Online Supplementary of "RSOME in Python: An Open-Source Package for Robust Stochastic Optimization Made Easy"

1 The Robust Optimization Framework

The `ro` module in RSOME is designed for robust optimization problems, where tailored modeling tools are developed for specifying random variables, uncertainty sets, the objective function or constraints under the worst-case scenarios that may arise from the uncertainty set, as well as decision rules for recourse decisions. The general framework is given by

$$\begin{array}{ll} \underset{\boldsymbol{x},\,\boldsymbol{y}}{\text{minimize}} & \max\limits_{\boldsymbol{z}\in\mathcal{Z}_0}\left\{\boldsymbol{a}_0^\top(\boldsymbol{z})\boldsymbol{x}+\boldsymbol{b}_0^\top\boldsymbol{y}(\boldsymbol{z})+c_0(\boldsymbol{z})\right\} \\ \text{subject to} & \max\limits_{\boldsymbol{z}\in\mathcal{Z}_m}\left\{\boldsymbol{a}_m^\top(\boldsymbol{z})\boldsymbol{x}+\boldsymbol{b}_m^\top\boldsymbol{y}(\boldsymbol{z})+c_m(\boldsymbol{z})\right\}\leq 0 \quad \forall m\in\mathcal{M} \\ & y_i\in\mathcal{L}(\mathcal{J}_i) \qquad\qquad\qquad\qquad\qquad\qquad \forall i\in[I_y] \\ & \boldsymbol{x}\in\mathcal{X}. \end{array}$$

Here, parameters of proper dimensions,

$$\boldsymbol{a}_m(\boldsymbol{z}):=\boldsymbol{a}_m^0+\sum_{j\in[J]}\boldsymbol{a}_m^j z_j \quad\text{and}\quad c_m(\boldsymbol{z}):=c_m^0+\sum_{j\in[J]}c_m^j z_j,$$

are defined similarly as in the distributionally robust optimization framework and $\mathcal{X}$ is an SOC representable feasible set of the here-and-now decision $\boldsymbol{x}$. The wait-and-see decision $\boldsymbol{y}$ is restricted to a simpler and easy-to-optimize affine function in the following form:

$$\mathcal{L}(\mathcal{J}):=\left\{y:\mathbb{R}^{[J]}\mapsto\mathbb{R}\;\middle|\;y(\boldsymbol{z})=y^0+\sum_{j\in\mathcal{J}}y^j z_j\right\},$$

where the prescribed subset $\mathcal{J}\subseteq[J]$ includes indices of those random components $\tilde{z}_1,\ldots,\tilde{z}_J$ of $\tilde{\boldsymbol{z}}$ to which a particular non-anticipative decision can adapt.

2 Vehicle Pre-Allocation Problem Revisit

In this section, we revisit the vehicle pre-allocation problem and use the `ro` framework to build a robust model and a sample robust model (proposed by Bertsimas et al. 2022b) for this problem.

We also look at the sample robust model from the distributionally robust optimization perspective and relate it to other data-driven distributionally robust models.

2.1 The Robust Model

The robust model is given by

$$\begin{array}{lll} \underset{\boldsymbol{x},\,\boldsymbol{y}}{\text{minimize}} & \max\limits_{\boldsymbol{d}\in\mathcal{Z}} \left\{ \sum\limits_{i\in[I]}\sum\limits_{j\in[J]} (c_{ij}-r_j)x_{ij} + \sum\limits_{j\in[J]} r_j y_j(\boldsymbol{d}) \right\} & \\ \text{subject to} & y_j(\boldsymbol{d}) \geq \sum\limits_{i\in[I]} x_{ij} - d_j & \forall \boldsymbol{d}\in\mathcal{Z},\ j\in[J] \\ & y_j(\boldsymbol{d}) \geq 0 & \forall \boldsymbol{d}\in\mathcal{Z},\ j\in[J] \\ & y_j \in \mathcal{L}([J]) & \forall j\in[J] \\ & \sum\limits_{j\in[J]} x_{ij} \leq q_i & \forall i\in[I] \\ & x_{ij} \geq 0 & \forall i\in[I],\ j\in[J]. \end{array} \tag{1}$$

Here, the wait-and-see decision $\boldsymbol{y}$ is approximated by a linear decision rule $\mathcal{L}([J])$, implying that each y_j affinely depends on the demand realization $\boldsymbol{d}$. The uncertainty set $\mathcal{Z}$ is a box with upper and lower bounds identified as follows.

```
import pandas as pd

data = pd.read_csv('taxi_rain.csv')          # read data from the csv file

demand = data.loc[:, 'Region1':'Region10']   # taxi demand data

d_ub = demand.max().values                   # upper bound of demand
d_lb = demand.min().values                   # lower bound of demand
```

The robust optimization model can be implemented with the following code segment.

```
from rsome import ro                         # import the ro module
from rsome import grb_solver as grb          # import the Gurobi interface

model = ro.Model()                           # create an RO model

d = model.rvar(J)                            # create an array of random variables
zset = (d <= d_ub, d >= d_lb)                # define a box uncertainty set

x = model.dvar((I, J))                       # define here-and-now decisions as array x
```

```
y = model.ldr(J)                                # define linear decision rules as array y
y.adapt(d)                                      # y affinely adapts to d

model.minmax(((c-r)*x).sum() + r@y, zset)       # minimize the worst-case objective
model.st(y >= x.sum(axis=0) - d, y >= 0)        # robust constraints
model.st(x.sum(axis=1) <= q, x >= 0)            # deterministic constraints

model.solve(grb)                                # solve the model with Gurobi
```

2.2 The Sample Robust Model Using the `ro` Framework

Suppose there are a collection $\{\hat{\boldsymbol{d}}_1, \ldots, \hat{\boldsymbol{d}}_S\}$ of historical demand samples available. For a general two-stage problem, Bertsimas et al. (2022b) recently propose a sample robust model with a specific linear decision rule called multi-policy approximation. In particular, for the vehicle pre-allocation problem, the corresponding sample robust model can be cast as follows:

$$
\begin{aligned}
\underset{\boldsymbol{x},\,\boldsymbol{y}}{\text{minimize}} \quad & \sum_{i\in[I]}\sum_{j\in[J]}(c_{ij}-r_j)x_{ij} + \frac{1}{S}\sum_{s\in[S]} a_s \\
\text{subject to} \quad & a_s \geq \sum_{j\in[J]} r_j y_{sj}(\boldsymbol{d}) && \forall \boldsymbol{d}\in\mathcal{Z}_s, s\in[S] \\
& y_{sj}(\boldsymbol{d}) \geq \sum_{i\in[I]} x_{ij} - d_j && \forall \boldsymbol{d}\in\mathcal{Z}_s,\ j\in[J],\ s\in[S] \\
& y_{sj}(\boldsymbol{d}) \geq 0 && \forall \boldsymbol{d}\in\mathcal{Z}_s,\ j\in[J],\ s\in[S] \\
& y_{sj} \in \mathcal{L}([J]) && \forall j\in[J],\ s\in[S] \\
& \sum_{j\in[J]} x_{ij} \leq q_i && \forall i\in[I] \\
& x_{ij} \geq 0 && \forall i\in[I],\ j\in[J].
\end{aligned}
\tag{2}
$$

Here, $\boldsymbol{a}\in\mathbb{R}^S$ is a vector of intermediate variables for the worst-case costs in each scenario and an uncertainty set $\mathcal{Z}_s = \{\boldsymbol{d}\in[\underline{\boldsymbol{d}},\bar{\boldsymbol{d}}] \mid \|\boldsymbol{d}-\hat{\boldsymbol{d}}_s\|\leq\varepsilon\}$—an ε-neighbourhood defined by a general norm $\|\cdot\|$—is constructed around each demand sample $\hat{\boldsymbol{d}}_s$. The multiple-policy approximation then allows different affine dependencies around different samples, leading to the two-dimensional decision rules $(y_{sj}(\boldsymbol{d}))_{s\in[S],j\in[J]}$. Such a sample robust model (assuming the parameter $\varepsilon = 0.25$) is implemented as shown in the following code segment.

```
from rsome import ro                                    # import the ro module
from rsome import norm                                  # import the norm function
from rsome import grb_solver as grb                     # import the Gurobi interface

dhat = demand.values                                    # sample demand as an array
S = dhat.shape[0]                                       # sample size of the dataset
epsilon = 0.25                                          # parameter of robustness

model = ro.Model()                                      # create an RO model

d = model.rvar(J)                                       # random variable d
a = model.dvar(S)                                       # variable as the recourse cost
x = model.dvar((I, J))                                  # here-and-now decision x
y = model.ldr((S, J))                                   # linear decision rule y
y.adapt(d)                                              # y affinely adapts to d

model.min(((c-r)*x).sum() + (1/S)*a.sum())              # minimize the objective
for s in range(S):
    zset = (d <= d_ub, d >= d_lb,
            norm(d - dhat[s]) <= epsilon)               # sample-wise uncertainty set
    model.st((a[s] >= r@y[s]).forall(zset))             # constraints for the sth sample
    model.st((y[s] >= x.sum(axis=0) - d).forall(zset))  # constraints for the sth sample
    model.st((y[s] >= 0).forall(zset))                  # constraints for the sth sample
model.st(x.sum(axis=1) <= q, x >= 0)                    # constraints

model.solve(grb)                                        # solve the model by Gruobi
```

We would like to highlight that the `ro` module enables users to specify different uncertainty sets for the objective function and each of the constraints: in the above sample robust model, different uncertainty sets are defined around samples and these sets for constraints can be easily specified by calling the `forall()` method. Such a feature makes the robust optimization framework more flexible than that in the MATLAB version and can be used to address a rich range of robust models, including the distributional interpretation of robust formulation (Xu et al. 2012), as well as the notion of Pareto robustly optimal solution (de Ruiter et al. 2016).

2.3 The Sample Robust Model Using the `dro` Framework

As pointed out by Bertsimas et al. (2022b), the sample robust model can also be cast as the following distributionally robust optimization problem:

$$
\begin{array}{lll}
\underset{\boldsymbol{x},\,\boldsymbol{y}}{\text{minimize}} & \displaystyle\sum_{i\in[I]}\sum_{j\in[J]}(c_{ij}-r_j)x_{ij}+\sup_{\mathbb{P}\in\mathcal{F}}\mathbb{E}_{\mathbb{F}}\left[\sum_{j\in[J]} r_j y_j(\tilde{s},\tilde{\boldsymbol{d}})\right] & \\
\text{subject to} & y_j(\tilde{s},\boldsymbol{d})\geq \displaystyle\sum_{i\in[I]} x_{ij}-d_j & \forall \boldsymbol{d}\in\mathcal{Z}_s,\ s\in[S],\ j\in[J] \\
& y_j(\tilde{s},\boldsymbol{d})\geq 0 & \forall \boldsymbol{d}\in\mathcal{Z}_s,\ s\in[S],\ j\in[J] \\
& y_j\in\bar{\mathcal{A}}(\mathcal{C},\mathcal{J}) & \forall j\in[J] \\
& \displaystyle\sum_{j\in[J]} x_{ij}\leq q_i & \forall i\in[I] \\
& x_{ij}\geq 0 & \forall i\in[I],\ j\in[J],
\end{array}
\tag{3}
$$

where $\tilde{s}$ is a random scalar corresponds to demand samples. The ambiguity set is given by

$$
\mathcal{F}=\left\{\mathbb{P}\in\mathcal{P}_0(\mathbb{R}^J\times[S])\;\middle|\;\begin{array}{ll}(\tilde{\boldsymbol{d}},s)\sim\mathbb{P} & \\ \mathbb{P}[\tilde{\boldsymbol{d}}\in\mathcal{Z}_s\mid\tilde{s}=s]=1 & \forall s\in[S]\\ \mathbb{P}[\tilde{s}=s]=w_s & \forall s\in[S]\end{array}\right\},
\tag{4}
$$

where for each scenario $s\in[S]$, the weight is $w_s=1/S$ and the corresponding support set $\mathcal{Z}_s=\{\boldsymbol{d}\in[\underline{\boldsymbol{d}},\bar{\boldsymbol{d}})\mid\|\boldsymbol{d}-\hat{\boldsymbol{d}}_s\|\leq\varepsilon\}$ is an ε-neighbourhood around the demand sample $\hat{\boldsymbol{d}}_s$.[1] The multiple-policy approximation is equivalent to the event-wise recourse adaptation that states $y_j\in\bar{\mathcal{A}}(\{\{1\},\cdots,\{S\}\},[J])$, suggesting that y_j affinely depends on the demand realization and around each sample, the affine adaptation is different. The distributionally robust model can be implemented by the sample code below.

```
from rsome import dro                    # import the dro module
from rsome import norm                   # import the norm function
from rsome import E                      # import the expectation notion
```

[1] If the uncertainty parameter $\varepsilon=0$, the sample robust model is equivalent to the SAA approach. Conceptually, the sample robust model is based on a specific type of Wasserstein metric. We refer to Chen et al. (2020, section 5.5) for models based on other types of Wasserstein metric and their approximations of the wait-and-see decisions.

```
4  from rsome import grb_solver as grb                  # import the Gurobi interface
5
6  dhat = demand.values                                 # sample demand as an array
7  S = dhat.shape[0]                                    # sample size of the dataset
8  epsilon = 0.25                                       # parameter of robustness
9  w = 1/S                                              # weights of scenarios
10
11 model = dro.Model(S)                                 # a DRO model with S scenarios
12
13 d = model.rvar(J)                                    # random variable d
14 fset = model.ambiguity()                             # create an ambiguity set
15 for s in range(S):                                   # for each scenario
16     fset[s].suppset(d <= d_ub, d >= d_lb,
17                     norm(d - dhat[s]) <= epsilon)    # define the support set
18 pr = model.p                                         # an array of scenario weights
19 fset.probset(pr == w)                                # specify scenario weights
20
21 x = model.dvar((I, J))                               # here-and-now decision x
22 y = model.dvar(J)                                    # wait-and-see decision y
23 y.adapt(d)                                           # y affinely adapts to d
24 for s in range(S):
25     y.adapt(s)                                       # y adapts to each scenario s
26
27 model.minsup(((c-r)*x).sum() + E(r@y), fset)         # the worst-case expectation
28 model.st(y >= x.sum(axis=0) - d, y >= 0)             # robust constraints
29 model.st(x.sum(axis=1) <= q, x >= 0)                # deterministic constraints
30
31 model.solve(grb)                                     # solve the model by Gruobi
```

Recall the `ro` framework in Section 2.2, the decision rules $(y_{sj}(\boldsymbol{d}))_{s\in[S],j\in[J]}$ therein is defined as a two-dimensional array. Here, in the the `dro` framework, $\boldsymbol{y}$ is one-dimensional and the multiple-policy adaptation is defined by a loop in lines 24 and 25, where event-wise affine adaptation is automatically created by calling the `adapt()` method, with `s` being the sample index.

2.4 Comparison of Models

The robust model (1) with support information, sample robust model (2) with sample information, and distributionally robust model with side information—problem (2) in the main paper—all admit a tractable deterministic reformulation as a linear or second-order cone (SOC) program that is well recognized by the external solver. The reformulations, however, differ largely in terms of the problem size, auxiliary variables, constraints, as well as coefficients; see a summary in Table 1. This reminds us the lack of intuition in the tedious and error-prone transition from (distributionally) robust models to their deterministic mathematical equivalents, and more importantly, shows the

need of an accompanying technology to free modelers/practitioners from the transition procedure.

Model	Support information	Sample information	Side information
Number of variables	373	44132	4889
Continuous/binaries/integers	373/0/0	44132/0/0	4889/0/0
Number of linear constraints	232	17789	1930
Inequalities/equalities	211/21	16172/1617	502/1428
Number of coefficients	823	92526	10759
Number of SOC constrains	0	1617	840

Table 1. Summary of deterministic reformulations.

Basic information of the deterministic reformulation (in a standard form) of a (distributionally) robust model can be retrieved by calling the `do_math()` method the `Model` object; see below.

```
dc = model.do_math()                # deterministic counterpart of the model
formula = dc.show()                 # formula of the deterministic counterpart
```

Here, the `show()` method further exports detailed information of the deterministic reformulation as a Pandas `DataFrame`, shown in Figure 1. The `DataFrame` contains coefficients of the objective function (`Obj`) and linear constraints (`LC`), as well as upper/lower bounds (`UB`/`LB`) and types (`Type`) of decision variables. Such data can be conveniently processed by functions and operations provided by the Pandas library, and could be useful for debugging or exploring the problem's structure.

2.5 Alternative Data-Driven Approaches

To incorporate with side information, Bertsimas et al. (2022a) propose to adjust the weights $\boldsymbol{w}$ of samples in the original sample robust model of Bertsimas et al. (2022b), where the robustness parameter ε is used to control the distance to (or equivalently, admissible deviation from) the sample point. Indeed, as pointed out by Bertsimas et al. (2022a), the ambiguity set (4) corresponds to a variety of data-driven approaches, provided that the robustness parameter `epsilon` and weights

```
         x1   x2   x3   x4  ... x372 x373 sense constant
Obj       1    0    0    0  ...    0    0     -        -
LC1       0    1    0    0  ...    0    0    <=       -0
LC2       0    0    1    0  ...    0    0    <=       -0
...     ...  ...  ...  ...  ...  ...  ...   ...      ...
UB      inf  inf  inf  inf  ...    0    0     -        -
LB     -inf    0    0    0  ... -inf -inf     -        -
Type      C    C    C    C  ...    C    C     -        -

[236 rows x 375 columns]
```

Figure 1. Information of the deterministic reformulation of the robust model (1).

`w` are properly specified; see a summary in Table 2. With the help of third-party packages (*e.g.*, PyTorch, Scikit-Learn and TensorFlow) in Python ecosystem, it is easy to implement various machine learning methods, such as K-nearest neighbors, kernel regression, classification and regression tree, and random forest, for determining the weight factors $\boldsymbol{w}$. For more detail, we refer interested readers to Bertsimas et al. (2022a) and Bertsimas and Kallus (2020). Equipped with the distributionally robust optimization framework in RSOME, *different* data-driven approaches in Table 2 can be implemented by using the *same* sample code as that in Section 2.3.

	`w` $= 1/S$	`w` from machine learning
`epsilon` $= 0$	SAA	Bertsimas and Kallus (2020)
`epsilon` > 0	Bertsimas et al. (2022b)	Bertsimas et al. (2022a)

Table 2. Data-driven approaches.

References

Bertsimas, Dimitris, Nathan Kallus. 2020. From predictive to prescriptive analytics. *Management Science* **66**(3) 1025–1044.

Bertsimas, Dimitris, Christopher McCord, Bradley Sturt. 2022a. Dynamic optimization with side information. *Forthcoming in European Journal of Operational Research.*

Bertsimas, Dimitris, Shimrit Shtern, Bradley Sturt. 2022b. Two-stage sample robust optimization. *Operations Research* **70**(1) 624–640.

Chen, Zhi, Melvyn Sim, Peng Xiong. 2020. Robust stochastic optimization made easy with RSOME. *Management Science* **66**(8) 3329–3339.

de Ruiter, Frans, Ruud Brekelmans, Dick den Hertog. 2016. The impact of the existence of multiple adjustable robust solutions. *Mathematical Programming* **160**(1) 531–545.

Xu, Huan, Constantine Caramanis, Shie Mannor. 2012. A distributional interpretation of robust optimization. *Mathematics of Operations Research* **37**(1) 95–110.