基于 C&CG 算法的两阶段鲁棒优化

Matlab 复现

文献来源：B. Zeng, L. Zhao. *Solving two-stage robust optimization problems using a column-and-constraint generation method*. Operations Research Letters 41 (2013) 457–461.

参考解读：鲁棒优化| C&CG 算法求解两阶段鲁棒优化：全网最完整、最详细的【入门-完整推导-代码实现】笔记

一、 两阶段鲁棒优化（Two-stage Robust Optimization）

【两阶段鲁棒优化】

在传统的确定性优化模型中，我们通常假设所有参数在决策前都是已知的，例如需求、负荷、风光出力等。然而在实际系统中，这些参数往往存在显著不确定性。**鲁棒优化（Robust Optimization, RO）**通过在一个不确定集内寻找**“最坏情形”**的最优解，从而提高方案的抗扰性。但单阶段鲁棒优化把所有决策一次性做完，往往过于保守。为此，引入了**两阶段鲁棒优化（Two-stage RO）**框架：

- **第一阶段：** 在不确定性揭晓前做出的决策。在电力中对应机组组合（Unit Commitment, UC），即决定哪些发电机开机。
- **第二阶段：** 在不确定性（如风电出力实际值）揭晓后做出的调整决策。在电力中对应经济调度（Economic Dispatch, ED），即调整发电机的具体出力。

【两阶段鲁棒模型的典型形式】

$$\min_{y} c^{\mathrm{T}} y + \max_{u\in\mathcal{U}} \min_{x\in F(y,u)} b^{\mathrm{T}} x \tag{1-1}$$

决策变量：y为第一阶段决策（如容量、选址），x为第二阶段决策（如潮流、运输流）

参数不确定性：u为不确定参数（如负荷、需求、新能源等），$\mathcal{U}$为有限的不确定集。

【参考论文所完成的工作】

Zeng & Zhao (2013)的工作，选取**鲁棒选址–运输（Location–Transportation）问题**作为算例，采用**列与约束生成算法（Column-and-Constraint Generation, C&CG）**求解两阶段鲁棒模型，本文用 Matlab+YALMIP+Gurobi 复现。

二、 数学模型

1. 通用线性两阶段鲁棒优化模型 Two-Stage RO

【目标函数】

最小化“第一阶段投资成本”$c^{\mathrm{T}}y$和“最恶劣场景下第二阶段运输成本”$b^{\mathrm{T}}x$之和。

$$\min_{y} c^{\mathrm{T}}y + \max_{u\in\mathcal{U}} \min_{x\in F(y,u)} b^{\mathrm{T}}x \tag{2-1-1}$$

【约束方程】

$$Ay \geq d,\ y \in \mathcal{S}_y \tag{2-1-2}$$

$$F(y,u) \coloneqq \{x \in \mathcal{S}_x : Gx \geq h - Ey - Mu\} \tag{2-1-3}$$

其中，约束（2-2）是第一阶段的“先天约束”（如预算、二进制变量等）；约束（2-3）是第二阶段在给定y,u下x的可行性约束（功率平衡、容量限制等）。

2. 选址–运输 Two-Stage RO

【目标函数】

$$\min_{y,z} \sum_i f_i y_i + \sum_i a_i z_i + \sum_i \sum_j c_{ij} x_{ij} \tag{2-2-1}$$

f_i为固定建设成本（fixed cost），a_i为单位容量建设成本（unit capacity cost），c_{ij}为单位运输成本。

第一阶段变量：$y_i \in \{0,1\}$，表示是否在节点i建设设施（facility location variable），z_i表示在节点i的设施容量（the capacity variable）。

第二阶段变量：x_{ij}表示从i到j的运输量（transportation variable），x是一个（源节点数，目的节点数）大小的矩阵变量。

不确定参数：d_j表示客户j的需求，D为多面体不确定集。

该问题决策分为两个阶段：

第一阶段：在需求未知时，决定设施选址和容量。

第二阶段：在需求不确定集揭示后，决定运输方案以满足需求并最小化成本。

【约束方程】

容量建设约束 $z_i \leq K_i y_i, \forall i$ （2-2-2）

其中K_i为节点i的最大允许建设容量。

容量限制（流出量不超过容量） $\sum_j x_{ij} \leq z_i, \forall i$ （2-2-3）

需求限制（流出量满足需求） $\sum_i x_{ij} \geq d_j, \forall j$ （2-2-4）

运输量为正 $$x_{ij} \geq 0 \quad (2\text{-}2\text{-}5)$$

【不确定集D】

论文采用如下方式构建不确定集：

$$d_j = \underline{d}_j + \hat{d}_j g_j \quad (2\text{-}2\text{-}6)$$

$$\text{s.t. } g_j \in \{0,1\}, j = 1,\ldots,n \quad (2\text{-}2\text{-}7)$$

$$\sum_j g_j \leq \Gamma, j = 1,\ldots,n \quad (2\text{-}2\text{-}8)$$

其中，d_j为实际需求，$\underline{d}_j$为基准需求，$\hat{d}_j$为最大需求偏差，g_j为不确定系数（Perturbation Factor），表示偏差发生的程度，Γ为“不确定性预算”参数。

3. Two-Stage RO 和 Column-and-Constraint Generation Algorithm

与 Benders 分解法在主问题中不断增加对偶割平面的思想不同，Column-and-Constraint Generation（C&CG）算法核心思想是在主问题中不断引入最坏场景的“新变量（列）”和“新约束”。

【主问题（MP）】

C&CG 的主问题只考虑已发现的最坏场景集$\mathcal{O} = \{u_1, u_2, \ldots, u_k\}$合，$k$为迭代次数。

在第k次迭代时，MP 如下：

MP_k $$\min_{y,\eta,x^1,\ldots,x^l} c^\text{T} y + \eta \quad (2\text{-}3\text{-}1)$$

$$\text{s.t. } Ay \geq d, \quad (2\text{-}3\text{-}2)$$

$$\eta \geq b^\text{T} x^l, l = 1,\ldots,k \quad (2\text{-}3\text{-}3)$$

$$Gx^l \geq h - Ey - Mu^l, l = 1,\ldots,k \quad (2\text{-}3\text{-}4)$$

$$y \in \mathcal{S}_y, x^l \in \mathcal{S}_x \quad (2\text{-}3\text{-}5)$$

主问题的决策变量为$y, \eta, x^1, \ldots, x^l$，其中，对于每一个场景$u^l$，都对应引入一组决策变量$x^l$。

【子问题（SP）】

C&CG 的子问题与 Benders 方法一致：在给定第一阶段解$\bar{y}$下，寻找使第二阶段成本$b^\text{T} x$最大的场景u：

$$\max_{\lambda,u} (h - E\bar{y} - Mu)^\text{T} \lambda \quad (2\text{-}3\text{-}6)$$

$$\text{s.t. } G^\text{T} \lambda \leq b, \quad (2\text{-}3\text{-}7)$$

$$\lambda \geq 0, \quad (2\text{-}3\text{-}8)$$

$$u \in \mathcal{U}. \tag{2-3-9}$$

写成函数形式：

$$\text{SP} \qquad \mathcal{Q}(y) = \max_{\lambda,u} \{(h - E\bar{y} - Mu)^{\mathrm{T}}\lambda : G^{\mathrm{T}}\lambda \le b, \lambda \ge 0, u \in \mathcal{U}\} \tag{2-3-10}$$

其中，$\bar{y}$为固定参数，决策变量是不确定量u和对偶变量λ。

如果满足强对偶性，则求解该单层问题得到的最优值即为原子问题的最优值，否则，由弱对偶性，该单层问题的最优值提供了一个下界。【参见9602PPTDuality①】

【求解过程】

Step 1: 初始化

- 设定下界$LB = -\infty$，上界$UB = +\infty$。
- 设定初始迭代次数$k = 0$，场景集合$\mathcal{O} = \emptyset$（或者包含一个标称场景）。

Step 2: 求解主问题 (MP)

- 求解包含当前场景集合$\mathcal{O}$的主问题。
- 得到最优解：第一阶段决策y_{k+1}^*，以及预估的最坏成本η_{k+1}^*。
- **更新下界：**$LB = c^{\mathrm{T}}y_{k+1}^* + \eta_{k+1}^*$。

※因为主问题只考虑了有限个场景，是对原问题的松弛，所以它的最优值是原问题的下界。

Step 3: 求解子问题 (SP)

- 固定$y = y_{k+1}^*$，求解子问题，找到使成本最大的最坏场景u^*。
- 得到子问题的真实成本$\mathcal{Q}(y_{k+1}^*)$。
- **更新上界：**$UB = \min\{UB, c^{\mathrm{T}}y_{k+1}^* + \mathcal{Q}(y_{k+1}^*)\}$

※这是一个可行解在最坏情况下的真实成本，所以是原问题的上界。

Step 4: 收敛性检查

- 计算 Gap：$(UB - LB)/|LB|$。
- 如果 Gap$\le \epsilon$，停止迭代，输出y^*。

Step 5: 列与约束生成 (核心步骤)

- 如果未收敛，将找到的最坏场景u^*加入集合$\mathcal{O}$。
- **创建新变量：** 在主问题中定义一组新变量x^{k+1}。
- **添加新约束：** 将对应于u^*的物理约束$Gx^{k+1} \ge h - Ey - Mu^*$添加到主问题中。

- 令$k=k+1$，返回 Step 2。

4. 选址–运输 Two-Stage RO 和 C&CG 算法

【主问题 MP】

$$\min_{y,z,\eta,x^1,\ldots,x^l} [400\ 414\ 326]\begin{bmatrix}y_0\\y_1\\y_2\end{bmatrix}+[18\ 25\ 20]\begin{bmatrix}z_0\\z_1\\z_2\end{bmatrix}+\eta \quad (2\text{-}4\text{-}1)$$

总容量约束 $\text{s.t. } z_i \le K_i y_i, \forall i$ (2-4-2)

第二阶段成本约束 $\eta \ge \sum_i \sum_j c_{ij}x_{ij}$ (2-4-3)

需求限制（流出量满足需求） $\sum_i x_{ij} \ge d_j, \forall j$ (2-4-4)

容量限制（流出量不超过容量） $\text{s.t. } \sum_j x_{ij} \le z_i, \forall i$ (2-4-5)

0-1选址 $y \in \{0,1\}$ (2-4-6)

容量为非负 $z \ge 0$ (2-4-7)

运输量非负 $x \ge 0$ (2-4-8)

※主问题直接利用了网络流结构，求解器（如CPLEX/Gurobi）可以利用这种结构加速求解，通过增加变量来换取更强的下界。

【子问题对偶问题 SPDP】

同 Benders 方法，得到子问题的对偶问题：

$$\max_{\pi,\lambda} \sum_j \left(\underline{d}_j + \widehat{d}_j g_j\right)\lambda_j - \sum_i z_i \pi_i \quad (2\text{-}4\text{-}9)$$

对偶可行性约束 $\text{s.t. } \lambda_j - \pi_i \le c_{ij}, \forall (i,j)$ (2-4-10)

不确定集 $0 \le g_j \le 1, \forall j$ (2-4-11)

不确定集 $\sum_j g_j \le \Gamma, \forall j$ (2-4-12)

对偶变量约束 $\pi_i \ge 0,$ (2-4-13)

对偶变量约束 $\lambda_j \ge 0.$ (2-4-14)

三、 编写代码

【参数定义】

```
%% 1. 参数定义 (Parameter Definition)
num_facilities = 3; %3 个源点
num_customers = 3; %3 个终点
% 成本参数
f = [400; 414; 326];    % 固定建设成本 Fixed cost
a = [18; 25; 20];       % 单位容量建设成本 Unit capacity cost
C_trans = [             % 单位运输成本 Transportation cost
```

```
    22, 33, 24;
    33, 23, 30;
    20, 25, 27
];
K = [800; 800; 800];% 最大允许建设容量

% 不确定集参数
d_bar = [206; 274; 220]; % 基准需求
d_hat = [40; 40; 40]; % 最大需求偏差
Gamma = 1.8;

%  惩罚系数 (Penalty Cost): 代表切负荷的高昂成本。
PENALTY_COST = 10000;
```

由于后文将讨论在子问题中加入对偶变量上界约束，因此设置惩罚系数。

【主问题：初始化】

```
%% 2. 初始化
LB = -inf;
UB = inf;
epsilon = 1e-3;
Max_Iter = 30;

% 用于存储识别出的"最坏场景" (Worst-case Scenarios)
% CCG 的核心：保存每一次子问题找到的 d 值
Scenario_Set = {};

% 初始给一个基准场景 (Nominal Scenario) 启动算法
Scenario_Set{1} = d_bar;

fprintf('Starting CCG Algorithm (Replication of Table 1)...\n');
fprintf('%-5s | %-12s | %-12s | %-10s\n', 'Iter', 'LB', 'UB', 'Gap');
```

【开始迭代】

```
for iter = 1:Max_Iter
```

【主问题：定义决策变量】

```
    % 3. 求解主问题 (Master Problem, MP)
    % 第一阶段变量
    y = binvar(num_facilities, 1); % 选址
    z = sdpvar(num_facilities, 1); % 容量
```

```
    eta = sdpvar(1, 1);              % 辅助变量（代表第二阶段的最坏成本）
```

【主问题：目标函数】

```
    % 目标函数
    Obj_MP = f'*y + a'*z + eta;
```

【主问题：约束】

```
    % 主问题约束
    Constraints_MP = [];
    % 容量建设约束
    Constraints_MP = [Constraints_MP, z <= K .* y];
    % 设施容量非负
    Constraints_MP = [Constraints_MP, z >= 0];
    % [附录 A-4.3] 添加总容量下限约束
    % 这能避免第一轮 z=0 导致的无界，加速收敛
    Constraints_MP = [Constraints_MP, sum(z) >= Max_Total_Demand_Est];

    % --- CCG 核心：为每个已知场景 l 添加一套约束 ---
    for l = 1:length(Scenario_Set)
        d_current = Scenario_Set{l}; % 取出第 l 个场景的需求数值

        % 创建针对场景 l 的第二阶段变量 x^(l) (num_facilities x
num_customers)
        % 变量命名技巧：在 YALMIP 中由于循环变长，建议临时定义，放入约束
        x_l = sdpvar(num_facilities, num_customers, 'full');
        % 需求限制
        Constraints_MP = [Constraints_MP, sum(x_l, 1)' >= d_current];
        % 容量限制
        Constraints_MP = [Constraints_MP, sum(x_l, 2) <= z];
        % 运输量非负
        Constraints_MP = [Constraints_MP, x_l >= 0];
        % 第二阶段成本约束（Epigraph 约束）eta 必须大于所有已知场景的成本
        Cost_Scenario = sum(sum(C_trans .* x_l));
        Constraints_MP = [Constraints_MP, eta >= Cost_Scenario(:)];
    end
```

【主问题：求解】

```
    % 求解 MP
    ops = sdpsettings('solver', 'gurobi', 'verbose', 0);
    sol_MP = optimize(Constraints_MP, Obj_MP, ops);
```

```
    if sol_MP.problem ~= 0
        error('Master Problem Failed! Info: %s', sol_MP.info);
    end
```

【主问题：得到求解结果，更新主问题下界】

```
    % 更新下界 LB
    % LB = c*y* + eta*
    y_star = value(y);
    z_star = value(z);
    eta_star = value(eta);
    Current_LB = value(Obj_MP);
    LB = Current_LB; % CCG 的 MP 随着约束增加，LB 单调递增
```

【子问题：初始化】

```
    %% 4. 求解子问题 (Subproblem)

    % 对偶变量和不确定性变量
    pi_sp = sdpvar(num_facilities, 1);      % 对应容量约束 (dual variable)
    lambda_sp = sdpvar(num_customers, 1);   % 对应需求约束 (dual variable)
    g = sdpvar(num_customers, 1);           % 不确定性变量
    d_real = d_bar + d_hat .* g;            % 实际需求表达式
```

【子问题：目标函数】

```
    % 子问题对偶问题的目标函数
    Obj_SPDP=sum(d.*lambda_sp)-sum(z_star.*pi_sp);
```

【子问题：约束】

```
    % 子问题约束
    Constraints_SPDP = [];

    % 对偶约束 (Dual Feasibility): lambda_j - pi_i <= c_ij
    for i = 1:num_facilities
        for j = 1:num_customers
            Constraints_SPDP = [Constraints_SPDP, lambda_sp(j) - pi_sp(i) <=
C_trans(i,j)];
        end
    end

    % 对偶变量非负 & 有界 (防止 Unbounded)
    Constraints_SPDP = [Constraints_SPDP, 0 <= pi_sp <= PENALTY_COST];
    Constraints_SPDP = [Constraints_SPDP, 0 <= lambda_sp <= PENALTY_COST];

    % 不确定集约束 (Uncertainty Set) [cite: 351]
```

```
Constraints_SPDP = [Constraints_SPDP, 0 <= g <= 1];
Constraints_SPDP = [Constraints_SPDP, sum(g) <= Gamma];
Constraints_SPDP = [Constraints_SPDP, g(1) + g(2) <= 1.2];
```

【子问题：求解】

```
% 求解子问题
ops_sp = sdpsettings('solver', 'gurobi', 'verbose', 0);
ops_sp.gurobi.NonConvex = 2; % 开启非凸支持

sol_SP = optimize(Constraints_SPDP, -Obj_SPDP, ops_sp);

if sol_SP.problem ~= 0
    error('Subproblem Failed! Info: %s', sol_SP.info);
end
```

【子问题：得到求解结果】

```
% 提取最坏场景
d_worst = value(d_real);
Actual_Subproblem_Cost = value(Obj_SPDP);
```

【更新主问题上界】

```
% 更新上界 UB
% UB = min(UB, FirstStageCost + MaxRecourseCost)
Current_UB = f'*y_star + a'*z_star + Actual_Subproblem_Cost;
UB = min(UB, Current_UB);
```

【检查主问题上下界是否收敛，若收敛，求解成功】

```
% 计算 Gap
Gap = abs(UB - LB) / (abs(LB) + 1e-5);

fprintf('%-5d | %-12.2f | %-12.2f | %-10.4f\n', iter, LB, UB, Gap);

%% 5. 收敛检查与列生成
if Gap <= epsilon
    fprintf('\nCCG Converged!\n');
    break;
```

【若不收敛，添加新列，继续迭代】

```
else
    % --- CCG 核心步骤：添加新列 (Column Generation) ---
    % 将找到的最坏需求 d_worst 加入场景集合
```

```
        % 下一次 MP 会自动为这个新场景创建对应的变量 x 和约束
        Scenario_Set{end+1} = d_worst;
    end
end
```

【求解结果】

```
Starting CCG Algorithm (Replication of Table 1)...
Iter  | LB          | UB          | Gap
1     | 31832.00    | 33680.00    | 0.0581
2     | 33680.00    | 33680.00    | 0.0000
CCG Converged!
=== Final Results (Comparison with Table 1) ===
Facilities Built (y): [1 0 1]
Capacities Built (z): [252 0 520]
Optimal Cost: 33680.00
Total Iterations: 2
```

相较于论文结果（Table 1），本文 C&CG 最终收敛值一致，完成 C&CG 方法的复现