ROmodel: Modeling robust optimization problems in Pyomo

Johannes Wiebe[1], Ruth Misener[1]

Monday 7th June, 2021

[1]Department of Computing, Imperial College London, London, UK

Funding: EP/L016796/1, EP/R511961/1 no. 17000145, and EP/P016871/1

Robust optimization

Consider a generic robust optimization problem:

$$\min_{\mathrm{x}\in\mathcal{X},\mathrm{y}(\boldsymbol{\xi})} \max_{\boldsymbol{\xi}\in\mathcal{U}(\mathrm{x})} \quad f(\mathrm{x},\mathrm{y}(\boldsymbol{\xi}),\boldsymbol{\xi}) \tag{1a}$$

$$\text{s.t} \quad g(\mathrm{x},\mathrm{y}(\boldsymbol{\xi}),\boldsymbol{\xi}) \leq 0 \qquad \forall \boldsymbol{\xi} \in \mathcal{U}(\mathrm{x}) \tag{1b}$$

where $\mathrm{x} \in \mathbb{R}^n$ are ”here and now” variables, $\mathrm{y}(\boldsymbol{\xi})$ are adjustable variables, and the uncertain parameter vector $\boldsymbol{\xi}$ is bounded by the uncertainty set $\mathcal{U}(\mathrm{x})$.

Robust optimization

Consider a generic robust optimization problem:

$$\min_{x \in \mathcal{X}, y(\boldsymbol{\xi})} \max_{\boldsymbol{\xi} \in \mathcal{U}(x)} \quad f(x, y(\boldsymbol{\xi}), \boldsymbol{\xi}) \tag{1a}$$

$$\text{s.t} \quad g(x, y(\boldsymbol{\xi}), \boldsymbol{\xi}) \leq 0 \qquad \forall \boldsymbol{\xi} \in \mathcal{U}(x) \tag{1b}$$

where $x \in \mathbb{R}^n$ are "here and now" variables, $y(\boldsymbol{\xi})$ are adjustable variables, and the uncertain parameter vector $\boldsymbol{\xi}$ is bounded by the uncertainty set $\mathcal{U}(x)$.

Solution approaches

1. Robust reformulation: based on duality
2. Cutting planes: iterative approach

Existing tools

There are a number of existing tools for solving robust optimization problems:

[1] Goh and Sim (2011), [2] Chen et al. (2020), [3] Isenberg et al. (2020), [4] Vayanos et al. (2020), [5] Dunning (2016)

Existing tools

There are a number of existing tools for solving robust optimization problems:

Solvers

Designed to *solve* RO problems

- ROME [1], RSOME [2]: Matlab
- PyROS [3]: Python/Pyomo
- ROC++ [4]: C++

[1] Goh and Sim (2011), [2] Chen et al. (2020), [3] Isenberg et al. (2020), [4] Vayanos et al. (2020), [5] Dunning (2016)

Existing tools

There are a number of existing tools for solving robust optimization problems:

Solvers

Designed to *solve* RO problems

- ROME [1], RSOME [2]: Matlab
- PyROS [3]: Python/Pyomo
- ROC++ [4]: C++

Modeling languages

Designed for *modeling* RO problems

- JumPeR [5]: Julia/JumP
- AIMMS

[1] Goh and Sim (2011), [2] Chen et al. (2020), [3] Isenberg et al. (2020), [4] Vayanos et al. (2020), [5] Dunning (2016)

Existing tools

There are a number of existing tools for solving robust optimization problems:

Solvers

Designed to *solve* RO problems

- ROME [1], RSOME [2]: Matlab
- PyROS [3]: Python/Pyomo
- ROC++ [4]: C++

Modeling languages

Designed for *modeling* RO problems

- JumPeR [5]: Julia/JumP
- AIMMS

ROmodel

Focuses on modeling. ROmodel is open source, tightly integrated with Pyomo, and allows modeling uncertainty sets with Pyomo constraints.

[1] Goh and Sim (2011), [2] Chen et al. (2020), [3] Isenberg et al. (2020), [4] Vayanos et al. (2020), [5] Dunning (2016)

New modeling objects

ROmodel introduces three new modeling components:

New modeling objects

ROmodel introduces three new modeling components:

- `UncParam`: Similar to Pyomo's `Var` or `Param`; used to model uncertain parameters

New modeling objects

ROmodel introduces three new modeling components:

- `UncParam`: Similar to Pyomo's `Var` or `Param`; used to model uncertain parameters
- `UncSet`: Based on Pyomo's `Block` component; used to model uncertainty sets

New modeling objects

ROmodel introduces three new modeling components:

- `UncParam`: Similar to Pyomo's `Var` or `Param`; used to model uncertain parameters
- `UncSet`: Based on Pyomo's `Block` component; used to model uncertainty sets
- `AdjustableVar`: Very similar to Pyomo's `Var`; used for ”wait and see” variables

New modeling objects

ROmodel introduces three new modeling components:

- `UncParam`: Similar to Pyomo's `Var` or `Param`; used to model uncertain parameters
- `UncSet`: Based on Pyomo's `Block` component; used to model uncertainty sets
- `AdjustableVar`: Very similar to Pyomo's `Var`; used for "wait and see" variables

These components can be used in Pyomo models like any Pyomo modeling component.

Modeling uncertain parameters

Uncertain parameters are the central component of robust optimization problems:

```
import pyomo.environ as pe
import romodel as ro
# Construct model
m = pe.ConcreteModel()
# Add uncertain parameters
m.c = ro.UncParam(range(3), nominal=[0.1, 0.2, 0.3], uncset=m.U)
```

Modeling uncertain parameters

Uncertain parameters are the central component of robust optimization problems:

```python
import pyomo.environ as pe
import romodel as ro
# Construct model
m = pe.ConcreteModel()
# Add uncertain parameters
m.c = ro.UncParam(range(3), nominal=[0.1, 0.2, 0.3], uncset=m.U)
```

Arguments:

- `index`: (optional), `UncParam` can be indexed or not
- `nominal`: specifies nominal values for the uncertain parameters
- `uncset`: specifies an uncertainty set for the uncertain parameters

Modeling uncertainty sets

Uncertainty sets can be specified in two ways:

1. Generic sets with Pyomo constraints

```
# Define uncertainty set & uncertain parameters
m.U = ro.UncSet()
m.c = UncParam(range(2), uncset=m.U, nominal=[0.5, 0.5])
# Add constraints to uncertainty set
m.U.cons1 = Constraint(expr=m.c[0] + m.c[1] <= 1)
m.U.cons2 = Constraint(expr=m.c[0] - m.c[1] <= 1)
```

2. Library sets

```
# Define polyhedral set
m.U = ro.uncset.PolyhedralSet(rhs=[1, 1, 1, 1], mat=[[ 1,  1],
                                                     [ 1, -1],
                                                     [-1,  1],
                                                     [-1, -1]])
```

Modeling uncertain constraints

Users can model uncertain constraints implicitly by using `UncParam` objects in Pyomo constraints.

Consider a deterministic Pyomo constraint:

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

If `c` is uncertain, the robust formulation is:

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

Modeling adjustable variables

Adjustable variable: Decision variable who's value is determined after the uncertainty has been revealed.

Defining adjustable variables in ROmodel with the `AdjustableVar` class:

```
# Define uncertain parameters and adjustable variables
m.w = UncParam(range(3), nominal=[1, 2, 3], uncset=m.U)
m.y = AdjustableVar(range(3), uncparams=[m.w], bounds=(0, 1))
```

Arguments `uncparams` specifies which uncertain parameters are revealed before the decision is made. This can be set individually for each adjustable variable:

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

ROmodel currently only implements linear decision rules.

Solvers

ROmodel includes three solvers:

1. Robust reformulation:

```python
solver = SolverFactory('romodel.reformulation')
```

2. Cutting planes:

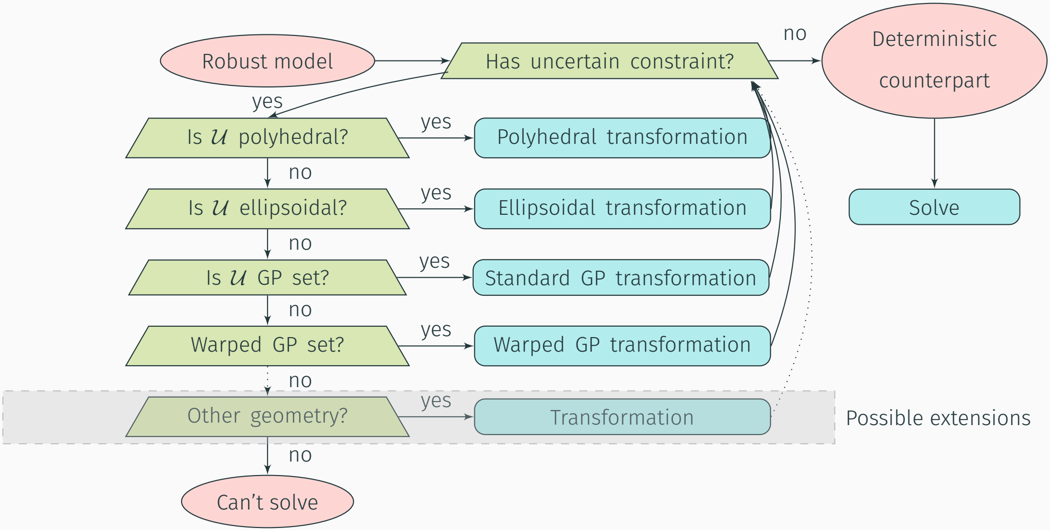
```python
solver = SolverFactory('romodel.cuts')
```

3. Nominal:

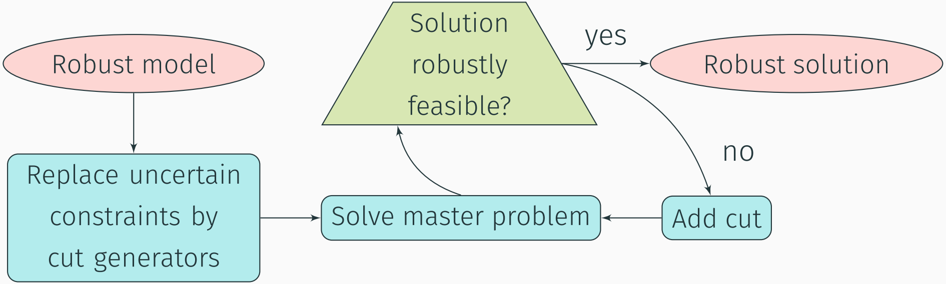
```python
solver = SolverFactory('romodel.nominal')
```

```python
# Set master & sub solver
solver.options['solver'] = 'gurobi'
solver.options['subsolver'] = 'ipopt'
# Solve
solver.solve(m)
```

Solvers: reformulation



Solvers: cutting planes



Extending ROmodel

ROmodel can be extended in a number of ways:

1. Implementing new library uncertainty set
2. Adding new reformulations

Extending ROmodel

ROmodel can be extended in a number of ways:

1. Implementing new library uncertainty set
2. Adding new reformulations

Gaussian process-based uncertainty sets

Gaussian processes are often used as surrogates for uncertain black-box constraints. We have developed reformulation approaches for chance constraints based on (warped) Gaussian processes [1].

[1] Wiebe et al. (2020)

Extending ROmodel: adding library sets

Class which collects relevant data:

```
class EllipsoidalSet(UncSet):
    '''
    Defines an ellipsoidal uncertainty set of shape:
        (param - mu)^T * A * (param - mu) <= 1
    '''
    def __init__(self, mean, cov, *args, **kwargs):
        self.mean = mean
        self.cov = cov
        super().__init__(*args, **kwargs)
```

Make compatible with cutting planes:

```
def generate_cons_from_lib(self, param):
    raise NotImplementedError
```

Extending ROmodel: adding reformulations

```
    def _reformulate(self, c, param, uncset, counterpart):
        """
        Reformulate an uncertain constraint or objective
            c: Constraint or Objective
            param: UncParam
            uncset: UncSet
            counterpart: Block
        """
        return counterpart
```

Make compatible with generic uncertainty sets (`UncSet`):

```
    def _check_applicability(self, uncset):
        """
        Returns `True` if the reformulation is applicable to `uncset`
        """
        return uncset.__class__ == GPSet
```

Extending ROmodel: Gaussian process-based sets

Train (warped) Gaussian process with GPy:

```python
import GPy
# Set up kernel
kernel = GPy.kern.RBF(input_dim=1)
# Set up GP and train
gp = GPy.models.WarpedGP(x, y, kernel=kernel, warping_terms=3)
gp.optimize()

```

Use GPy model to construct uncertainty set:

```python
from romodel.uncset import WarpedGPSet
m.z = pe.Var(range(3), within=pe.NonNegativeReals)
# Set up GP-based uncertainty set
m.uncset = WarpedGPSet(gp, m.z, 0.95)

```

Case studies

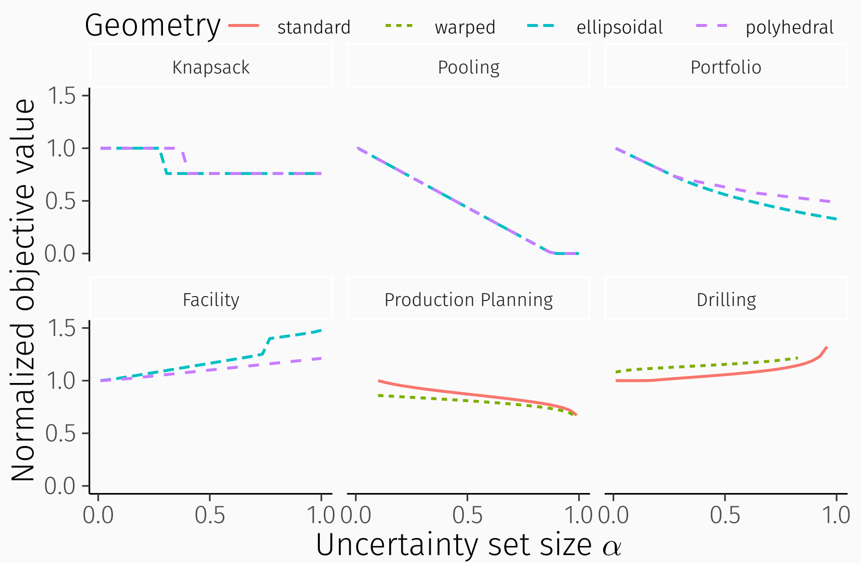
1. Portolio optimization with uncertain returns [1],
2. Knapsack problem with uncertain item weights,
3. Pooling problem instance with uncertain product demands [2],
4. Capacitated facility location problem with uncertain demand,
5. Production planning problem with prices dependent on uncertain black-box function [3],
6. Drill scheduling problem with uncertain degradation rate dependent on black-box function [3]

[1] Bertsimas and Sim (2004), [2] Adhya et al. (1999), [3] Wiebe et al. (2020)

Results: median times

		Reformulation	Cuts	Overall
Knapsack	Polyhedral	54	272	85
	Ellipsoidal	50	183	91
Pooling	Polyhedral	74	329	173
	Ellipsoidal	638	331	349
Portfolio	Polyhedral	50	276	126
	Ellipsoidal	49	1659	129
Facility	Polyhedral	261	13353	5588
	Ellipsoidal	–	31275	31275
Planning	Standard	2776	NA	2776
	Warped	8536	NA	8536
Drilling	Standard	13646	NA	13646
	Warped	75325	NA	75325
Overall		74	330	271

Results: price of robustness



ROmodel...

- ...makes modeling & solving robust problems with Pyomo intuitive.
- ...makes it easy to compare different uncertainty sets & solution approaches.
- ...is open source and can be extended to other uncertainty sets & reformulations.

Thank you!

Try ROmodel: https://github.com/cog-imperial/romodel

Paper: https://arxiv.org/abs/2105.08598

Twitter: @CogImperial

Funding: EP/L016796/1, EP/R511961/1 no. 17000145, and EP/P016871/1

References

Adhya, N., Tawarmalani, M., and Sahinidis, N. V. (1999). A Lagrangian Approach to the Pooling Problem. *Ind. Eng. Chem. Res.*, 38(5):1956–1972.

Bertsimas, D. and Sim, M. (2004). The price of robustness. *Oper. Res.*, 52:35–53.

Chen, Z., Sim, M., and Xiong, P. (2020). Robust stochastic optimization made easy with rsome. *Management Science*, 66:3329–3339.

Dunning, I. R. (2016). *Advances in Robust and Adaptive Optimization: Algorithms, Software, and Insights*. PhD thesis, Sloan School of Management, MIT.

Goh, J. and Sim, M. (2011). Robust optimization made easy with rome. *Oper. Res.*, 59:973–985.

Isenberg, N. M., Siirola, J. D., and Gounaris, C. E. (2020). Pyros: A pyomo robust optimization solver for robust process design. In *2020 Virtual AIChE Annual Meeting*.

Vayanos, P., Jin, Q., and Elissaios, G. (2020). Roc++: Robust optimization in c++.

Wiebe, J., Cecílio, I., Dunlop, J., and Misener, R. (2020). A robust approach to warped Gaussian process-constrained optimization.