to make non-linear problems more tractable (Houska and Diehl 2013). This large number of alternative approaches and the required domain knowledge can discourage practitioners from making the transition from deterministic optimization to optimization under uncertainty. Furthermore, the lack of a platform that allows the easy implementation and application of new algorithms means that it is difficult for researchers to compare different approaches (Marc and Schöbel 2016).

This paper introduces RModel, a Python package extending the popular, Python-based algebraic modeling language Pyomo (Hart et al. 2011, 2017) to facilitate modeling of robust optimization problems and implementation of robust optimization algorithms. RModel aims to (i) make robust optimization more accessible to practitioners and facilitate moving from deterministic to robust optimization, (ii) demonstrate how robust optimization problems can be modeled in close analogy to their mathematical formulation, and (iii) provide an open-source platform for researchers to implement and compare new robust reformulations and uncertainty sets. To this end, RModel introduces intuitive modeling which closely resembles the optimization problem's underlying mathematical formulation and will feel familiar to Pyomo users. RModel uses the richness of Pyomo's solver interfaces and methods for model transformations and Python's data processing capabilities. It supports both automatic reformulation and cutting plane algorithms. Uncertainty sets can be chosen from a library of common geometries, or custom defined using Pyomo constraints. RModel also support adjustable robust optimization through linear decision rules. Because RModel and Pyomo are open-source, RModel can be extended to incorporate additional uncertainty set geometries and reformulations. As an example, we implemented uncertainty sets based on (warped) Gaussian processes for black-box constrained problems.

Robust optimization has been applied to many types of engineering problems. It has, for example, extensively been applied to scheduling and planning problems in process systems engineering (Janak and Floudas 2005; Li and Ierapetritou 2008; Ning and You 2017; Shang and You 2018; Wiebe et al. 2018), but also to robust design problems (Yuan et al. 2018; Gong and You 2018) and robust control (Diehl et al. 2008). Types of uncertainties considered have ranged from uncertain demands (Li et al. 2012) and uncertain prices (Wiebe et al. 2020) to uncertain rates of equipment degradation (Wiebe et al. 2018, 2020) and uncertain product/reagent purities and consumption rates (Zhang et al. 2018; Wiebe et al. 2019). A large number of these applications use ellipsoidal and/or polyhedral uncertainty sets which are available as library sets in RModel (Li et al. 2012; Shang et al. 2017; Zhang et al. 2018; Wiebe et al. 2018, 2019). Other geometries could be accommodated using RModel's capabilities for modeling generic convex sets using Pyomo constraints or by extending RModel to new, custom uncertainty sets. In many engineering applications, the optimal choice of uncertainty set geometry and solution method (reformulation or cutting plane) is not obvious before solving the problem. With RModel, engineers can implement their robust model once and then solve it using different uncertainty set geometries and solvers. While most engineering applications have been LP and

MILP problems, robust optimization has also been applied to non-linear engineering problems (Diehl et al. 2008; Li et al. 2012; Gong and You 2018; Wiebe et al. 2019). ROmodel can solve both MILP and MINLP problems depending on the Pyomo solver interface used.

There are other software packages for *solving* robust optimization problems, e.g. ROME (Goh and Sim 2011), RSOME (Chen et al. 2020), ROC++ (Vayanos et al. 2020), and PyROS Isenberg et al. (2020). All four approaches are designed as robust optimization *solvers* and, except for PyROS which is based on Pyomo, do not rely on a general purpose algebraic modeling language. In contrast, ROmodel focuses on *modeling* robust optimization problems. By building on Pyomo, ROmodel simplifies the transition from deterministic to robust optimization, since both the deterministic and robust model can be implemented in the same environment. ROmodel also allows access to a much larger number of deterministic solvers than these other solvers. Unlike PyROS, which uses Pyomo's Var component to model uncertain parameters and adjustable variables, ROmodel introduces new, dedicated Pyomo modeling components for robust problems allowing for more intuitive modeling. ROmodel also supports both the cutting plane and reformulation methods as well as modeling custom uncertainty sets with Pyomo constraints. In contrast to AIMMS, which has some capabilities for modeling robust optimization problems (AIMMS 2021), ROmodel is open source and allows possibilities for extension. Ju MPeR Dunning (2016), which extends Julia's modeling language Ju MP to robust optimization problems, is the most similar to ROmodel. In contrast to JuMPeR, which uses polyhedral sets by default, ROmodel automatically recognizes uncertainty set geometries and applies appropriate reformulations, if available, without the user having to specify which geometry they are using. ROmodel also supports both uncertain constraints and objectives while JuMPeR only supports uncertain constraints.

A further advantage of ROmodel is that it is based on Python, which is popular in data analytics and machine learning. ROmodel therefore allows data-based techniques to be integrated seamlessly with robust optimization methods. As an example, we implement (warped) Gaussian process-based uncertainty sets in ROmodel Wiebe et al. (2020). These sets seamlessly integrate Gaussian processes trained in the Python library GPy (2012) into robust optimization problems. ROmodel is open source and available on Wiebe et al. (2020).

The rest of this paper is structured as follows. Section 2 introduces ROmodel's new modeling objects and shows how they can be used to model robust optimization problems. Section 3 introduces ROmodel's three solvers: a reformulation based solver, a cutting plane solver, and a nominal solver for obtaining nominal solutions of robust problems. Section 4 discusses how ROmodel can be extended and demonstrates our implementation of Gaussian process-based uncertainty sets for black-box constrained problems. Section 5 introduces six case studies and presents results.

2 Modeling

Consider the following generic deterministic optimization problem

$$\min_{x \in \mathcal{X}, y} f(x, y, \bar{\xi}) \quad (1a)$$

$$\text{s.t.} \quad g(x, y, \bar{\xi}) \leq 0 \quad (1b)$$

where $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ are decision variables and $\bar{\xi} \in \mathbb{R}^k$ is a vector of (nominal) parameters. If the parameter vector $\bar{\xi}$ is not known exactly, we can construct the following robust version of Problem 1:

$$\min_{x \in \mathcal{X}, y(\xi)} \max_{\xi \in \mathcal{U}(x)} f(x, y(\xi), \xi) \quad (2a)$$

$$\text{s.t.} \quad g(x, y(\xi), \xi) \leq 0 \forall \xi \in \mathcal{U}(x) \quad (2b)$$

Here $x \in \mathbb{R}^n$ are “here and now” variables, determined before the uncertainty is revealed, while $y(\xi)$ are adjustable variables, determined after the uncertainty is revealed. The uncertain parameter vector ξ is bounded by the uncertainty set $\mathcal{U}(x)$, which may depend on x and which contains the nominal values $\bar{\xi}$ from Problem 1. Optimization Problem 2, a generic robust optimization problem with uncertainty in the objective and constraints, is the basis for RModel. Note that we are limiting ourselves here to one uncertain constraint for simplicity of notation only, RModel can handle multiple robust constraints. Also note that there can be an arbitrary number of deterministic constraints which define the set $\mathcal{X}$.

RModel introduces three new modeling objects to represent robust optimization problems like Problem 2 within Pyomo:

- 1 **UncParam**: A class similar to Pyomo’s Param and Var class used to model uncertain parameters ξ . One can supply a nominal argument, which defines the vector of nominal values $\bar{\xi}$ used to replace the uncertain parameters when solving the deterministic (nominal) problem (Eq. 1).
- 2 **UncSet**: Each UncParam object has an UncSet object associated with it. The UncSet class, based on Pyomo’s Block class, models the uncertainty sets $\mathcal{U}$. Uncertainty sets $\mathcal{U}$ can be defined by (i) constructing generic sets by adding Pyomo constraints to the UncSet object, or (ii) through a library of common uncertainty set geometries, using their matrix representation as an input.
- 3 **AdjustableVar**: A class similar to Pyomo’s Var class used to model adjustable variables which can be determined after some of the uncertainty has been resolved, i.e. $y(\xi)$.

These three new modeling objects are sufficient for modeling quite generic robust optimization problems. They are set up to allow modeling problems in an intuitive way which is closely related to their mathematical formulation (Problem 2). We discuss each modeling object and how it is used to construct robust optimization problems in the subsequent sections.

2.1 Uncertain Parameters

Indexed uncertain parameters are constructed in analogy to Pyomo's Var type for variables:

```
1 m.c = UncParam(range(3), nominal=[0.1, 0.2, 0.3], uncset=m.U)
```

The nominal argument specifies a list of nominal values $\bar{\xi}$ for the uncertain parameters ξ . The nominal values are currently only used in conjunction with the nominal solver outlined in Sect. 3.3. ROmodel does not check if the nominal values are contained in the uncertainty set. The uncset argument specifies the uncertainty set to use for these parameters. The two approaches for constructing the uncertainty set $m.U$ are discussed in the next two chapters.

2.2 Generic uncertainty sets

Generic uncertainty sets are constructed with the UncSet class. This class inherits from Pyomo's Block class. Users can construct generic uncertainty sets by adding Pyomo constraints to an UncSet object in the same way as they would add constraints to a Block object in Pyomo. The following example shows how a polyhedral set can be modeled using the UncSet class:

```
1 from romodel import UncSet, UncParam
2 # Define uncertainty set
3 m.U = UncSet()
4 # Define uncertain parameter
5 m.w = UncParam(range(2), uncset=m.U, nominal=[0.5, 0.5])
6 # Add constraints to uncertainty set
7 m.U.cons1 = Constraint(expr=m.w[0] + m.w[1] <= 1)
8 m.U.cons2 = Constraint(expr=m.w[0] - m.w[1] <= 1)
9 ...
```

The uncset and nominal arguments define the uncertainty set and a vector of nominal values for the uncertain parameter $m.w$. ROmodel's strategy of modeling uncertainty sets using Pyomo's Constraint modeling object is analogous to typical robust optimization formulations. Users can also define multiple uncertainty sets and replace one by another:

```

1      # Define second uncertainty set
2      m.U2 = ro.UncSet()
3      # Swap uncertainty sets and solve
4      m.coef.uncset = m.U2
5      solver = pe.SolverFactory('romodel.cuts')
6      solver.solve(m)

```

2.3 Library uncertainty sets

For commonly used, standard uncertainty sets, the generic approach (Sect. 2.2) is unnecessarily complicated. Therefore, ROmodel implements custom classes which can define an uncertainty set using its matrix representation. ROmodel implements polyhedral and ellipsoidal sets. The user can define polyhedral sets of the form $\mathcal{U} = \{w \mid Pw \leq b\}$ by passing the matrix P and the right hand side b to the PolyhedralSet class:

```

1      from romodel.uncset import PolyhedralSet
2      # Define polyhedral set
3      m.U = PolyhedralSet(mat=[[ 1,  1],
4                                [ 1, -1],
5                                [-1,  1],
6                                [-1, -1]],
7                                rhs=[1, 1, 1, 1])

```

ROmodel creates ellipsoidal sets of the form $(w - \mu)\Sigma^{-1}(w - \mu) \leq 1$ using the EllipsoidalSet class, the covariance matrix Σ and the mean vector μ :

```

1      from romodel.uncset import EllipsoidalSet
2      # Define ellipsoidal set
3      m.U = EllipsoidalSet(cov=[[1, 0, 0],
4                                [0, 1, 0],
5                                [0, 0, 1]],
6                                mean=[0.5, 0.3, 0.1])

```

Note that ROmodel does not currently have capabilities for taking intersections of (library) uncertainty sets. Library sets specifying intersections of common uncertainty set geometries could be added to ROmodel. In Sect. 4 we discuss how additional library sets can be added to ROmodel using data-driven uncertainty sets based on (warped) Gaussian processes as an example.

2.4 Constructing uncertain constraints

After defining uncertain parameters and an uncertainty set, the user can construct uncertain constraints implicitly by using the uncertain parameters in a Pyomo constraint. Consider the following deterministic Pyomo constraint:

```

1      # deterministic
2      m.x = Var(range(3))
3      c = [0.1, 0.2, 0.3]
4      m.cons = Constraint(expr=sum(c[i]*m.x[i] for i in m.x) <= 0)

```

If the coefficients c are uncertain, we can model the robust constraint $c^T x \leq 0, \forall c \in \mathcal{U}$ as:

```

1      # robust
2      m.x = Var(range(3))
3      m.c = UncParam(range(3), nominal=[0.1, 0.2, 0.3], uncset=m.U)
4      m.cons = Constraint(expr=sum(m.c[i]*m.x[i] for i in m.x) <= 0)

```

The uncertain parameter $m.c$ can be used in Pyomo constraints in the same way as Pyomo Var or Param objects. ROmodel's solvers automatically recognize constraints and objectives containing uncertain parameters.

2.5 Adjustable variables

ROmodel also has capabilities for modeling adjustable variables. Adjustable variables $y(\xi)$ are variables which can be determined after some (or all) of the uncertainty has been revealed. Defining an adjustable variable is analogous to defining a regular variable in Pyomo, with an additional `uncparam` argument specifying a list of uncertain parameters which the adjustable variable depends on:

```

1  # Define uncertain parameters
2  m.w = UncParam(range(3), nominal=[1, 2, 3])
3  # Define adjustable variable which depends on uncertain
   parameter
4  m.y = AdjustableVar(range(3), uncparams=[m.w], bounds=(0, 1))

```

The uncertain parameters can also be set individually for each element of the adjustable variables index using the `set_uncparams` function:

```

1  # Set uncertain parameters for individual indicies
2  m.y[0].set_uncparams([m.w[0]])
3  m.y[1].set_uncparams([m.w[0], m.w[1]])

```

This means that multi-stage problems, in which each adjustable variable depends on the uncertain parameters from previous stages, can be modeled in ROmodel. ROmodel only implements linear decision rules for solving adjustable robust optimization problems. If a model contains adjustable variables in a constraint or objective, ROmodel automatically replaces it by a linear decision rule based on the specified uncertain parameters.

3 Solvers

ROmodel's `UncSet` component can be used to model any uncertainty set geometry which can be expressed using Pyomo constraints. However, not every geometry that can be modeled can necessarily also be solved. ROmodel has three solvers: A robust reformulation based solver, a cutting plane based solver, and a nominal solver. This section discusses how each of these solvers works and which types of problems and uncertainty set geometries they can solve.

3.1 Reformulation

The reformulation-based solver, illustrated in Fig. 1, implements standard duality based techniques for reformulating robust optimization problems into deterministic counterparts (Bertsimas and Sim 2004).

First, it detects every constraint containing uncertain parameters. Second, it checks the structure of each uncertain constraint and the corresponding uncertainty set to determine if a known reformulation is applicable. Finally, it applies a model transformation, generating the deterministic counterpart of each robust constraint. The deterministic counterpart is then solved using an appropriate

Fig. 1 Schematic of reformulation solver. For each uncertain constraint, the reformulation solver goes through all known transformations and identifies the geometry of the constraint/uncertainty set. It then applies the corresponding model transformation. If all uncertain constraints can be reformulated, the resulting deterministic counterpart is solved using one of Pyomo's solver interfaces. If one or more constraints cannot be reformulated, the problem cannot be solved.

solver available in Pyomo. The structure of the optimization problem and the uncertainty set geometry determine which solvers are applicable. If no applicable transformation can be identified for one or more constraints, the problem cannot be solved and ROmodel will raise an error.

ROmodel implements standard reformulations for ellipsoidal and polyhedral uncertainty sets and linear uncertain constraints (Ben-Tal and Nemirovski 1999; Bertsimas and Sim 2004). It also implements reformulations for black-box constrained problems (Wiebe et al. 2020). These are discussed in more detail in Sect. 4, which also discusses how ROmodel can be extended to include further reformulations.

3.2 Cutting planes

The cutting plane solver, outlined in Fig. 2, implements an iterative strategy for solving robust optimization problems (Mutapcic and Boyd 2009). It replaces each uncertain constraint and objective by a CutGenerator object which initially just contains the nominal constraint. The solver then iteratively solves the master problem and generates cuts to cut off solutions which are not robustly feasible.

Fig. 2 Schematic of cutting plane solver. The cutting plane solver iteratively solves the master problem and adds cutting planes until a robustly feasible solution is found.

A solution x^* is considered to be robustly feasible when for each uncertain constraint $g(x^*, y(\xi), \xi) \leq 0$, the objective value of the separation problem is smaller than some tolerance ϵ :

$$\max_{\xi \in \mathcal{U}} g(x^*, y(\xi), \xi) \leq \epsilon \quad (3)$$

ROmodel's cutting plane solver can generally be applied to any convex uncertainty set. The Pyomo solvers for solving the master and separation problems can be set individually using:

```

1 solver = SolverFactory('romodel.cuts')
2 # Master solver
3 solver.options['solver'] = 'gurobi'
4 # Sub solver
5 solver.options['subsolver'] = 'ipopt'

```

The solvers need to be appropriate for the corresponding problem, i.e., the above choice would be sensible if the master problem is a mixed-integer linear problem and the uncertain constraints and uncertainty set are continuous and convex. If the solvers are not appropriate, Pyomo will raise an error.

3.3 Nominal

ROmodel also includes a nominal solver. This solver replaces all occurrences of the uncertain parameters by their nominal values and solves the resulting deterministic problem:

```

1      # Obtaining the nominal solution
2      solver = SolverFactory('romodel.nominal')
3      solver.solve(m)
4

```

The nominal solver allows users to combine their implementations of the nominal and robust problem. An implementation of the robust model can be used to obtain the solution of the nominal problem.

4 Extending ROmodel for black-box constrained problems

ROmodel can be extended to incorporate additional reformulations and uncertainty set geometries. This section outlines how ROmodel can be extended using (warped) Gaussian process-based uncertainty sets for black-box constrained problems as an example. This example showcases the ease with which ROmodel can integrate Python's machine learning and data analytics capabilities with Pyomo's mathematical optimization modeling.

(Wiebe et al. 2020) propose a robust optimization-based approximation of a class of chance-constraints containing uncertain black-box functions $g(\cdot)$:

$$P\left(\sum_i g(y_i)x_i \leq b\right) \geq 1 - \alpha \quad (4)$$

The approach models the black-box function and associated uncertainty using a (warped) Gaussian process as a stochastic model. The standard Gaussian process is well known and commonly used as a surrogate model (Bhosekar and Ierapetritou 2018). Warped Gaussian processes are a more flexible variant of standard Gaussian process in which observations are mapped into a latent space using a non-linear, often neural net-style warping function (Snelson et al. 2003). If a standard Gaussian process models $g(\cdot)$, the chance constraint Eq. 4 can be reformulated exactly. For the warped Gaussian process, (Wiebe et al. 2020) propose an approximation based on Wolfe duality. Choosing between the standard and warped approach for a specific application is not trivial. We therefore implemented both approaches in ROmodel, so that practitioners can easily apply both to their application and compare.

In order to make these approaches available in ROmodel, we need to (i) implement two library uncertainty sets, GPSet and WarpedGPSet, which collect the relevant data, and (ii) implement two corresponding model transformations which perform the reformulations for standard and warped Gaussian process-based sets. The implementation is based on the Python module (Wiebe 2020), which includes Gaussian process models trained in the Python library (GPY 2012) in Pyomo models.

4.1 Implementing new library sets

Implementing a new library set mainly requires a new Python class collecting the necessary data. For the standard and warped Gaussian process set, this data consist of three arguments:

```

1      from romodel.uncset import GPSet, WarpedGPSet
2
3      # Define variables
4      m.y = Var([0, 1])
5
6      # Define uncertainty set based on standard GP
7      m.uncset_standard = GPSet(gp_standard, m.y, 0.95)
8
9      # Define uncertainty set based on warped GP
10     m.uncset_warped = WarpedGPSet(gp_warped, m.y, 0.95)

```

The first argument `gp_standard/warped` is a (warped) Gaussian process object trained in GPy. The second is an indexed Pyomo variable on which the GP depends, i.e. `y` in Eq. 4. The third parameter specifies the confidence level $1 - \alpha$ with which the true parameter is contained in the uncertainty set. I.e., in this case the confidence that the true parameter vector is an element of the uncertainty set is at least 95%.

The new sets can be used in the same way as other library sets, e.g.:

```

1      # Define variables
2      m.x = Var([0, 1])
3
4      # Define uncertain parameters
5      m.w = UncParam([0, 1], uncset=m.uncset_warped)
6
7      # Define constraint
8      m.cons = Constraint(expr=m.x[0]*m.w[0] + m.x[1]*m.w[1] <= 1)

```

Note that the uncertain parameter `m.w` corresponds to $g(y_i)$ in Formulation 4. Constraints which use this type of uncertainty set need to be linear in the uncertain parameter. Note that the indices of `m.y` and `m.w` need to be identical in the formulation above. If the black-box function depends on more than one variable, the Gaussian process-based sets can alternatively be specified using a dictionary:

```

1      # Define variables
2      m.y = Var([0, 1], ['a', 'b'])
3      # Define uncertainty set
4      y_dict = {0: [m.y[0, 'a'], m.y[0, 'b']],
5                  1: [m.y[1, 'a'], m.y[1, 'b']]},
6      m.uncset_warped = WarpedGPSet(gp, y_dict, 0.95}

```

The dictionary indicates that the uncertain parameter $m.w[0]$ depends on the variables $m.y[0, 'a']$ and $m.y[0, 'b']$ through the black-box function $g(\cdot)$, modeled in GPy by the Gaussian process gp.

Note that ROmodel's cutting plane solver is not applicable to the Gaussian process-based sets because the sets are decision dependent. Attempting to solve a problem with one of these sets therefore results in an error. When implementing new library sets which *can* be solved using cutting planes, an additional Python function `generate_cons_from_lib`, which generates Pyomo constraints for the uncertainty set based on the data collected by the library set, is required. As an example, the following code shows the implementation of the polyhedral set in ROmodel:

```

1  class PolyhedralSet(UncSet):
2      '''
3      Defines a polyhedral uncertainty set of shape:
4          mat * param <= rhs
5      '''
6      def __init__(self, mat, rhs, *args, **kwargs):
7          self.mat = mat
8          self.rhs = rhs
9          super().__init__(*args, **kwargs)
10         self._lib = True
11
12     def generate_cons_from_lib(self, param):
13         for i, row in enumerate(self.mat):
14             yield (None,
15                   quicksum(row[j]*param[ind]
16                           for j, ind in enumerate(param)),
17                   self.rhs[i])

```

4.2 Implementing new reformulations

For ROmodel to be able to solve models containing the two new Gaussian process-based sets, we need to implement the corresponding reformulations. Adding new reformulations to ROmodel generally requires two Python functions:

- 1 A function `_check_applicability` which detects whether a constraint and uncertainty set have the required structure, and
- 2 A function `_reformulate` which generates the robust counterpart.

The function `_check_applicability` iterates through the constraints of an uncertainty set to identify its geometry. E.g., for polyhedral sets, it checks that all constraints are linear. This function is only required if the reformulation is supposed to work with generically constructed uncertainty sets as described in Sect. 2.2. For library sets like the Gaussian process-based sets, only the `_reformulate` function is required. This function takes data describing the constraints and uncertainty set as an input and returns a Pyomo block containing the deterministic counterpart. E.g., the transformation for warped Gaussian process-based sets returns a block containing the Wolfe dual formulation described in Wiebe et al. (2020) while the transformation for polyhedral sets returns a block containing the linear dual. For a full example see the implemented reformulations in `romodel/reformulate/` on the (Wiebe et al. 2020).

5 Results

We use ROmodel to model and solve six case studies:

- (1) A portfolio optimization problem with uncertain returns (Bertsimas and Sim 2004),
- (2) A knapsack problem with uncertain item weights,
- (3) A pooling problem instance (Adhya et al. 1999) with uncertain product demands,
- (4) A capacitated facility location problem as an example for adjustable robust optimization, where the decision which facilities to build has to be made under demand uncertainty, while the decision from which facility to supply individual customers can be made once the uncertainty is resolved,
- (5) A production planning in which the price at which products can be sold depends on the amount produced through an uncertain black-box function modelled by a (warped) Gaussian process (Wiebe et al. 2020),
- (6) And a drill scheduling problem in which the equipment used to drill a well degrades at a rate which depends other drill parameters through a black-box function (Wiebe et al. 2020).

For the reformulation approach, we use Gurobi 9.0.3 to solve the reformulation for case studies 1–4 and Ipopt 3.12.12 for case studies 5 and 6. For the cutting plane approach, we use Gurobi 9.0.3 as the main and subsolver for all problems. All experiments were run on an i7-6700 CPU with 8x3.40GHz and 16GB RAM. All examples except for the drill scheduling example are included with ROmodel and can be used as follows:

```

1      import romodel.example as ex
2
3      portfolio = ex.Portfolio()
4
5      knapsack = ex.Knapsack()
6
7      pooling = ex.Pooling()
8
9      facility = ex.Facility()
10
11     planning = ex.ProductionPlanning(alpha=0.95, warped=True)

```

The implementation of the drill scheduling example is separately available on (Wiebe 2020). Table 1 shows a summary of the type of problem and number of variables, constraints, and uncertain parameters for each case study.

Below we discuss two case studies, the Knapsack and Facility location problems, in more detail. Detailed code for the Portfolio, Pooling, and Production planning case studies can be found on the ROmodel Github at [romodel/examples](https://github.com/romodel/romodel).

5.1 Knapsack problem

Our first case study is a classic Knapsack problem:

$$\max_x \sum_i v_i x_i \quad (5a)$$

$$\text{s.t. } w_i x_i \leq b \quad \forall w \in \mathcal{U} \quad (5b)$$

$$x_i \in \{0, 1\} \quad (5c)$$

The goal is to select from a list of items i with value v_i and weight w_i such that the total value selected is maximized while the total weight is below some limit b . The binary variable x_i is 1 if the item is selected and 0 otherwise. We consider the vector of weights w to be uncertain but bounded by the uncertainty set $\mathcal{U}$. The following code shows how the problem can be modeled and solved in ROmodel using an ellipsoidal uncertainty set:

Table 1 Type of problem, number of variables, constraints, and uncertain parameters for each case study.

Problem	Type	# Variables	# Constraints	# Unc. params
Portfolio	LP	6	2	5
Knapsack	MILP	4	1	4
Pooling	NLP	13	45	4
Facility	adj. MILP	4	9	5
Planning	NLP	4	1	3
Drilling	NLP	63	39	12

```

1 import pyomo.environ as pe
2 import romodel as ro
3 SolverFactory
4 # Set up parameters
5 items = ['i1', 'i2', 'i3', 'i4']
6 v = {'i1': 8, 'i2': 3, 'i3': 6, 'i4': 11}
7 w = {'i1': 5, 'i2': 7, 'i3': 4, 'i4': 3}
8 b = 14
9
10 # Set up model
11 m = pe.ConcreteModel()
12 # Set up variables
13 m.x = pe.Var(items, within=Binary)
14
15 # Data for ellipsoidal uncertainty set
16 mu = [w[i] for i in items]
17 Sig = [[10, -1, 0, 0],
18         [-1, 10, 0, 0],
19         [0, 0, 10, 0],
20         [0.0, 0.0, 0.0, 10]]
21 # Ellipsoidal set from library
22 m.U = ro.uncset.EllipsoidalSet(mu, Sig)
23
24 # Add uncertain parameter
25 m.w = ro.UncParam(items, nominal=w, uncset=m.U)
26 # Add objective & constraint
27 m.value = pe.Objective(expr=sum(v[i]*m.x[i] for i in items),
28                             sense=maximize)
29 m.weight = pe.Constraint(expr=sum(m.w[i]*m.x[i] for i in items)
30                             <= b)
31
32 # Solve model
33 solver = pe.SolverFactory('romodel.reformulation')
34 solver.solve(m)

```


Note that unlike other robust optimization solvers, ROmodel's modeling components behave exactly like Pyomo's modeling components. Practitioners who are used to implementing their deterministic models in Pyomo will find the syntax of ROmodel very familiar. Using ROmodel, they can apply robust optimization to their existing Pyomo models by making minimal changes to the model.

The results can be accessed as in any other Pyomo model. E.g. the variable values can be printed using `m.x.pprint()`. When using the reformulation solver, the reformulation for each uncertain constraint can be accessed using the postfix `_counterpart`. E.g. for the problem above `m.weight_counterpart.pprint()` gives the output:

```

1 1 Var Declarations
2   padding : Size=1, Index=None
3       Key : Lower : Value : Upper : Fixed : Stale : Domain
4       None : 0      : 5.46 : inf  : False : False : Reals
5 2 Constraint Declarations
6   det : Size=1, Index=None, Active=True
7       Key : Lower : Body                                     : Upper : Active
8       None : -Inf  : 10.0*x[i1]*x[i1] + 0.0 : True
9           - 1.0*x[i1]*x[i2]
10          - 1.0*x[i2]*x[i1]
11          + 10.0*x[i2]*x[i2]
12          + 10.0*x[i3]*x[i3]
13          + 10.0*x[i4]*x[i4]
14          - weight_counterpart.upper.padding**2
15   rob : Size=1, Index=None, Active=True
16       Key : Lower : Body                                     : Upper : Active
17       None : -Inf  : 5*x[i1] + 14.0 : True
18          + 7*x[i2]
19          + 4*x[i3]
20          + 3*x[i4]
21          + weight_counterpart.upper.padding

```

In this case, the counterpart consists of one additional variable, padding, and two constraints det and rob.

For the cutting plane solvers, the cuts for each uncertain constraint can be accessed using the postfix `_generator`, e.g. `m.weight_generator.pprint()` outputs a list of two cuts:

```

1 weight_generator : Size=1, Index=None, Active=True
2     1 Constraint Declarations
3         _constraints : Size=2, Index=weight_generator.
4         _constraints_index, Active=True
5             Key : Lower : Body : Upper : Active
6                 1 : -Inf : 5*x[i1] : 14.0 : True
7                     + 7*x[i2]
8                     + 4*x[i3]
9                     + 3*x[i4]
10                2 : -Inf : 6.84*x[i1] : 14.0 : True
11                    + 6.81*x[i2]
12                    + 5.82*x[i3]
                    + 4.82*x[i4]

```

5.2 Facility location problem

The facility location case study is an adjustable robust optimization problem with the following mathematical formulation:

$$\max_x \min_{d \in \mathcal{U}} \sum_i^N \sum_j^M c_{ij}^t y_{ij}(d_j) + \sum_i^N c_i^f x_i \quad (6a)$$

$$\text{s.t.} \quad \sum_i^N y_{ij}(d_j) = d_j \quad \forall j \in [M], d \in \mathcal{U} \quad (6b)$$

$$\sum_j^M y_{ij}(d_j) \leq m_i x_i \quad \forall i \in [N], d \in \mathcal{U} \quad (6c)$$

$$x_i \in \{0, 1\} \quad (6d)$$

Variables x_i are here and now binary variables which are 1 if facility i is built and 0 otherwise. Variables $y_{ij}(d_j)$ are adjustable variables which depend on the uncertain customer demand d_j for each customer j and determine how much of the customer's demand is satisfied from facility i . The maximum capacity for each facility is given by m_i . The uncertain parameters d_i are assumed to be bounded by the uncertainty set $\mathcal{U}$. The code below shows how RModel can be used to implement this problem using the generic uncertainty set approach to define a box uncertainty set:

```

1 import pyomo.environ as pe
2 import romodel as ro
3
4 # Set up parameters
5 N = 4
6 M = 5
7
8 m.uncset.cons = pe.ConstraintList()
9
10 for i in range(M):
11     expr = pe.inequality(0.9*demand[i],
12                          m.demand[i],
13                          1.1*demand[i])
14     m.uncset.cons.add(expr=expr)
15
16 # Add objective
17 expr = 0
18 for i in range(N):
19     for j in range(M):
20         expr += cost_transport[i][j]*m.y[i, j]
21     expr += cost_facility[i]*m.x[i]
22 m.obj = pe.Objective(expr=expr, sense=pe.minimize)
23
24 # Add constraints
25 def sum_y_rule(m, j):
26     return sum(m.y[i, j] for i in range(N)) == m.demand[j]
27 m.sum_y = pe.Constraint(range(M), rule=sum_y_rule)
28
29 def max_demand_rule(m, i):
30     lhs = sum(m.y[i, j] for j in range(M))
31     return lhs <= max_dem[i]*m.x[i]
32 m.max_dem = pe.Constraint(range(N), rule=max_demand_rule)

```

The coefficients of the linear decision rules can be accessed through the name of the adjustable variable, the name of the uncertain parameter and the postfix `_coef`. E.g. for the problem above `m.y_demand_coef.pprint()` will print:

```

1 y_demand_coef : Size=100, Index=y_demand_coef_index
2   Key          : Lower : Value : Upper : Fixed : Stale : Domain
3   (0, 0, 0)    : None  : 0.0  : None  : False  : False  : Reals
4   (0, 0, 1)    : None  : 0.0  : None  : False  : False  : Reals
5   (0, 0, 2)    : None  : 0.0  : None  : False  : False  : Reals
6   ...

```

5.3 Solution times

We solve the portfolio, knapsack, pooling, and facility location problems with both the reformulation and cutting plane solver for ellipsoidal and polyhedral uncertainty sets and using both the library approach to generating uncertainty sets as well as the generic, Pyomo constraint-based approach. We solve the production planning and drill scheduling problems using the reformulation solver with uncertainty sets based on both standard and warped Gaussian processes. We solve 30 instances with different uncertainty set sizes for each case study.

Table 2 shows the median time in milliseconds taken to solve each problem for a given uncertainty set geometry and solver. The reformulation solver generally outperforms the cutting plane solver with median times of 74ms and 330ms respectively. An exception is the the non-linear, non-convex pooling problem with an

Table 2 Median time in milliseconds taken to solve the six example problems with different uncertainty set geometries. Times are shown for the reformulation and cutting plane solvers and both combined (Overall). The median time taken to *reformulate* the model is shown in parantheses. The cutting plane solver is not applicable (NA) for Gaussian process-based sets and the reformulation solver cannot solve the facility problem with ellipsoidal sets to optimality (–).

Problem	Unc. Set	Reformulation		Cuts		Overall	
Knapsack	Polyhedral	54	(3.7)	272	(1.4)	85	(1.9)
	Ellipsoidal	50	(2.2)	183	(1.4)	91	(1.8)
Pooling	Polyhedral	74	(7.7)	329	(2.9)	173	(7.4)
	Ellipsoidal	638	(11.6)	331	(2.9)	349	(6.7)
Portfolio	Polyhedral	50	(3.0)	276	(0.8)	126	(2.3)
	Ellipsoidal	49	(2.7)	1659	(0.8)	129	(1.2)
Facility	Polyhedral	261	(106.0)	13353	(37.1)	5588	(74.5)
	Ellipsoidal	–	(–)	31275	(71.4)	31275	(71.4)
Planning	Standard	2776	(871.7)	NA		2776	(871.7)
	Warped	8536	(997.9)	NA		8536	(997.9)
Drilling	Standard	13646	(272.8)	NA		13646	(272.8)
	Warped	75325	(577.8)	NA		75325	(577.8)
Overall		74	(7.5)	330	(2.8)	271	(3.2)

ellipsoidal set. For this instance, the cutting plane solver achieves significantly better results, which is in line with previous work on robust pooling problems (Wiebe et al. 2019). Similarly, for the facility location problem with an ellipsoidal set, the reformulation approach does not solve the problem to optimality within a 10 minute time frame, while the cutting plane solver does. For the production planning and drills scheduling examples only the reformulation solver can be applied. The Wolfe duality-based reformulation for warped Gaussian processes generally takes longer to solve than the chance-constraint reformulation for standard Gaussian processes. Note that most of this time is the time taken by the subsolvers. The transformations which ROmodel performs are generally very quick: the median transformation time across all instances is 3.2 milliseconds, while the maximum transformation time is 1.7 seconds. The median transformation time for each problem and geometry/solver is shown in parantheses in Table 2.

6 Conclusion

ROmodel formulates robust versions of common optimization problems. The modeling environment it provides makes (adjustable) robust optimization methods more readily available to practitioners and makes trying different solution approaches and uncertainty sets very easy. ROmodel is open source and available free of charge and could play a vital role as a platform for prototyping novel robust optimization algorithms and comparing them to existing approaches.

Acknowledgements This work was funded by the Engineering & Physical Sciences Research Council (EPSRC) Center for Doctoral Training in High Performance Embedded and Distributed Systems (EP/L016796/1), an EPSRC/Schlumberger CASE studentship to J.W. (EP/R511961/1, voucher 17000145), and an EPSRC Research Fellowship to R.M. (EP/P016871/1).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- AIMMS. <http://aimms.com>. Accessed: 2021-03-25
- Adhya N, Tawarmalani M, Sahinidis NV (1999) A Lagrangian approach to the pooling problem. *Ind Eng Chem Res* 38(5):1956–1972
- Ben-Tal A, Nemirovski A (1999) Robust solutions of uncertain linear programs. *Op Res Lett* 25(1):1–13
- Bertsimas D, Gupta V, Kallus N (2018) Data-driven robust optimization. *Math Prog* 167:235–292
- Bertsimas D, Sim M (2004) The price of robustness. *Op Res* 52:35–53
- Bhosekar A, Ierapetritou M (2018) Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Comput Chem Eng* 108:250–267
- Chen Z, Sim M, Xiong P (2020) Robust stochastic optimization made easy with rsome. *Manag Sci* 66:3329–3339

- Diehl M, Gerhard J, Marquardt W, Mönnigmann M (2008) Numerical solution approaches for robust nonlinear optimal control problems. *Comput Chem Eng* 32:1287–1300
- Dunning IR (2016) Advances in robust and adaptive optimization: algorithms, software, and insights. Ph.D. thesis, Sloan School of Management, MIT
- Goh J, Sim M (2011) Robust optimization made easy with rome. *Op Res* 59:973–985
- Gong J, You F (2018) Resilient design and operations of process systems: nonlinear adaptive robust optimization model and algorithm for resilience analysis and enhancement. *Comput Chem Eng* 116:231–252
- GPy (2012) A Gaussian process framework in python. <http://github.com/SheffieldML/GPy> (since)
- Grossmann IE, Apap RM, Calfa BA, García-Herreros P, Zhang Q (2016) Recent advances in mathematical programming techniques for the optimization of process systems under uncertainty. *Comput Chem Eng* 91:3–14
- Hart WE, Laird CD, Watson JP, Woodruff DL, Hackebeil GA, Nicholson BL, Sirola JD (2017) *Pyomo — Optimization Modeling in Python*, vol. 67. Springer International Publishing
- Hart WE, Watson JP, Woodruff DL (2011) Pyomo: modeling and solving mathematical programs in python. *Math Prog Comput* 3(3):219–260
- Houska B, Diehl M (2013) Nonlinear robust optimization via sequential convex bilevel programming. *Math Prog* 142:539–577
- Isenberg NM, Sirola JD, Gounaris CE Pyros (2020) A pyomo robust optimization solver for robust process design. In: 2020 Virtual AIChE Annual Meeting
- Janak SL, Floudas CA (2005) Advances in robust optimization approaches for scheduling under uncertainty. *Comput Chem Eng* 20(C):1051–1056
- Li J, Misener R, Floudas CA (2012) Scheduling of crude oil operations under demand uncertainty: a robust optimization framework coupled with global optimization. *AIChE J* 58(8):2373–2396
- Li Z, Ierapetritou MG (2008) Robust optimization for process scheduling under uncertainty. *Ind Eng Chem Res* 47(12):4148–4157
- Marc AG, Schöbel (2016) Algorithm engineering in robust optimization. *Algorithm engineering: selected results and surveys* pp. 245–279
- Mutapcic A, Boyd S (2009) Cutting-set methods for robust convex optimization with pessimizing oracles. *Optim Method Softw* 24:381–406
- Ning C, You F (2017) A data-driven multistage adaptive robust optimization framework for planning and scheduling under uncertainty. *AIChE J* 63(10):4343–4369
- Pistikopoulos EN, Barbosa-Povoa A, Lee JH, Misener R, Mitsos A, Reklaitis GV, Venkatasubramanian V, You F, Gani R (2021) Process systems engineering-The generation next? *Comput Chem Eng* 147:107252
- Shang C, Huang X, You F (2017) Data-driven robust optimization based on kernel learning. *Comput Chem Eng* 106:464–479
- Shang C, You F (2018) Distributionally robust optimization for planning and scheduling under uncertainty. *Comput Chem Eng* 110:53–68
- Snelson E, Rasmussen CE, Ghahramani Z (2003) Warped Gaussian processes. In: *NIPS*
- Vayanos P, Jin Q, Elissaios G (2020) Roc++: Robust optimization in c++
- Wiebe J (2020) Drill scheduling. github.com/cog-imperial/drill-scheduling
- Wiebe J (2020) ROGP: Robust GPs in Pyomo. github.com/cog-imperial/rogp
- Wiebe J, Cecilio I, Dunlop J, Misener R (2020) A robust approach to warped Gaussian process-constrained optimization
- Wiebe J, Cecilio I, Misener R (2019) Robust optimization for the pooling problem. *Ind Eng Chem Res*
- Wiebe J, Cecilio I, Misener R (2018) Data-driven optimization of processes with degrading equipment. *Ind Eng Chem Res* 57:17177–17191
- Wiebe J, Misener R (2020) Romodel 0.1.0. github.com/cog-imperial/romodel. 10.5281/zenodo.4715841
- Yuan Y, Li Z, Huang B (2018) Nonlinear robust optimization for process design. *AIChE J* 64(2):481–494
- Zhang Q, Grossmann IE, Heuberger CF, Sundaramoorthy A, Pinto JM (2015) Air separation with cryogenic energy storage: optimal scheduling considering electric energy and reserve markets. *AIChE J* 61(5):1547–1558
- Zhang Y, Feng Y, Rong G (2018) Data-driven rolling-horizon robust optimization for petrochemical scheduling using probability density contours. *Comput Chem Eng* 115:342–360