第 10 章 鲁棒优化模型求解（Model implementation）

汤勤深，熊鹏

随着鲁棒优化方法的价值不断地被学界和业界发现，出现了越来越多的语言包支持鲁棒优化模型的直接求解。其中，有基于 Julia 的 JuMPeR (Dunning et al., 2017)，基于 C++ 语言的 ROC (https://sites.google.com/site/meilinzhang1985/platform)，基于 Matlab 的XProg和 YALMIP 以及 XProg 的升级版 RSOME。JUMPeR 和 YALMIP 主要支持经典鲁棒优化模型，而 ROC 和 RSOME 既支持经典鲁棒优化模型也支持分布鲁棒优化模型，并且还可以比较简洁地定义各种决策规则。在本章中，我们将简要介绍如何在 JUMPeR 和 YALMIP 上对鲁棒优化模型求解而着重介绍如何用 RSOME 求解不同形式的鲁棒优化模型。由于编码原因，本章代码块部分中的 ξ 由 “xi" 进行表示。

10.1 鲁棒模型在 JuMPeR 上的实现（Implementation on JuMPeR）

JuMPeR全称 Robust Optimization with JuMP。是基于 Julia 中 JuMP 而开发的一个用来求解鲁棒优化模型的语言包。Julia是一门免费和开源的编程语言，于 2018 年 8 月 1 日正式发布。

JuMPeR 可以处理大部分的经典鲁棒优化问题：

$$\begin{aligned} &\min\ g(\boldsymbol{x}) \\ &\text{s.t. } f_i(\boldsymbol{x},\boldsymbol{\xi}) \le 0,\ \forall \boldsymbol{\xi} \in \mathcal{U} \triangleq \{\boldsymbol{\xi} : h(\boldsymbol{\xi}) \le 0\}, i \in [I], \end{aligned} \tag{10.1}$$

其中，在 JuMPeR 中，鲁棒优化模型不能包含以下几种情况：

1. 不能有二次项，也即不能有变量乘变量（$x * y$），不确定项乘不确定项（$\xi * \varepsilon$）。但是允许不确定项和变量相乘（$\xi * x$）。
2. 目标函数中不能包含不确定项。如果有的话，通过引入辅助变量将其转为约束条件。
3. 在有不确定性的约束中不支持宏（macros），所以必须使用 “addConstraint”。

使用 JuMPeR 时，先用

```
using JuMPeR
```

申明调用 JuMPeR，并且用

```
m = RobustModel()
```

来定义模型 m。之后可以定义变量“@defVar()”、约束“addConstraint()”、不确定集“@defUnc()”，或者设定目标函数“setObjective()”。举个例子：

示例 10.1:考虑如下鲁棒优化问题：

$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & \xi_1 x_1 + x_2 \le 2 \quad \xi_1 \in [0.3, 0.5] \\ & \xi_2 x_1 + x_2 \le 6 \quad \xi_2 \in [0, 2] \\ & x_1, x_2 \ge 0, \end{aligned}$$

其 JuMPeR 程序为：

```
# 调用JuMPeR
using JuMPeR

# 定义鲁棒模型$m$
m = RobustModel()

# 定义决策变量$x_1, x_2$
@defVar(m, x[1:2] >= 0)

# 定义不确定参数$\xi_1, \xi_2$
@defUnc(m, 0.3 <= xi1 <= 0.5)
@defUnc(m, 0.0 <= xi2 <= 2.0)

# 设定目标函数
setObjective(m, :Max, x[1] + x[2])

# 添加约束条件
addConstraint(m, xi1*x[1] + 1*x[2] <= 2.0)
addConstraint(m, xi2*x[1] + 1*x[2] <= 6.0)

# 求解鲁棒模型
status = solveRobust(m)

# 输出解
println(getValue(x[1]))  # = 2.6666
println(getValue(x[2]))  # = 0.6666
```

由此，可以看出，类似 JuMPeR 这种编程语言包，已经最大化地节约了编程时间。只要模型满足一定的条件（标准型），即可直接输入进行求解。然而，JuMPeR 暂时只能对经典鲁棒优化问题求解。而对于分布鲁棒优化问题，则需要先手动求对偶，化成 JuMPeR 可以识别的形式之后才可直接编程求解。同样的，接下来要介绍的 YALMIP 也存在这个问题。不过，这些语言包已经大大减少了编程时间。

10.2 鲁棒优化在 YALMIP 和 CVX 上的实现

YALMIP和CVX都是依托于 MATLAB 而建的优化语言包，旨在用最直观简洁的语言进行优化模型的编程和求解。其中，YAPMIP 主要由林雪平大学（Linköping University）的 Johan Löfberg 教授，而 CVX 由斯坦福大学的 Stephen P .Boyd 和 Michael C. Grant 开发和维护。除 YALMIP 自有部分优化器外，二者主要都通过调用开源（比如 OR-Tools, CyLP）或者商业优化器（比如 Gurobi, MOSEK）对模型进行求解。这两个优化语言包功能都非常强大，各有其长短。感兴趣的读者可以到官网（https://yalmip.github.io/）和（http://cvxr.com/cvx/）进行了解。现阶段，YALMIP 支持经典鲁棒优化模型（当然模型本身必须是凸问题），而如果要用 CVX 得化成相应的 LP，SOCP，或者 SDP 才能求解。在这一节中，我们主要介绍如何用 YAPMIP 这个语言包进行经典鲁棒模型的求解。

YALMIP 可以直接处理以下鲁棒优化问题：

$$\begin{aligned} &\min \max_{\boldsymbol{\xi}} g(\boldsymbol{x}, \boldsymbol{\xi}) \\ &\text{s.t. } f_i(\boldsymbol{x}, \boldsymbol{\xi}) \leq 0, \forall \boldsymbol{\xi} \in \mathcal{U} \triangleq \{\boldsymbol{\xi} : h(\boldsymbol{\xi}) \leq 0\}, i \in [I], \end{aligned} \tag{10.2}$$

其中，约束和不确定集 $\mathcal{U}$ 需满足以下几种情况之一：

1. 给定 $\boldsymbol{x}$，每一个约束（elementwise constraints）都仿射依赖于不确定变量 $\boldsymbol{\xi}$，并且不确定集为多面（polytopic）或者锥（conic）集。
2. 每一个约束都仿射依赖于 $\boldsymbol{\xi}$，并且 $\boldsymbol{\xi}$ 在一个范数球（norm-ball, $p = 1, 2, \infty$）。
3. 约束条件为锥约束并且仿射依赖于 $\boldsymbol{\xi}$，而不确定集为多面集。

另外 YALMIP 在还有某些特殊条件下适用。详情可参考https://yalmip.github.io/tutorial/robustoptimization/。

YALMIP 通过 sdpvar 来定义变量，uncertain 来定义不确定集，optimize 来进行问题求解（默认为最小化问题）。接下来，我们举两个简单的例子来阐述如何在 YALMIP 上进行鲁棒优化模型的求解。

示例 10.2:考虑如下简单鲁棒优化问题：

$$\begin{aligned} \max \quad & x \\ \text{s.t.} \quad & x + \xi \leq 1, \forall \xi \in [-0.5, 0.5]. \end{aligned} \tag{10.3}$$

此问题的 YALMIP 的程序如下：

```
sdpvar x xi
F = [x + xi <= 1];
W = [-0.5 <= xi <= 0.5, uncertain(w)];
objective = -x;
sol = optimize(F + W,objective)
```

对于不确定集为锥的问题，亦可用类似的方法求解：

示例 10.3:考虑如下鲁棒锥优化问题：

$$\begin{aligned} \max \quad & x \\ \text{s.t.} \quad & x + \boldsymbol{\xi}'\mathbf{1} \leq 1, \ \forall \boldsymbol{\xi} \in \mathcal{U}. \end{aligned} \tag{10.4}$$

其中 $\mathcal{U} = \{\boldsymbol{\xi} : \|\boldsymbol{\xi}\|_2 \leq 1/\sqrt{2}\}$。此问题的 YALMIP 的程序如下：

```
sdpvar x xi(2,1)
F = [x + sum(xi) <= 1];
W = [norm(xi) <= 1/sqrt(2), uncertain(xi)];
objective = -x;
sol = optimize(F + W,objective)
```

10.3 鲁棒模型在 RSOME 上的实现

RSOME全称 Robust Stochastic Optimization Modeling Environment，是一个针对 MATLAB 的优化建模工具包，可以在 Windows，macOS，以及 Linux 操作系统下运行。该软件工具包为用户提供了丰富便捷的语法环境，可以直观高效地处理各类优化模型，如经典的随机规划和鲁棒优化模型，以及分布鲁棒优化和鲁棒随机优化模型等。目前版本的 RSOME 提供了调用商用求解器，如 Gurobi，MOSEK，以及 CPLEX 的接口。用户可以根据自己的需求选择求解器对模型进行求解。更多详情和下载资源请参考官网（https://www.rsomerso.com/）。

10.3.1 RSOME 建模求解的基本步骤

RSOME 工具包提供了灵活的函数和语法环境来定义各类优化模型。具体的建模和求解步骤如下：

1. 用 rsome 函数创建一个优化模型对象
2. 为优化模型创建随机变量和决策变量
3. 创建和定义模糊集合
4. 定义动态决策的近似表达式

5. 定义目标函数和约束条件
6. 求解模型和返回最优解

我们首先用一个简单的凸优化问题来演示使用 RSOME 工具包建模求解的基本步骤。

示例 10.4: Bertsimas and Sim (2004) 介绍了一个经典的投资组合优化（Portfolio optimization）问题：

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i - \phi \sum_{i=1}^{n} \sigma_i^2 x_i^2 \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \forall i = 1, 2, ..., n \end{aligned} \tag{10.5}$$

这个模型考虑了 $n = 150$ 支股票。对第 i 支股票，回报率的期望是 p_i，标准差是 σ_i。我们用 x_i 定义决策变量代表每支股票的投资比例，并用参数 $\phi = 5$ 来平衡投资的回报期望和风险。这个投资组合优化问题可以写成下面的代码。

```
%% 定义模型参数
n = 150;                                    % 股票支数
p = 1.15 + 0.05/150*(1:n)';                 % 回报率期望
sigma = 0.05/450*sqrt(2*n*(n+1)*(1:n)');    % 回报率标准差
phi = 5;                                    % 平衡参数

%% 创建模型和决策变量
model = rsome('投资组合优化');               % 创建一个模型model
x = model.decision(n);                      % 定义决策变量x

%% 定义目标函数和约束条件
model.max(p'*x - phi*sumsqr(sigma.*x));     % 最大化目标函数
model.append(sum(x) == 1);                  % 定义约束条件
model.append(x >= 0);                       % 定义变量边界

%% 求解模型和返回最优解
model.solve;                                % 求解模型
obj = model.get;                            % 最优目标值
X = x.get                                   % x的最优解
```

这个投资组合优化模型被转化成一个二阶锥规划问题并由内置的 CPLEX 求解器求解，得到的最优值是 1.185。在建模和求解过程中，我们提醒读者注意：

1. 就像 MATLAB 中的其他程序应用，RSOME 工具包在处理向量和矩阵时的效率会远远高于循环。因此在这个例子中，我们把决策变量 x_i 创建成一个向量 $\boldsymbol{x}$，并

用矩阵运算来定义目标函数和约束条件。

2. MATLAB 中的绝大部分矩阵运算符号和法则也适用于 RSOME 模型的表达式，比如这个例子中的矩阵乘法、点乘、以及加减法等。
3. RSOME 工具包提供了一系列凸函数，比如绝对值 `abs`，范数 `norm`，和这个示例中的平方和 `sumsqr` 等。在模型中使用凸函数时，必须保证该模型的可行域为凸集，否则程序会出现错误信息。

10.3.2 用 RSOME 定义事件式分布模糊集合

RSOME 利用在5.5.2中介绍的事件式分布模糊集合来表达随机变量的概率分布情况。和之前介绍的软件工具相比，这个事件式分布模糊集合更加的通用，因此我们可以使用 RSOME 方便地建立各类鲁棒优化和随机规划的模型，比如下面的例子。

示例 10.5: Bertsimas and Sim (2004) 介绍的投资组合优化（Portfolio optimization）问题可以写成一个经典鲁棒优化模型：

$$\begin{aligned} \max \quad & \min_{\xi\in\mathcal{U}} \sum_{i=1}^{n}(p_i - \sigma_i\xi_i)x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \forall i = 1, 2, ..., n \end{aligned} \tag{10.6}$$

在该模型中，随机变量 ξ 代表实际股票回报率和期望值间的偏差。这个随机被约束在一个预算不确定集（budget uncertainty set）$\mathcal{U}$ 中，其表达式如下：

$$\mathcal{U} = \{\xi | \|\xi\|_\infty \leq 1, \|\xi\|_1 \leq \Gamma\} \tag{10.7}$$

其中 Γ 代表不确定性预算（budget of uncertainty）。以上的预算不确定集可以用事件式分布模糊集合中的支撑集合来表示，这个经典鲁棒优化模型因而由以下代码建立。

```
%% 定义模型参数
n = 150;                                    % 股票支数
p = 1.15 + 0.05/150*(1:n)';                 % 回报率期望
sigma = 0.05/450*sqrt(2*n*(n+1)*(1:n)');    % 回报率标准差
gamma = 5;                                  % 不确定性预算

%% 创建模型，随机变量和决策变量
model = rsome('投资组合鲁棒优化');           % 创建一个模型model
xi = model.random(n);                       % 定义随机变量xi
x = model.decision(n);                      % 定义决策变量x
```

```
%% 创建和定义分布模糊集合
P = model.ambiguity;                          % 创建分布模糊集合
P.suppset(norm(xi, Inf) <= 1, ...
          norm(xi, 1) <= Gamma);              % 定义支撑集信息
model.with(P);                                % 传递模糊信息到模型

%% 定义目标函数和约束条件
model.max((p + sigma.*xi)' * x);              % 最大化目标函数
model.append(sum(x) == 1);                    % 定义约束条件
model.append(x >= 0);                         % 定义变量边界

%% 求解模型
model.solve;                                  % 求解模型
```

在这个例子中，我们在定义一个模型 `model` 和其对应的随机变量 `xi` 和决策变量 `x` 之后，使用 `ambiguity` 方法函数创建一个模糊集对象。该模糊集通过 `suppset` 方法函数定义了随机变量 ξ 的多面体（polyhedral）支撑集，等价于这个鲁棒优化模型的不确定集 U (10.7). 最后我们用 `with` 方法函数将模糊集信息传递到模型本身，并在定义模型的目标函数以及约束条件之后，即可调用 solve 函数求解。在求解过程中，RSOME 将模型自动转化为一个线性规划标准型，并通过求解器得到最后的最优解。

示例 10.6: Chen et al. (2019) 介绍的报童问题可以写成以下分布鲁棒优化模型：

$$\begin{array}{ll} \max & (p-c)x - \sup\limits_{\mathbb{P}\in\mathcal{F}} \mathbb{E}_{\mathbb{P}}\left[\max\left\{p(x-z),0\right\}\right] \\ \text{s.t.} & x \geq 0 \end{array} \tag{10.8}$$

其中参数 p 和 c 分别是产品的销售价格和预定价格，决策变量 x 代表商品的预定量，随机变量 z 代表商品的需求。随机变量 z 的分布由一个基于 Wasserstein 距离的分布模糊集（Wasserstein ambiguity set）刻画。Chen et al. (2019) 证明了这个模糊集可以写成:

$$\mathcal{F} = \left\{ \mathbb{P} \in \mathcal{P}_0\left(\mathbb{R}\times[S]\right) \;\middle|\; \begin{array}{ll} (\tilde{z},\tilde{s}) \sim \mathbb{P} & \\ \mathbb{E}_{\mathbb{P}}\left[\rho(\tilde{z},\hat{z}_{\tilde{s}}) \mid \tilde{s}\in[S]\right] \leq \theta & \\ \mathbb{P}\left[\tilde{z}\in[0,\bar{Z}] \mid \tilde{s}=s\right] = 1 & \forall s\in[S] \\ \mathbb{P}\left[\tilde{s}=s\right] = 1/S & \forall s\in[S] \end{array} \right\}$$

其中 $\bar{Z}$ 是商品需求的上限，而 $\hat{z}_s$, $s = 1,2,3,...,S$ 代表随机变量 z 的历史经验数据（empirical data）。这个模糊集定义了所有和经验数据的 Wasserstein 距离不大于常数 θ

的所有分布函数，并且可以写成下面的事件式分布模糊集：

$$\mathcal{G}=\left\{\mathbb{P}\in\mathcal{P}_0\left(\mathbb{R}^2\times[S]\right)\;\middle|\;\begin{array}{ll}((\tilde{z},\tilde{u}),\tilde{s})\sim\mathbb{P} & \\ \mathbb{E}_{\mathbb{P}}\left[\tilde{u}\mid\tilde{s}\in[S]\right]\leq\theta & \\ \mathbb{P}\left[\tilde{z}\in[0,\bar{Z}],\ \rho(\tilde{z},\hat{z}_{\tilde{s}})\leq\tilde{u}\mid\tilde{s}=s\right]=1 & \forall s\in[S]\\ \mathbb{P}\left[\tilde{s}=s\right]=1/S & \forall s\in[S]\end{array}\right\}$$

基于上面的事件式分布模糊集，这个报童问题可以写成下面的 RSOME 程序。

```
%% 定义模型参数
Zbar = 100;                                % 商品需求上限
S=500;                                     % 经验数据的数目
Zhat = Zbar * rand(1, S);                  % 生成经验数据
p = 1.5;                                   % 销售价格
c = 1.0;                                   % 预定成本
theta = Zbar * 0.01;                       % Wasserstein 距离

%% 创建模型，随机变量和决策变量
model = rsome('报童问题');                  % 创建一个模型model
z = model.random;                          % 定义随机变量z
u = model.random;                          % 定义辅助随机变量u
x = model.decision;                        % 定义决策变量x

%% 创建和定义分布模糊集合
P = model.ambiguity(S);                    % 创建S个情景的分布模糊集
for s = 1:S
    P(s).suppset(0 <= z, z <= Zbar, ...
                 norm(z-Zhat(s)) <= u); % 定义每个情景s的支撑集
end
P.exptset(expect(u) <= theta)              % 定义整体的期望值信息
pr = P.prob;                               % 每个情景s的概率
P.probset(pr == 1/S);                      % 定义每个情景概率的值
model.with(P);                             % 传递模糊信息到模型

%% 定义目标函数和约束条件
model.max((p-c)*x - ...
          expect(maxfun({p*(x-z), 0})));% 最大化目标函数
model.append(x >= 0);                      % 定义变量边界
```

```
%% 求解模型
model.solve;                                    % 求解模型
```

请注意以上程序中的函数 `expect()` 代表了表达式的最坏期望。这个例子体现了RSOME工具包的优势：我们可以直观便捷地定义分布模糊集的各类概率分布信息，比如：

1. 我们用 `P(s).suppset()` 来分别定义每一个情景 `s` 下的支撑集；
2. 函数 `P.exptset()` 定义随机变量期望值的集合表达式；
3. 模糊集 `P` 的属性 `P.prob` 返回一个包含每个情景 `s` 概率的列向量，并且情景概率的集合可以通过 `P.probset()` 来定义。

由此可见，RSOME 代码有着更好的可读性和通用性。用户可以轻松地将模糊集合的数学表达式用程序表达出来，因此能够更快更方便地求解和调试各类鲁棒优化模型。

10.3.3 用 RSOME 定义事件式近似法则

在前文5.1中，我们提到在多阶段问题（Multi-stage problems）中，一些决策变量可以根据未来观测到的一部分随机变量的值而动态变化。这类“等待决策（wait-and-see decisions)”可能给建模和求解带来巨大的困难。因此在5.5中我们介绍了两种近似处理的方法：一是事件式静态近似法则，也就是在不同的事件下，决策取不同的常数值；另一种称为事件式线性近似法则，在该法则中，决策在不同的事件下为随机变量的不同仿射函数。在实际应用中，我们可以使用 RSOME 方便灵活地定义这两类近似法则。在定义各类近似法则中，用户可以设定：

1. 一个相互独立又完全穷尽（MECE）的事件集合，使得被定义的决策在不同的事件下动态变化。这样的事件适应（event-wise adaptation) 集合可以通过函数 `evtadapt()` 来实现。
2. 被定义的决策对随机变量的仿射适应（affine adaptation）表达式。这个仿射适应表达式可以通过函数 `affadapt()` 来设定。

接下来，我们用一个实际的例子来演示如何使用 RSOME 处理此类问题。

示例 10.7:在之前的示例10.6中，表达式 $\max\{p(x-z),0\}$ 可以用动态决策近似函数 $y(\tilde{s},(\tilde{z},\tilde{u}))$ 替代，由此之前的问题可以写成一个两阶段（two-stage）模型如下：

$$\begin{array}{ll}\max & (p-c)x-\sup\limits_{\mathbb{P}\in\mathcal{F}}\mathbb{E}_{\mathbb{P}}\left[y(\tilde{s},(\tilde{z},\tilde{u}))\right]\\ \text{s.t.} & x\geq 0\\ & y(\tilde{s},(\tilde{z},\tilde{u}))\geq p(x-z)\\ & y(\tilde{s},(\tilde{z},\tilde{u}))\geq 0\\ & y\in\bar{\mathcal{A}}([S],2)\end{array}\tag{10.9}$$

其中 $y\in\bar{\mathcal{A}}([S],2)$ 意味着动态决策 y 在每一个情景 $s\in[S]$ 下，都是不同的关于随机变量 z 和 u 的仿射函数。这样的一个两阶段模型可以用如下程序来表达。

```
%% 定义模型参数
Zbar = 100;                                % 商品需求上限
S=500;                                     % 经验数据的数目
Zhat = Zbar * rand(1, S);                  % 生成经验数据
p = 1.5;                                   % 销售价格
c = 1.0;                                   % 预定成本
theta = Zbar * 0.01;                       % Wasserstein 距离

%% 创建模型，随机变量和决策变量
model = rsome('报童问题');                  % 创建一个模型model
z = model.random;                          % 定义随机变量z
u = model.random;                          % 定义辅助随机变量u
x = model.decision;                        % 定义决策变量x
y = model.decision;                        % 定义动态决策变量y

%% 创建和定义分布模糊集合
P = model.ambiguity(S);                    % 创建S个情景的分布模糊集
for s = 1:S
    P(s).suppset(0 <= z, z <= Zbar, ...
                 norm(z-Zhat(s)) <= u); % 定义每个情景s的支撑集
end
P.exptset(expect(u) <= theta)              % 定义整体的期望值信息
pr = P.prob;                               % 每个情景s的概率
P.probset(pr == 1/S);                      % 定义每个情景概率的值
model.with(P);                             % 传递模糊信息到模型

%% 定义动态决策的事件适应和仿射适应表达式
for s = 1:S
    y.evtadapt(s);                         % y随不同的情景s变化
end
y.affadapt(z);                             % y线性依赖随机变量z
y.affadapt(u);                             % y线性依赖随机变量u

%% 定义目标函数和约束条件
model.max((p-c)*x - expect(y));            % 最大化目标函数
model.append(y >= 0);                      % 定义约束条件
```

```
model.append(y >= p * (w-z));          % 定义约束条件
model.append(x >= 0);                  % 定义变量边界

%% 求解模型
model.solve;                           % 求解模型
```

这一节主要介绍了 RSOME 优化建模工具包的基本概念和使用方法。从以上的示例中，我们可以看出，RSOME 提供了一个简洁友好的建模环境，让用户可以通过近似数学表达式的程序语言定义复杂的模糊集合动态决策表达式，极大地提高了用户建立和求解各类鲁棒和随机优化模型的效率。关于 RSOME 工具包的更多细节可以参考https://www.rsomerso.com/。另 RSOME 已开发 Python 版本，且其语法与 Matlab 版本略有不同。有关于Python 版本的更多细节可以参考https://xiongpengnus.github.io/rsome/。

本章参考文献

Bertsimas, Dimitris and Melvyn Sim, "The price of robustness," *Operations Research*, 2004, *52* (1), 35–53.

Chen, Zhi, Melvyn Sim, and Peng Xiong, "Robust Stochastic Optimization: The Synergy of Robust Optimization and Stochastic Programming.," 2019, p. *Available at Optimization Online*.

Dunning, Iain, Joey Huchette, and Miles Lubin, "JuMP: A modeling language for mathematical optimization," *SIAM Review*, 2017, *59* (2), 295–320.