中国科学: 数学 2021 年 第 51 卷 第 12 期: 2065～2086
SCIENTIA SINICA Mathematica
论 文

计算最小体积覆盖椭球的坐标轴下降算法

陶杰[1], 张威[2*], 卢超[3], Mark Goh[4]

1. 上海理工大学管理学院, 上海 200093;
2. Center of Excellence in Modelling and Simulation for Next Generation Ports, National University of Singapore, Singapore 117602, Singapore;
3. 上海大学管理学院, 上海 200444;
4. Business School, National University of Singapore, Singapore 117592, Singapore
E-mail: taojie@usst.edu.cn, lindelfeel@gmail.com, 06luchao@163.com, bizgohkh@nus.edu.sg

收稿日期: 2018-11-07; 接受日期: 2019-11-01; 网络出版日期: 2021-03-02; * 通信作者
国家自然科学基金 (批准号: 71601117 和 71704101)、上海市软科学项目 (批准号: 19692104600) 和教育部人文社科项目 (批准号: 17YJC630094) 资助项目

摘要 最小体积覆盖椭球问题是一个基本的凸优化问题. 本文给出最小体积覆盖椭球问题的新性质—对算法依坐标轴光滑性, 据此提出一种坐标轴下降算法来计算最小体积覆盖椭球并证明该算法的收敛速度是全局次线性收敛且局部线性收敛的. 从计算时间角度来看, 该算法优于经典的 Frank-Wolfe 算法, 并且这种优势对于高维数据集尤为明显. 更进一步, 我们发现该算法在计算最小体积覆盖椭球问题方面比随机坐标轴下降算法更有优势. 最后通过大规模数值算例测试来验证我们得到的理论结果.

关键词 最小体积覆盖椭球 一阶导数算法 坐标轴下降

MSC (2020) 主题分类 90C30

1 引言

设 $\mathcal{X}=\{x_1,\ldots,x_m\}$ 是 n 维 Euclid 空间中的点集. $\mathcal{X}$ 的最小体积覆盖椭球 (minimum volume enclosing ellipsoid, MVEE) 是所有包含点集 $\mathcal{X}$ 的椭球中体积最小者, 它在很多领域有重要的应用 (参见文献 [1–6]). John [7] 是最早提出 MVEE 概念的学者, 他给出了 MVEE 的存在性条件, 并指出 $\mathbb{R}^n$ 空间中任意一个椭球最多可由 $\frac{n(n+1)}{2}$ 个点确定. 这一论断对后来的核心集 (core set) 理论的发展起到关键性作用 [7,8]. 计算 MVEE 的方法大致可以分为两类: Newton 型方法和条件梯度型方法 (即 Frank-Wolfe (FW) 型算法). 对于 Newton 型方法, Nesterov 和 Nemirovskii [9] 提出了内点算法用以计算 MVEE 的 $(1+\epsilon)$- 近似椭球, 即点集的另一个覆盖椭球, 其体积与 MVEE 的体积之比介于 1 和 $1+\epsilon$ 之间, 其中 ϵ 是事先设定好的误差界. 该内点算法的计算复杂度为 $O(m^{3.5}\ln(\frac{Rm}{\epsilon r}))$, 其中 $0\leqslant r\leqslant R$ 是两个 Euclid 球 B_r 和 B_R 的半径, 满足 $B_r\subseteq \mathrm{conv.hull}(\mathcal{X})\subseteq B_R$. Vandenberghe 等 [10] 利用内点法

英文引用格式: Tao J, Zhang W, Lu C, et al. A coordinate gradient descent algorithm for computing the minimum volume enclosing ellipsoid (in Chinese). Sci Sin Math, 2021, 51: 2065–2086, doi: 10.1360/SCM-2018-0795

© 2021《中国科学》杂志社

求解一个最大行列式问题, 该方法也可以被用于求解 MVEE 问题. Sun 和 Freund [11] 提出了对偶削减 Newton 法用以计算 MVEE, 该算法对于中等规模的样本十分有效. 对于大规模样本, 他们提出了一种积极集策略, 可以提升计算效率. 然而, 对偶削减 Newton 法对高维样本并不适用, 会出现内存不足的问题.

对于 FW 型算法, Khachiyan [12] 提出了 ϵ- 原始可行解的概念并给出一种计算 $(1+\epsilon)$- 近似椭球的 FW 型算法 (后文简称为 Frank Wolfe-Khachiyan (FW-K) 算法), 该算法的计算复杂度为 $O(mn^2(\frac{1}{\epsilon}+\ln n+\ln\ln m))$. 基于 Khachiyan 的开创性工作, Kumar 和 Yıldırım [13] 提出了一种初始化策略, 它可以将 Khachiyan 算法的复杂度降为 $O(mn^2(\frac{1}{\epsilon}+\ln n))$. 同时, 该初始化策略还可以用于构造给定空间点集的 ϵ 核心集, 这对于计算几何领域有着重要的意义. Yıldırım [14] 利用 Khachiyan 算法计算了空间中给定一组椭球的 MVEE. Todd 和 Yıldırım [15] 在 Khachiyan 算法的基础上提出了 "丢弃点策略", 该策略与 FW 型算法中的后退策略 (away step) 是一致的. 需要说明的是, 丢弃点策略也出现在求解 D- 最优设计问题中 (参见文献 [16]), 因此加入了丢弃点策略的 FW 型算法也称为 Wolfe-Atwood (WA) 算法. Ahipasaoglu 等 [17] 将 ϵ- 原始可行性条件推广为 ϵ- 最优性条件, 并据此证明了 WA 算法具有局部线性收敛的性质. 随后, Kumar 和 Yıldırım [1] 将 WA 算法应用于非线性支持向量机问题上.

本文提出一种梯度下降型算法—坐标轴下降 (coordinate gradient descent, CGD) 算法, 用以求解 MVEE 问题. 一方面, CGD 算法每次只更新迭代点的一个分量, 因此, 在迭代过程中, 矩阵的逆和行列式都可以通过矩阵的秩一修正公式来简化, 极大地降低了计算量. 另一方面, 梯度下降型算法是一类非常成熟的方法, 很多实用、有效的技巧 (如坐标轴的选取准则) 可以被用来提升计算效率. 需要注意的是, 虽然 WA 算法在每次迭代过程中也是更新迭代点的一个分量, 但它的坐标轴选取准则、迭代方向和迭代步长与 CGD 算法不同. 第 3.3 小节对 WA 算法与 CGD 算法进行详细地比较.

本文余下内容的结构如下: 第 2 节介绍 MVEE 的相关基础理论; 第 3 节提出 "对算法依坐标轴下降" 的概念, 并据此提出 CGD 算法; 第 4 节证明 CGD 算法的全局次线性收敛性和局部线性收敛性; 第 5 节给出数值实验的结果; 第 6 节给出结论.

2 预备知识

首先介绍与本文密切相关的定义和引理 [18].

定义 2.1 设 $f:\mathbb{R}^n\to(-\infty,\infty]$ 是正常函数, 若对于任意 $x\in\mathbb{R}^n$, 有

$$f(x)=f(x^{\downarrow}),$$

其中 $x^{\downarrow}$ 表示对 x 中元素按从大到小的次序重新排列后的向量, 则函数 f 称为置换对称函数.

定义 2.2 设 $X\in\mathbb{S}^n$, 若函数 $\lambda:\mathbb{S}^n\to\mathbb{R}^n$ 满足 $\lambda(X)=(\lambda_1(X),\ldots,\lambda_n(X))^{\mathrm{T}}$ (其中 $\lambda_1(X)\geqslant\lambda_2(X)\geqslant\cdots\geqslant\lambda_n(X)$), 则称 λ 是定义在 $\mathbb{S}^n$ 上的谱函数.

定义 2.3 $g:\mathbb{R}^n\to(-\infty,\infty]$ 是正常函数, 若存在置换对称函数 $f:\mathbb{R}^n\to(-\infty,\infty]$ 和谱函数 $\lambda:\mathbb{S}^n\to\mathbb{R}^n$ 使得 $g=f\circ\lambda$, 则称 g 是 $\mathbb{S}^n$ 上的对称谱函数.

引理 2.1 设 $F=f\circ\lambda$, 其中 $f:\mathbb{R}^n\to(\infty,\infty]$ 是正常的置换对称函数, λ 是定义在 $\mathbb{S}^n$ 上的谱函数, 则 F 是闭凸函数当且仅当 f 是闭凸函数.

2.1 中心对称点集最小体积覆盖椭球

设 $\mathbb{S}^n$、$\mathbb{S}^n_+$ 和 $\mathbb{S}^n_{++}$ 分别表示对称矩阵空间、半正定矩阵空间和正定矩阵空间. 设 $E(H,c)=\{x\mid$

$(x-c)^{\mathrm{T}}H(x-c)\leqslant n\}$ 是 n 维空间中的椭球, 其中 $H\in\mathbb{S}^n_{++}$, $c\in\mathbb{R}^n$, 椭球体积与 $\det(H)$ 呈负相关[19]. 假设 n 维空间中的点集 $\mathcal{X}=\{x_1,\ldots,x_m\}$ 是中心对称的, 则其 MVEE ($E(H,0)$) 也是中心对称的, 且可以由如下凸优化模型表示[12]:

$$\begin{aligned}&\min_{H\in\mathbb{S}^n_{++}}\ -\ln\det H\\&\text{s.t.}\ \ x_i^{\mathrm{T}}Hx_i\leqslant n,\quad i=1,\ldots,m.\end{aligned}\tag{2.1}$$

首先, 注意到模型 (2.1) 等价于

$$\begin{aligned}&\min\ -\ln\det H\\&\text{s.t.}\ \ x_i^{\mathrm{T}}Hx_i\leqslant n,\quad i=1,\ldots,m,\\&\qquad\det H\geqslant\epsilon,\end{aligned}\tag{2.2}$$

其中 ϵ 是一个充分小的正数. 模型 (2.2) 的目标函数是连续函数, 约束集合是一个闭集, 由 Weierstrass 定理可知模型 (2.2) 最优解存在, 进而说明了模型 (2.1) 最优解的存在性. 其次, 注意到模型 (2.1) 的目标函数 $-\ln\det H$ 是置换对称函数 $f(x)=-\sum_{i=1}^n\ln(x_i)$ 和谱函数 $\lambda(H)=(\lambda_1(H),\ldots,\lambda_n(H))$ 的复合, 则由 f 在 $\mathbb{R}^n_+$ 上的凸性和引理 2.1 可知, $-\ln\det(H)$ 在 $H\in\mathbb{S}^n_+$ 上是凸函数. 需要注意的是, 当定义域变为 $\mathbb{S}^n_{++}$ 时, $-\ln\det(H)$ 在 $H\in\mathbb{S}^n_{++}$ 上还是严格凸函数. 根据文献 [20] 所述, 函数 $f:E\to(-\infty,\infty]$ (其中 E 为某给定空间) 在 $\mathrm{dom}(f)$ 上是严格凸的当且仅当对于任意 $x\in\mathrm{dom}(f)$, $v\in E$, 函数 $g(t)=f(x+tv)$ 是严格凸函数, 其中 $x+tv\in\mathrm{dom}(f)$. 因此, 欲说明 $-\ln\det H$ 在 $H\in\mathbb{S}^n_{++}$ 上是严格凸函数, 只需证明对于任意 $F\in\mathbb{S}^n_{++}$, $g(t)=-\ln\det(H+tF)$ 是严格凸函数. 根据文献 [19] 可知,

$$g'(t)=\mathrm{tr}(-(H+tF)^{-1}F),\quad g''(t)=\mathrm{tr}(H^{-1}FH^{-1}F)=\|H^{-1}F\|^2_{\mathrm{fro}},$$

其中 $\|\cdot\|_{\mathrm{fro}}$ 表示矩阵的 Frobenius 范数, tr 表示矩阵的迹. 由于 H 和 F 都是正定矩阵, 因此, $g''(t)>0$, 即证明了 $-\ln\det H$ 在 $H\in\mathbb{S}^n_{++}$ 上是严格凸的. 于是模型 (2.1) 有唯一最优解.

下面考虑模型 (2.1) 的对偶问题及最优对偶解的存在性. 假设向量 $u\in\mathbb{R}^m_+$ 为模型 (2.1) 中约束条件 $x_i^{\mathrm{T}}Hx_i\leqslant n$ 所对应的 Lagrange 乘子, 则对应的 Lagrange 函数为 $L(H;u)=-\ln\det H+\sum_{i=1}^m u_i(x_i^{\mathrm{T}}Hx_i-n)$, 对偶函数为

$$q(u)=\min_{H\in\mathbb{S}^n_{++}}L(H,u)=\begin{cases}\ln\det(XUX^{\mathrm{T}})+n(1-e^{\mathrm{T}}u), & \text{若 } H^{-1}=XUX^{\mathrm{T}},\\-\infty, & \text{其他},\end{cases}$$

其中 U 是一个对角矩阵, 其对角线元素为 $U_{ii}=u_i$, $i=1,\ldots,m$. 据此可以得到模型 (2.1) 的 Lagrange 对偶规划模型:

$$\begin{aligned}&\min_u\ h(u)=-\ln\det(XUX^{\mathrm{T}})+n(e^{\mathrm{T}}u-1)\\&\text{s.t.}\ \ u\geqslant 0,\end{aligned}\tag{2.3}$$

或者

$$\begin{aligned}&\min_u\ g(u)=-\ln\det(XUX^{\mathrm{T}})\\&\text{s.t.}\ \ u\in\Delta_m,\end{aligned}\tag{2.4}$$

其中 $\Delta_m = \{u \mid e^{\mathrm{T}}u = 1, u \geqslant 0\}$ 是单位单纯形集合. 注意到, 当中心对称点集 $\mathcal{X}$ 构成的矩阵 $X = [x_1, \ldots, x_m]$ 是行满秩时, $\mathcal{X}$ 的覆盖椭球 $E(H, 0)$ 的维数为 n, 因此存在 $H \in \mathbb{S}^n_{++}$ 使得约束条件 $x_i^{\mathrm{T}} H x_i \leqslant n$ 成立. 又因为模型 (2.1) 的最优值有限, 所以根据非线性 Farkas 引理[18] 可知模型 (2.1) 与对偶规划模型 (2.3) 或 (2.4) 之间强对偶性成立, 且对偶规划模型的最优解集非空. 为了不失一般性, 全文假设矩阵 $X = [x_1, \ldots, x_m]$ 是行满秩的.

下面通过定理给出模型 (2.1) 的最优性条件.

定理 2.1 假设对偶规划模型 (2.4) 的最优解为 u^*, 则最优性条件为

$$
\begin{aligned}
&u^* \in \Delta_m, \\
&x_i^{\mathrm{T}}(XU^*X^{\mathrm{T}})^{-1}x_i \leqslant n, \quad i = 1, \ldots, m, \\
&u_i^*(x_i^{\mathrm{T}}(XU^*X^{\mathrm{T}})^{-1}x_i - n) = 0, \quad i = 1, \ldots, m.
\end{aligned}
$$

定理 2.1 可由 KKT (Karush-Kuhn-Tucker) 条件直接得到. 附录中将给出定理 2.1 的一种新的证明方法.

Khachiyan[12] 指出任意点集 $\mathcal{X}$ 的 MVEE 可以由中心对称点集的 MVEE 得到, 具体的变换方法参见文献 [12]. 基于此, 空间中任意点集 $\mathcal{X}$ 的 MVEE 可由中心对称点集的 MVEE 得到. 因此, 计算 MVEE 的核心工作是求解凸优化模型 (2.1) 或 (2.4).

2.2 计算最小体积覆盖椭球算法

求解模型 (2.1) 的经典算法是 Newton 法[11], 求解模型 (2.4) 的经典算法是 FW 型算法 (参见文献 [12,13,17,21,22]). 大量的数值实验表明, 当样本规模较大时, 尤其是当样本中点的维数较大时, FW 型算法比 Newton 法效率更高[19]. Atwood[16] 首次使用 FW 型算法来计算 MVEE. Khachiyan[12] 首次给出了 FW 型算法在 MVEE 问题上的计算复杂度. Kumar 和 Yıldırım[13] 在 Khachiyan 工作的基础上提出了一种初始化策略, 可以提升计算效率 (后文将该方法简称为 Frank Wofe-Kumar Yildirim (FW-KY) 算法). Ahipasaoglu 等[17] 提出了 “丢弃点” 策略并发现 FW 型算法在求解 MVEE 问题上是局部线性收敛的.

我们以 FW-K 算法为例介绍 FW 型算法在 MVEE 问题上的求解流程. 假设原始 - 对偶最优解对为 (H^*, u^*), 则由最优性条件可得

$$
\begin{aligned}
&u_i^* \geqslant 0, \quad i = 1, \ldots, m, \\
&e^{\mathrm{T}}u^* = 1, \\
&x_i^{\mathrm{T}}H^*x_i \leqslant n, \quad i = 1, \ldots, m, \\
&u_i^*(x_i^{\mathrm{T}}H^*x_i - n) = 0, \quad i = 1, \ldots, m, \\
&(XU^*X^{\mathrm{T}})^{-1} = H^*.
\end{aligned}
$$

因此, 模型 (2.4) 的可行解 u 是其最优解当且仅当 $H(u) = (XUX^{\mathrm{T}})^{-1}$ 是模型 (2.1) 的可行解且互补松弛性条件成立. 基于此, Khachiyan[12] 提出了 ϵ- 原始可行解的概念.

定义 2.4 假设 u 是模型 (2.4) 的可行解, 若 $H(u) = (XUX^{\mathrm{T}})^{-1}$ 满足 $x_i^{\mathrm{T}}H(u)x_i \leqslant (1+\epsilon)n$, $i = 1, \ldots, m$, 则 u 称为 ϵ- 原始可行解.

FW-K 算法用模型 (2.4) 目标函数的一阶 Taylor 展式代替原目标函数构造子问题, 子问题的最优解可以用来构造模型 (2.4) 目标函数值的下降方向, 利用该下降方向和最优步长, 模型 (2.4) 便可以迭代地求解. 假设当前迭代点为 u_k, 则模型 (2.4) 的子问题可写为

$$\begin{aligned} &\max_{u} \ \kappa(u_k)^{\mathrm{T}} u \\ &\text{s.t.} \ \ u \in \Delta_m, \end{aligned}$$

其中 $\kappa(u_k) = \nabla g(u_k)$. 令 $j = \arg\max_{i=1,\ldots,m} \kappa(u_k)_i$, 其中 $\kappa(u_k)_i$ 表示 $\kappa(u_k)$ 的第 i 个分量, 则 $e_j = (0,\ldots,1,\ldots,0)^{\mathrm{T}}$ (其中第 j 个分量为 1) 是上述子问题的最优解. 假设当前迭代的步长为 λ_k, 则新的迭代点为 $u_{k+1} = u_k + \lambda_k(e_j - u_k)$. 值得注意的是, 存在一个最优的步长 λ_k 使得目标函数 $g(u)$ 在该步长下沿方向 $e_j - u_k$ 改进后函数值可以达到最小, 即

$$\frac{d}{d\lambda} \ln\det(u_k + \lambda(e_j - u_k)) = 0 \Rightarrow \lambda_k = \frac{\kappa_j - n}{n(\kappa_j - 1)},$$

其中 $\kappa_j = \kappa(u)_j$. 下面简述 FW-K 算法的具体步骤, 见算法 1 (参见文献 [12]).

算法 1 FW-K 算法求解模型 (2.4)

1: 输入: $\mathcal{X} = \{x_1,\ldots,x_m\}$, $\epsilon > 0$.
2: 第 0 步. 初始化: $u_0 = (\frac{1}{m},\ldots,\frac{1}{m})$.
3: 第 1 步. 若不收敛, 则执行循环.
4: 第 1.1 步. $j \leftarrow \arg\max_{i=1,\ldots,m} \kappa(u_k)_i$,$\lambda_k = \frac{\kappa_j - n}{n(\kappa_j - 1)}$, $u_{k+1} \leftarrow u_k + \lambda_k(e_j - u_k)$.
5: 第 2 步. 输出: $u^* = u_k$.

在 FW-K 算法中, 初始迭代点 u_0 设置为 $(\frac{1}{m},\ldots,\frac{1}{m})$, 而 FW-KY 算法利用基于核心集的初始化策略来设置 u_0, 该初始化策略效率更高 (参见文献 [13]). 需要注意的是, FW-K 算法和 FW-KY 算法都只能得到 ϵ- 原始可行解, 而 ϵ- 原始可行解并不一定近似满足最优性条件, 基于此, 文献 [15,17,19] 提出了 ϵ- 近似最优解的概念.

定义 2.5 假设 u 是模型 (2.4) 的 ϵ- 可行解, 若 u 还满足对于所有的 $i \in \{i \mid u_i > 0\}$, $x_i^{\mathrm{T}} H(u) x_i \geqslant (1-\epsilon)n$ $(i = 1,\ldots,m)$ 成立, 则 u 称为 ϵ- 近似最优解.

根据定义 2.5 可知, 相比 ϵ- 可行解而言, ϵ- 近似最优解还满足互补松弛性条件, 因此其近似满足最优性条件. 为了求得 ϵ- 近似最优解, Todd 和 Yıldırım [15] 引入了 FW 型算法中的 "后退策略" 并提出了 WA 算法 [17] (见算法 2).

算法 2 WA 算法求解模型 (2.4)

1: 输入: $\mathcal{X} = \{x_1,\ldots,x_m\}$, $\epsilon > 0$.
2: 第 0 步. 应用 FW-KY 初始化策略设置 u_0.
3: 第 1 步. 若不收敛, 则执行循环.
4: 第 1.1 步. $j_+ \leftarrow \arg\max_{i=1,\ldots,m} \kappa(u_k)_i$, $\kappa_+ \leftarrow \kappa(u_k)_{j+}$; $j_- \leftarrow \arg\min_{i:u_i>0} \kappa(u_k)_i$, $\kappa_- \leftarrow \kappa(u_k)_{j-}$; $\epsilon_+ \leftarrow \frac{\kappa_+}{n} - 1$, $\epsilon_- \leftarrow 1 - \frac{\kappa_-}{n}$, $\epsilon_k \leftarrow \max\{\epsilon_+, \epsilon_-\}$.
5: 第 1.2 步. 若 $\epsilon_k = \epsilon_+$, 则 $\lambda_k = \frac{\kappa_j - n}{n(\kappa_j - 1)}$,$u_{k+1} \leftarrow u_k + \lambda_k(e_{j+} - u_k)$, 否则 $\lambda_k = \min\{\frac{n-\kappa_j}{n(\kappa_j-1)}, \frac{u_{j-}}{1-u_{j-}}\}$, $u_{k+1} \leftarrow u_k + \lambda_k(u_k - e_{j-})$.
6: 第 2 步. 输出: $u^* = u_k$.

对于 WA 算法有 3 点需要说明.

(1) 在 WA 算法迭代的第 k 步会产生对应的椭球 $E(H_k,0)$, 点集 $\mathcal{X}$ 中有一些点 x_i 会落入椭球内部, 即 $x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}x_i \leqslant n$; 而另一些点 x_j 会落在椭球外部, 即 $x_j^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}x_j > n$. WA 算法当前迭代步的目标是让模型 (2.4) 目标函数 $g(u)$ 的梯度的每个分量, 即 $x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}x_i$, $i=1,\ldots,m$, 尽可能接近 n, 直到近似最优性条件 $(1-\epsilon)n \leqslant x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}x_i \leqslant (1+\epsilon)n$ $(i=1,\ldots,m)$ 满足.

(2) 在 WA 算法迭代过程中, 任取某个落在当前椭球外部的点 x_i, 则下一步迭代点为 $u^{\dagger}=u+\lambda(e_i-u)$. 根据 WA 算法, 若选取步长 $\lambda=\frac{\kappa(u_k)_i-n}{n(\kappa(u_k)_i-1)}$, 则可以得到 $x_i^{\mathrm{T}}(XU^{\dagger}X^{\mathrm{T}})^{-1}x_i=n$, 即该点 x_i 落在了下一步迭代所对应椭球的边界上. 类似地, 任取某个落在当前椭球内部的点 x_j, 则下一步迭代点为 $u^{\dagger}=u+\lambda(e_j-u)$. 根据 WA 算法, 若选取步长 $\lambda=\frac{n-\kappa(u_k)_j}{n(\kappa(u_k)_j-1)}$, 则可以得到 $x_j^{\mathrm{T}}(XU^{\dagger}X^{\mathrm{T}})^{-1}x_j=n$, 即该点 x_j 同样也落在了下一步迭代所对应椭球的边界上. 基于此, 目标函数 $g(u)$ 梯度的每个分量都会渐近地趋于 n.

(3) 在 WA 算法迭代的每一步都需要目标函数 $g(u)$ 的梯度在当前迭代点处值的信息 (即 $\kappa(u)=(x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}x_i)_{i=1}^m$), 其中主要的计算量是求矩阵 XUX^{T} 的逆矩阵. 我们可以对其进行 Cholesky 分解, 即 $XUX^{\mathrm{T}}=\phi LL^{\mathrm{T}}$, 其中 ϕ 是一个常数. 这样做的好处有两个. 首先, 经过 Cholesky 分解后矩阵 XUX^{T} 的逆矩阵很容易求出; 其次, 下一步迭代中对应 $(XU^{\dagger}X^{\mathrm{T}})$ 的 Choleksy 分解可以由秩一修正公式得到, 即 $\frac{\phi}{1-\lambda}(XU^{\dagger}X^{\mathrm{T}})=LL^{\mathrm{T}}+\frac{\lambda}{1-\lambda}\phi x_jx_j^{\mathrm{T}}$, 而且逆矩阵 $(XU^{\dagger}X^{\mathrm{T}})^{-1}$ 也可以通过秩一修正公式得到, 即

$$\begin{aligned}(XU^{\dagger}X^{\mathrm{T}})^{-1} &= ((1-\lambda)XUX^{\mathrm{T}}+\lambda x_jx_j^{\mathrm{T}})^{-1}\\ &= \frac{1}{1-\lambda}\left[(XUX^{\mathrm{T}})^{-1}+\frac{\lambda}{1+\lambda(\kappa(u)_j-1)}(XUX^{\mathrm{T}})^{-1}x_jx_j^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}\right].\end{aligned}$$

3 坐标轴梯度下降算法

本节提出利用坐标轴梯度下降算法 (简称为 CGD 算法) 来计算 MVEE 问题. 值得注意的是, 虽然 WA 算法在每步迭代中也是按坐标轴更新点列, 但迭代方向却不是负梯度方向, 这也是 WA 算法与 CGD 算法之间最大的区别. 第 3.3 小节将对 CGD 算法与 FW 型算法详细对比.

首先给出依坐标轴光滑的概念.

定义 3.1 设 $f:\mathbb{R}^n\to\mathbb{R}$ 是一阶连续可微函数, 若对于任意 $x\in\mathbb{R}^n$, $\theta\in\mathbb{R}$, $i=1,\ldots,m$, 存在 $L_i>0$ 使得 $|\nabla_i f(x+\theta e_i)-\nabla_i f(x)|\leqslant L_i|\theta|$ 成立, 则称函数 f 依坐标轴光滑.

根据定义 3.1 可知, 模型 (2.3) 和 (2.4) 的目标函数并不是依坐标轴光滑的, 因此提出一个新的概念—对算法依坐标轴光滑.

3.1 对算法依坐标轴光滑

定义 3.2 假设 u_k 和 u_{k+1} 是由算法 $\mathcal{A}$ 产生的任意两个相邻的迭代点, 且 u_k 和 u_{k+1} 除了第 i_k 个元素外其他元素都相同. 设 $f:\mathbb{R}^n\to\mathbb{R}$ 是一阶连续可微函数. 若对于每个 k, 都存在一个正数 L_{i_k} 使得 $|\nabla_{i_k}f(u_{k+1})-\nabla_{i_k}f(u_k)|\leqslant L_{i_k}\|u_{k+1}-u_k\|$ 成立, 则 f 称为对算法 $\mathcal{A}$ 依坐标轴光滑, L_{i_k} 称为算法 $\mathcal{A}$ 在第 k 步迭代的光滑系数.

需要在这里强调的是, 对算法依坐标轴光滑性要弱于依坐标轴光滑性, 其描述光滑系数 L_{i_k} 并不是一个常数, 而是与迭代的步数 k 相关的. 下面将说明模型 (2.4) 的目标函数 $g(u)$ 对算法依坐标轴光滑. 首先给出 $g(u)$ 的一个性质.

引理 3.1 假设 u 是模型 (2.4) 的一个可行解, 令 $\kappa(u)=\nabla_u g(u)$, 且 $\kappa(u)_i$ 表示 $\kappa(u)$ 的第 i 个分量, 则有

$$|\kappa(u+\theta_i e_i)_i-\kappa(u)_i|\leqslant(\kappa(u)_i)^2|\theta_i|,\quad \forall\,\theta_i\geqslant 0,$$
$$|\kappa(u+\theta_i e_i)_i-\kappa(u)_i|\leqslant n\kappa(u)_i|\theta_i|,\quad \frac{\kappa(u)_i-n}{\kappa(u)_i n}\leqslant\theta_i\leqslant 0.$$

证明 对于 $\theta_i\geqslant 0$, 由秩一修正公式可得

$$\begin{aligned}|\kappa(u+\theta_i e_i)_i-\kappa(u)_i|&=|x_i^{\mathrm{T}}(XUX^{\mathrm{T}}+\theta_i x_i x_i^{\mathrm{T}})^{-1}x_i-x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}x_i|\\&=\left|x_i^{\mathrm{T}}\left(\frac{-\theta_i(XUX^{\mathrm{T}})^{-1}x_i x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}}{1+\theta_i\kappa(u)_i}\right)x_i\right|\\&=\frac{\theta_i(\kappa(u)_i)^2}{1+\theta_i\kappa(u)_i}\\&\leqslant(\kappa(u)_i)^2|\theta_i|.\end{aligned}$$

对于 $\frac{\kappa(u)_i-n}{\kappa(u)_i n}\leqslant\theta_i\leqslant 0$, 有 $\frac{\kappa(u)_i^2}{1+\kappa(u)_i\theta_i}\leqslant\kappa(u)_i n$. 于是, 由秩一修正公式可得

$$\begin{aligned}|\kappa(u+\theta_i e_i)_i-\kappa(u)_i|&=|x_i^{\mathrm{T}}(XUX^{\mathrm{T}}+\theta_i x_i x_i^{\mathrm{T}})^{-1}x_i-x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}x_i|\\&=\left|x_i^{\mathrm{T}}\left(\frac{-\theta_i(XUX^{\mathrm{T}})^{-1}x_i x_i^{\mathrm{T}}(XUX^{\mathrm{T}})^{-1}}{1+\theta_i\kappa(u)_i}\right)x_i\right|\\&=-\frac{\theta_i\kappa(u)_i^2}{1+\theta_i\kappa(u)_i}\\&\leqslant n\kappa(u)_i|\theta_i|.\end{aligned}$$

证毕. □

引理 3.1 揭示模型 (2.4) 的目标函数 $g(u)$ 具有某种光滑性质, 这种光滑性比依坐标轴光滑性要弱一些, 因为引理 3.1 中的描述光滑系数 L_{i_k} 与 $\kappa(u)_i$ $(i=1,\ldots,m)$ 相关. 需要注意的是, 模型 (2.3) 的目标函数 $h(u)$ 也具有相同的性质, 这是因为 $|\nabla_i h(u+\theta e_i)-\nabla_i h(u)|=|\kappa(u+\theta e_i)_i-\kappa(u)_i|$. 这也是我们选择模型 (2.3) 来实施坐标轴下降算法的一个原因. 在第 3.3 小节可以看到, 以模型 (2.3) 为基础进行分析, 有助于把求解 MVEE 所有 FW 型算法都纳入到统一的框架中来对比分析其坐标轴选取方法, 这是模型 (2.4) 所不具备的.

3.2 常数步长下的 CGD 算法

本节针对模型 (2.3), 提出坐标轴下降算法的理论框架, 并作出简要的分析. 设模型 (2.3) 的目标函数为 $h(u)=g(u)+n(1-e^{\mathrm{T}}u)$, $P_\Omega(x)$ 表示点 x 到集合 Ω 上的投影. CGD 算法的具体步骤见算法 3.

算法 3 坐标轴下降算法 (CGD 算法)

1: 输入: $\mathcal{X}=\{x_1,\ldots,x_m\}$, $\epsilon>0$.
2: 第 0 步. 初始化: 应用 FW-K 或者 FW-KY 初始化策略设置 u_0.
3: 第 1 步. 若不收敛, 则执行循环.
4: 第 1.1 步. 选择 $j=\arg\max_{i=1,\ldots,m}|\nabla h(u_k)_i|$.
5: 第 1.2 步. $u_{k+1}=P_{u\geqslant 0}(u_k-\frac{1}{L_j(u_k)}\nabla h(u_k)_j)$.
6: 第 2 步. 输出: $u^*=u_k$.

CGD 算法是坐标轴梯度下降算法, 坐标轴选取方案采用 Gauss-Southwell 准则. 根据引理 3.1, 如果在当前迭代点处 $\kappa(u)_j \geqslant n$, 则光滑系数 L_j 可取为 $(\kappa(u)_j)^2$, 且下一步迭代点 $u^\dagger$ 可更新为

$$\begin{aligned} u^\dagger &= P_{u\geqslant 0}\left(u - \frac{1}{L_j}\nabla h(u)_j e_j\right) \\ &= u - \frac{1}{(\kappa(u)_j)^2}(n-\kappa(u)_j)e_j \\ &= u + \frac{\kappa(u)_j - n}{(\kappa(u)_j)^2}e_j. \end{aligned}$$

相反地, 如果在当前迭代点处 $\kappa(u)_j \leqslant n$, 则光滑系数 L_j 可取为 $n\kappa(u)_j$, 且下一步迭代点 $u^\dagger$ 可更新为

$$\begin{aligned} u^\dagger &= P_{u\geqslant 0}\left(u - \frac{1}{L_j}\nabla h(u)_j e_j\right) \\ &= P_{u\geqslant 0}\left(u - \frac{1}{(n\kappa(u)_j)}(n-\kappa(u)_j)e_j\right) \\ \Rightarrow u_j^\dagger &= \begin{cases} u_j + \dfrac{\kappa(u)_j - n}{(n\kappa(u)_j)}, & 若\ u_j \geqslant \dfrac{n-\kappa(u)_j}{n\kappa(u)_j}, \\ 0, & 若\ u_j < \dfrac{n-\kappa(u)_j}{n\kappa(u)_j}, \end{cases} \end{aligned}$$

而 $u^\dagger$ 的其他分量 i $(i \neq j)$ 等于 u_i.

根据定义 3.2, 模型 (2.3) 的目标函数 $h(u)$ 是对 CGD 算法依坐标轴光滑的. 为了叙述方便, 我们将 CGD 算法在 $\kappa(u)_j \geqslant n$ 和 $\kappa(u)_j \leqslant n$ 的情形下选择的坐标轴分别记为 $j+$ 和 $j-$.

3.3 CGD 算法与 WA 算法的比较

本节从坐标轴选取的角度解释 FW 型算法, 并将其与 CGD 算法进行对比. 首先考虑 FW-K 算法. 由算法 1 可知, FW-K 算法选择坐标轴的方式为 $j \leftarrow \arg\max_{i=1,\ldots,m}\kappa(u_k)_i$, 所以 $\kappa(u_k)_j \geqslant n$, 否则对于所有 $i=1,\ldots,m$, 都有 $\kappa(u_k)_i \leqslant n$, 即 u_k 为 ϵ- 原始可行解, FW-K 算法终止. 根据引理 3.1 可知, 当 $\kappa(u_k)_j \geqslant n$ 时, 模型 (2.3) 目标函数 $h(u)$ 的光滑系数为 $\kappa(u_k)_j^2$, 由此可得 FW-K 算法在每步迭代过程中选择目标函数光滑性最差的坐标轴方向作为改进方向, 这种坐标轴选取策略与文献 [23] 中给出的坐标轴选取策略相似. 其次考虑 WA 算法. 由算法 2 可知, WA 算法选择坐标轴的方式为, 当 $\kappa_+ - n \geqslant n - \kappa_-$ 时, 选择坐标轴 $j+$, 否则选择坐标轴 $j-$. 考虑到 $\nabla h(u) = n - \kappa(u)$, 因此, WA 算法在每步迭代过程中实际上是按照 $\max_{i=1,\ldots,m}\{|\nabla_i h(u)|\}$ 来选取坐标轴, 即按照 Gauss-Southwell 坐标轴选取准则选取坐标轴. 由于大量的数值实验已经说明 Gauss-Southwell 坐标轴选取方式要优于现存的其他坐标轴选取方式 [24], 因此, 这也是 WA 算法比 FW-K 算法更高效的原因之一.

表 1 对比了 CGD 算法和 WA 算法. 它们的联系在于, CGD 算法和 WA 算法均使用 Gauss-Southwell 准则来选取坐标轴. 它们之间的区别主要有 3 个方面. 首先, 假设选取的坐标轴为 j, CGD 算法选取目标函数负梯度方向的第 j 个分量作为迭代方向, 而 WA 算法选取 $e_j - u$ 作为迭代方向. 值得注意的是, 这两个方向都是目标函数 $h(u)$ 的下降方向. 其次, WA 算法的步长是精确步长, 每步迭代都需要求出最优步长 $\lambda^* = \arg\min_\lambda\{-\ln\det(XU(\lambda)X^{\mathrm{T}})\}$, 而 CGD 算法选取的步长是常数步长, 即 $\frac{1}{L_j}$. 需要注意的是, 由于在每步迭代中, L_j 会发生变化, 这里称其为常数步长是因为步长的选取无须任何线搜索策略, 可以直接获得. 最后, WA 算法在每步迭代中需要求解一个线性规划子问题, 因此,

表 1 WA 算法与 CGD 算法对比表

算法	WA 算法	CGD 算法
选取坐标轴的准则	Gauss-Southwell 准则	Gauss-Southwell 准则
方向	$e_j - u$	$-e_j$
步长	$\arg\min_\lambda\{-\ln\det(XU(\lambda)X^{\mathrm{T}})\}$	$\frac{1}{L_j}$
类别	基于线性规划迭代算法	基于梯度迭代算法

它是基于线性规划的迭代算法; 而 CGD 算法在每步迭代中只需要知道目标函数梯度信息, 因此, 它是基于梯度的迭代算法.

4 CGD 算法的收敛性分析

4.1 CGD 算法的全局收敛性

本节讨论 CGD 算法的收敛性. 首先证明 CGD 算法是全局次线性收敛的, 接着证明 CGD 算法具有局部线性收敛性, 最后分析 CGD 算法的计算复杂度. 为了后文叙述方便, 下面将 CGD 算法的迭代过程分为 3 类.

(1) 若在该迭代步坐标轴 $j+$ 被选中, 则称该步迭代为上升迭代;

(2) 若在该迭代步坐标轴 $j-$ 被选中且 $\frac{n-\kappa_-}{\kappa_- n} < u_{j-}$, 则称该步迭代为下降迭代;

(3) 若在该迭代步坐标轴 $j-$ 被选中且 $\frac{n-\kappa_-}{\kappa_- n} \geqslant u_{j-}$, 则称该步迭代为删除点迭代.

引理 4.1 令 u 和 $u^\dagger$ 是 CGD 算法产生的两个相邻的点, 则

$$h(u)-h(u^\dagger) \geqslant \begin{cases} \dfrac{(n-\kappa_+)^2}{2\kappa_+^2}, & \text{若该步迭代是上升迭代,} \\ \dfrac{(n-\kappa_-)^2}{2n\kappa_-}, & \text{若该步迭代是下降迭代,} \\ 0, & \text{若该步迭代是删除点迭代.} \end{cases}$$

证明 首先, 若该步迭代是上升迭代, 此时步长为 $\lambda = \frac{\kappa_+ - n}{\kappa_+^2}$, 则

$$h(u)-h(u^\dagger) = \ln\left(2-\frac{n}{\kappa_+}\right) - \frac{n(\kappa_+-n)}{\kappa_+^2}. \tag{4.1}$$

下面证明, 在上升迭代步中, 函数值改进量大于等于 $\frac{(n-\kappa_+)^2}{2\kappa_+^2}$.

设 $g_1(t) = \ln(2-\frac{n}{t}) - \frac{n(t-n)}{t^2} - \frac{(n-t)^2}{2t^2}$, 其中 $t \geqslant n$. 由于 $g_1'(t) = -\frac{n}{t}(\frac{(n-t)^2}{t^2(n-2t)}) \geqslant 0$, 因此, $g_1(t)$ 是单调非减的. 又因为 $g_1(n)=0$, 所以对于任意 $t \geqslant n$ 有 $g_1(t) \geqslant 0$. 将 $t=\kappa_+$ 代入到 $g_1(t)$ 中并结合 (4.1) 可得

$$h(u)-h(u^\dagger) \geqslant \frac{(n-\kappa_+)^2}{2\kappa_+^2}.$$

其次, 若该步迭代是下降迭代, 此时步长为 $\lambda = \frac{\kappa_- - n}{n\kappa_-}$, 则

$$h(u)-h(u^\dagger) = \ln\left(\frac{\kappa_-}{n}\right) + \frac{n}{\kappa_-} - 1. \tag{4.2}$$

下面证明, 在下降迭代步中, 函数值改进量大于等于 $\frac{(n-\kappa_-)^2}{2n\kappa_-}$.

设 $g_2(t)=\ln(\frac{t}{n})+\frac{n}{t}-1-\frac{(n-t)^2}{2nt}$, 其中 $t\leqslant n$. 由于 $g_2'(t)=-\frac{(n-t)^2}{2nt^2}\leqslant 0$, 因此, $g_2(t)$ 对于 $t\leqslant n$ 是单调非增的. 又因为 $g_2(n)=0$, 所以对于任意 $t\leqslant n$ 有 $g_2(t)\geqslant 0$. 将 $t=\kappa_-$ 代入到 $g_2(t)$ 中并结合 (4.2) 可得

$$h(u)-h(u^\dagger)\geqslant\frac{(n-\kappa_-)^2}{2n\kappa_-}.$$

最后, 若该步迭代是删除点迭代, 此时步长为 $\lambda=-u_{j-}$, 则

$$h(u)-h(u^\dagger)=nu_{j-}+\ln(1-u_{j-}\kappa_-). \tag{4.3}$$

下面证明, 在删除点迭代步中, 函数值非减.

考虑 $0<t<\frac{n-\kappa_-}{n\kappa_-}$, 其中 $\kappa_-<n$, 设 $g_3(t)=nt+\ln(1-t\kappa_-)$. 由于 $g_3'(t)=n-\frac{\kappa_-}{1-\kappa_-t}$, 且 $t<\frac{n-\kappa_-}{n\kappa_-}$, 所以 $g_3'(t)\leqslant 0$. 于是可得 $g_3(t)$ 对于 $t>0$ 是单调非增的. 又因为 $g_3(0)=0$, 所以对于任意 $t>0$ 有 $g_3(t)\geqslant 0$. 将 $t=u_{j-}$ 代入到 $g_3(t)$ 中并结合 (4.3) 可得

$$h(u)-h(u^\dagger)\geqslant 0.$$

证毕. □

由第 2.1 小节可知, 模型 (2.3) 的最优解集非空. 下面证明 CGD 算法具有全局收敛性, 收敛到某个对偶最优解 u^* 且收敛速度是次线性的. 我们的分析基于文献 [12,17,19,25] 的研究结论.

假设初始点 u_0 是 ϵ_0- 原始可行的, 则在 FW-K 初始化策略下, $\epsilon_0\leqslant m-1$ [12]; 而在 FW-KY 初始化策略下, $\epsilon_0\leqslant n^5-1$ [19]. 令 $c=\max\{m-1,n^5-1\}$. 需要注意的是, 假设 u 是 δ- 原始可行解, 若 $\delta>1$, 则只会出现上升迭代的情形, 此时 δ 会单调递减趋向于 1 [19]. 因此, 我们按照文献 [12] 中的分析思路把 CGD 算法的迭代过程分成 $1<\delta<c$ 和 $\delta\leqslant 1$ 两个阶段.

在第 1 阶段, 令 $H(u_k)=(XU_kX^{\mathrm{T}})^{-1}$, $f(H(u_k))=-\ln\det H(u_k)$. 根据最优性条件, 若 u^* 是对偶模型 (2.3) 的最优解, 则 $f(H(u^*))$ 是原规划模型 (2.1) 的最优解. 假设 u 和 $u^\dagger$ 是相邻的两个迭代点, 则由引理 4.1 后面的分析可知,

$$\begin{aligned}f(H(u))-f(H(u^\dagger))&=h(u)-h(u^\dagger)+n(e^{\mathrm{T}}u-e^{\mathrm{T}}u^\dagger)\\&=\ln\left(2-\frac{n}{\kappa(u)_+}\right)\\&=\ln\left(2-\frac{1}{1+\delta}\right)\\&\geqslant c_1\ln(1+\delta),\end{aligned} \tag{4.4}$$

其中 c_1 是一个常数, 当采用 FW-K 初始化策略时, $c_1=\frac{\ln(2m-1)}{\ln m}-1$; 当采用 FW-KY 初始化策略时, $c_1=\frac{\ln(1+2n^5)}{\ln(1+n^5)}-1$. 根据文献 [12, 引理 3] 可知, 由于 $f(H(u))-f(H(u^*))=g(u)-g(u^*)\leqslant n\ln(1+\delta)$, 因此, $f(H(u))-f(H(u^\dagger))\geqslant\frac{c_1}{n}(f(H(u)))-f(H(u^*))$. 因此,

$$\begin{aligned}f(H(u_k))-f(H(u^*))&\leqslant\left(1-\frac{c_1}{n}\right)^k(f(H(u_0))-f(H(u^*)))\\&\leqslant\begin{cases}\mathrm{e}^{-\frac{c_1k}{n}}n\ln m, & \text{若采用 FW-K 初始化策略},\\ \mathrm{e}^{-\frac{c_1k}{n}}5n\ln n, & \text{若采用 FW-KY 初始化策略}.\end{cases}\end{aligned} \tag{4.5}$$

另一方面, 假设 u_k 是 ϵ_k- 原始可行点, 满足 $\epsilon_k > 1$, 则

$$\begin{aligned} f(H(u_k)) - f(H(u^*)) &\geqslant f(H(u_k)) - f(H(u_{k+1})) \\ &\geqslant c_1 \ln(1+\epsilon_k) \\ &\geqslant c_2, \end{aligned} \tag{4.6}$$

其中 $c_2 = c_1 \ln 2$. 因此由 (4.5) 和 (4.6) 可得, 当采用 FW-K 初始化策略来运行 CGD 算法时, 可以在 $\frac{n}{c_1}(\ln \frac{n \ln m}{c_2})$ 步内得到 1- 原始可行解; 而当采用 FW- KY 初始化策略来运行 CGD 算法时, 可以在 $\frac{n}{c_1}(\ln \frac{5n \ln n}{c_2})$ 步内得到 1- 原始可行解.

在第 2 阶段, 首先考虑仅含有上升迭代和下降迭代两种情形下的收敛性, 为此需要用到如下引理.

引理 4.2 假设 $\delta \leqslant \frac{1}{2}$.

(1) 若 u 不是 δ- 原始可行解, 则每次上升迭代可以使目标函数值 $h(u)$ 至少下降 $\frac{2}{5}\delta^2$;

(2) 若 u 不是 δ- 近似最优解, 则每次下降迭代可以使目标函数值 $h(u)$ 至少下降 $\frac{1}{2}\delta^2$.

证明 假设迭代点 u_k 是 ϵ_k- 原始可行解, 其中 $\epsilon_k \geqslant \delta$. 由引理 4.1 可知, 在上升迭代步骤中,

$$h(u_k) - h(u_{k+1}) \geqslant \frac{(n - \kappa(u_k)_+)^2}{2\kappa(u_k)_+^2}.$$

注意到函数 $\phi(x) = \frac{x^2}{(1+x)^2}$ 的单调非减性以及 $\delta \leqslant \frac{1}{2}$, 于是有

$$\begin{aligned} h(u_k) - h(u_{k+1}) &\geqslant \frac{\epsilon_k^2}{2(1+\epsilon_k)^2} \\ &\geqslant \frac{\delta^2}{2(1+\delta)^2} \\ &\geqslant \frac{2}{5}\delta^2. \end{aligned}$$

类似地, 在下降迭代步骤中, 由于 $\kappa(u_k)_- = (1-\epsilon_k)n$, 其中 $|\epsilon_k| \geqslant \delta$, 因此,

$$\begin{aligned} h(u_k) - h(u_{k+1}) &\geqslant \frac{(n - \kappa(u_k)_-)^2}{2n\kappa(u_k)_-} \\ &\geqslant \frac{\delta^2}{2(1-\delta)} \\ &\geqslant \frac{\delta^2}{2}. \end{aligned}$$

证毕. □

需要注意的是, 引理 4.2 说明, 当 $\delta \leqslant \frac{1}{2}$ 时, CGD 算法每步迭代使目标函数 $h(u)$ 的下降量要大于 WA 算法. 这也说明了在迭代至最优解附近时, CGD 算法效率要高于 WA 算法. 第 5 节的数值实验支持我们的这个结论.

下面分析第 2 阶段的收敛性. 假设 u 是一个 δ- 近似最优解, 其中 $0 < \delta \leqslant 1$, 则

$$\begin{aligned} h(u) - h(u^*) &\leqslant g(u) - g(u^*) + n(e^{\mathrm{T}}u - 1) \\ &\leqslant n\ln(1+\delta) + \sum_{\{i \mid u_i > 0\}} u_i(n - \kappa(u)_i) \\ &\overset{*}{\leqslant} n\ln(1+\delta) + mn\delta u_{\max} \end{aligned}$$

$$\leqslant n(1+mu_{\max})\delta, \tag{4.7}$$

其中 $*$ 是由于对于任意 $i\in\{i\mid u_i>0\}$, 有 $(1-\delta)n\leqslant\kappa(u)_i\leqslant(1+\delta)n$. 我们用 $k(\delta)$ 表示 u 从 δ- 近似最优变为 $\frac{\delta}{2}$- 近似最优所需要的迭代步数. 由 (4.7) 和引理 4.2 可知,

$$k(\delta)\leqslant\frac{n(1+mu_{\max})\delta}{\frac{2(\delta/2)^2}{5}}=\frac{10n(1+mu_{\max})}{\delta}.$$

因此, u 从 1- 近似最优变为 ϵ- 近似最优所需要的迭代步数 $K(\epsilon)$ 为

$$\begin{aligned}K(\epsilon)&\leqslant k(1)+k\left(\frac{1}{2}\right)+\cdots+k\left(\frac{1}{2^{\lceil\ln\frac{1}{\epsilon}\rceil-1}}\right)\\&\leqslant\frac{20n(1+mu_{\max})}{\epsilon}.\end{aligned}$$

迭代步数 $K(\epsilon)$ 为第 2 阶段中仅考虑上升迭代和下降迭代两种情形下点列 $\{u_k\}$ 从 1- 近似最优变为 ϵ- 近似最优所需要的步数. 下面再考虑删除点迭代情形对迭代步数的影响. 根据文献 [19], 删除点迭代情形所需迭代步数不超过 $(2n+m)+2K(\epsilon)$. 因此, 第 2 阶段总的迭代步数不超过 $2n+m+\frac{40n(1+mu_{\max})}{\epsilon}$. 综上可以得出 CGD 算法的全局收敛性结果.

定理 4.1 设 $\{u_k\}$ 是 CGD 算法产生的点列, 则对于任意 $\epsilon>0$, $\{u_k\}$ 收敛到 ϵ- 近似最优解, 且所需迭代步数不超过 $\frac{40n(1+mu_{\max})}{\epsilon}+\frac{n}{c_1}(\ln\frac{n\ln m}{c_2})+(m+2n)$.

4.2 CGD 算法的局部收敛性质

我们采用文献 [17,19] 中的方法来证明 CGD 算法具有局部线性收敛性. 假设 u 是一个 δ- 近似最优解, 设 $H(u)=(XUX^{\mathrm{T}})^{-1}$, 令

$$z_i=\begin{cases}n\delta, & \text{若 } u_i=0,\\ x_i^{\mathrm{T}}H(u)x_i-n, & \text{若 } u_i>0,\end{cases}$$

则有 $u^{\mathrm{T}}z=\sum_{\{i|u_i>0\}}u_i(x_i^{\mathrm{T}}H(u)x_i-n)=n(1-e^{\mathrm{T}}u)$. 考虑如下 z- 扰动模型:

$$\begin{aligned}&\min_{H\in\mathbb{S}^n}\ -\ln\det(H)\\&\text{s.t.}\ \ x_i^{\mathrm{T}}Hx_i\leqslant n+z_i,\quad i=1,\ldots,m,\end{aligned}$$

易知 $H(u)$ 是 z- 扰动模型的最优解.

引理 4.3 假设 u 是一个 δ- 近似最优解, 则

$$h(u)-h^*\leqslant\|u-u^*\|\|z\|.$$

证明 令 $\Phi(z)$ 表示 z- 扰动问题的最优解, 则由 u 是一个 δ- 近似最优解可得

$$\begin{aligned}h(u)&=\Phi(z)+n(e^{\mathrm{T}}u-1)\\&\overset{*}{\geqslant}\Phi(0)+(u^*-u)^{\mathrm{T}}z\\&\geqslant h^*-\|u-u^*\|\|z\|,\end{aligned}$$

其中 $*$ 是由于 $\Phi(z)$ 是一个凸函数且 $u^*\in\Phi'(0)$. □

定理 4.2 CGD 算法产生的点列是局部线性收敛的.

证明 由引理 4.3 和文献 [17, 命题 2.5] 可知, 存在一个常数 M 使得对于任意 δ- 近似最优解 u, 其中 δ 为充分小的正数, 都成立

$$g(u) - g^* \leqslant M\delta^2.$$

由引理 4.2 可知, u 由 δ- 近似最优变为 $\frac{\delta}{2}$- 近似最优最多需要

$$\frac{M\delta^2}{\frac{2(\delta/2)^2}{5}} = 10M$$

步迭代. 因此, u 由 δ- 近似最优变为 ϵ- 近似最优最多需要

$$\begin{aligned} K(\epsilon) &\leqslant k(\delta) + k\left(\frac{\delta}{2}\right) + \cdots + k\left(\frac{\delta}{2^{\lceil \ln \frac{1}{\epsilon} \rceil - 1}}\right) \\ &\leqslant 20M \log_2\left(\frac{1}{\epsilon}\right) \end{aligned}$$

步迭代. 因此, CGD 算法产生的点列是局部线性收敛的. □

4.3 CGD 算法的计算复杂度分析

本小节分析 CGD 算法的计算复杂度. CGD 算法包括两个阶段: 初始化阶段和迭代阶段. 在初始化阶段假设给定初始点 u_0, 我们需要计算目标函数的梯度 $\kappa(u_0) = (x_j^{\mathrm{T}}(XU_0X^{\mathrm{T}})^{-1}x_j)_{j=1}^m$. 设 $XU_0X^{\mathrm{T}} = A^{\mathrm{T}}A$, 对 A^{T} 利用矩阵 QR 分解可得, 存在正交矩阵 Q_0 和上三角矩阵 R_0 使得 $A^{\mathrm{T}} = Q_0R_0$, 于是对于 $j = 1, \ldots, m$, 有

$$\begin{aligned} \kappa_j &= x_j^{\mathrm{T}}(XU_0X^{\mathrm{T}})^{-1}x_j \\ &= x_j^{\mathrm{T}}R_0^{-1}{R_0^{-1}}^{\mathrm{T}}x_j \\ &= (R_0^{-1}x_j)^{\mathrm{T}}(R_0^{-1}x_j). \end{aligned}$$

从上述计算过程可以得到初始化阶段的计算复杂度如下:

(1) 计算 $A^{\mathrm{T}} = XU_0^{\frac{1}{2}}$, 其计算复杂度为 $O(mn)$.

(2) 对 A^{T} 进行 QR 分解, 其计算复杂度为 $O(mn^2)$. 需要注意的是, 由于 QR 分解比 Choleksy 分解在数值效果上更加稳定 (参见文献 [19]), 因此这里采用 QR 分解.

(3) 计算 $R_0^{-1}x_j$, 其计算复杂度为 $O(n^3)$.

(4) 计算 $\kappa_j = (R_0^{-1}x_j)^{\mathrm{T}}(R_0^{-1}x_j)$, 其计算复杂度为 $O(n)$.

由于一共需要计算 m 个不同的 κ_j, 所以初始化阶段整体计算复杂度为 $O(m^2n^2)$.

在迭代阶段, 每步迭代所涉及的计算工作包括:

(1) 查找 κ_j $(j = 1, \ldots, m)$ 中的最大和最小者, 其计算复杂度为 $O(m)$;

(2) 比较 $\frac{\kappa_+ - n}{n}$ 和 $\frac{n - \kappa_-}{n}$, 其计算复杂度为 $O(1)$;

(3) 计算步长 λ 并更新下一步迭代点 $u^\dagger$, 其计算复杂度为 $O(1)$;

(4) 采用秩一修正公式来更新 $XU^\dagger X^{\mathrm{T}} = XUX^{\mathrm{T}} - \lambda x_j x_j^{\mathrm{T}}$, 其计算复杂度为 $O(mn^2)$.

由于所需迭代步数为 $O(\frac{1}{\epsilon})$, 因此整个迭代阶段的计算复杂度为 $O(\frac{1}{\epsilon}(m+n^2+mn^2)) = O(\frac{1}{\epsilon}(mn^2))$. 综上, CGD 算法的整体计算复杂度为 $O(mn^2\frac{1}{\epsilon})$.

5 数值实验

5.1 样本描述

根据文献 [11,17,19] 中给出算例测试结果可知,

(1) FW-KY 初始化策略有助于提升 MVEE 问题的计算效率;

(2) 文献 [25] 中给出的排除点策略可以减少计算时间;

(3) 对于中小规模样本 (样本维数不超过 30, 样本点数量不超过 30,000), 对偶削减 Newton 法计算效率最高. 对于大规模样本 (样本维数超过 500, 样本点数量超过 500,000), 对偶削减 Newton 法会因为内存不足问题而失败, WA 算法是最有效的算法.

根据上述结论, 我们主要测试 CGD 算法和 WA 算法的数值效果. 样本包括小规模、中等规模、大规模和超大规模 4 类. 具体样本描述如下:

(1) 测试样本. 根据文献 [19], 如果按照正态分布来产生样本点, 则会出现大量的样本点位于椭球的边界上, 因此使用文献 [19] 中给出的样本点生成方法.

(2) 样本规模. 4 类测试样本规模的维数及数量分别为: 小规模样本 (样本维数为 10, 样本点数量为 500)、中等规模样本 (样本维数为 30, 样本点数量为 1,800)、大规模样本 (样本维数为 100, 样本点数量为 30,000) 和超大规模样本 (样本维数为 500, 样本点数量为 500,000). 另外, 为了分别研究样本维数和样本点的数量对算法的影响, 我们还测试了两类其他样本, 分别是高维度低数量样本 (样本维数为 500, 样本点数量为 1,000) 和低维度高数量样本 (样本维数为 20, 样本点数量为 100,000).

(3) 参数. 我们的测试均使用了 FW-KY 初始化策略以提升计算效率, 误差界设定为 $\epsilon = 10^{-7}$.

5.2 CGD 算法与 WA 算法数值效果对比

表 2 列示了 CGD 算法和 WA 算法在上述 4 类样本下的数值测试结果, 每个算法在每个样本上测试了 10 次. 需要注意的是, 表中的 “0” 表示计算时间小于 10^{-14} 秒.

对于小规模样本, WA 算法的平均计算时间略短于 CGD 算法, 10 次测试中有 7 次 WA 算法优于 CGD 算法. 对于中等规模样本, WA 算法的平均迭代次数为 268.6 次, 比 CGD 算法的 292.9 次要好,

表 2 CGD 算法与 WA 算法数值效果对比表

样本规模	样本参数		CGD 算法		WA 算法	
	点数量	维数	迭代次数	计算时间	迭代次数	计算时间
小规模	10	500	96	0.00	88	0.00
	10	500	34	0.00	46	0.00
	10	500	147	0.06	138	0.06
	10	500	386	0.08	375	0.06
	10	500	76	0.06	82	0.06
	10	500	165	0.06	133	0.00
	10	500	147	0.06	91	0.06
	10	500	117	0.06	116	0.06
	10	500	102	0.00	104	0.00
	10	500	110	0.00	81	0.00
	平均值		138	0.04	125.4	0.03

(续表)

样本规模	样本参数		CGD 算法		WA 算法	
	点数量	维数	迭代次数	计算时间	迭代次数	计算时间
中等规模	30	1,800	239	0.39	220	0.47
	30	1,800	234	0.16	234	0.14
	30	1,800	248	0.16	268	0.12
	30	1,800	236	0.17	265	0.14
	30	1,800	402	0.25	393	0.25
	30	1,800	229	0.14	232	0.09
	30	1,800	285	0.19	264	0.19
	30	1,800	446	0.11	230	0.11
	30	1,800	246	0.23	233	0.14
	30	1,800	364	0.20	347	0.18
	平均值		292.9	0.20	268.6	0.18
大规模	100	30,000	737	0.53	743	0.53
	100	30,000	658	0.48	710	0.53
	100	30,000	808	0.51	846	0.59
	100	30,000	746	0.56	841	0.66
	100	30,000	713	0.51	780	0.61
	100	30,000	770	0.62	802	0.66
	100	30,000	691	0.51	729	0.56
	100	30,000	661	0.51	680	0.48
	100	30,000	681	0.44	684	0.42
	100	30,000	845	0.53	856	0.62
	平均值		731	0.52	767.1	0.57
超大规模	500	500,000	3,081	63.20	3,687	66.64
	500	500,000	3,107	64.21	3,727	68.75
	500	500,000	3,213	73.09	3,663	76.80
	500	500,000	3,171	64.52	3,730	67.39
	500	500,000	3,084	68.56	3,831	68.06
	500	500,000	3,148	64.85	3,691	67.63
	500	500,000	3,096	67.88	3,611	69.58
	500	500,000	3,122	61.28	3,734	64.01
	500	500,000	3,224	67.17	3,829	74.05
	500	500,000	3,096	66.13	3,782	67.45
	平均值		3,134.2	66.09	3,728.5	69.04

10 次测试中有 6 次 WA 算法优于 CGD 算法. 对于大规模样本, CGD 算法的平均迭代次数和计算时间分别为 731 次和 0.52 秒, 优于 WA 算法的 767.1 次和 0.57 秒. 值得注意的是, 10 次测试都显示了 CGD 算法的优越性. 对于超大规模样本, 两种算法数值效果差异更大, CGD 算法的优势更加明显.

图 1 显示了在 4 类样本下 CGD 算法和 WA 算法的收敛性情形. 图 1(a)–1(d) 分别为小规模样

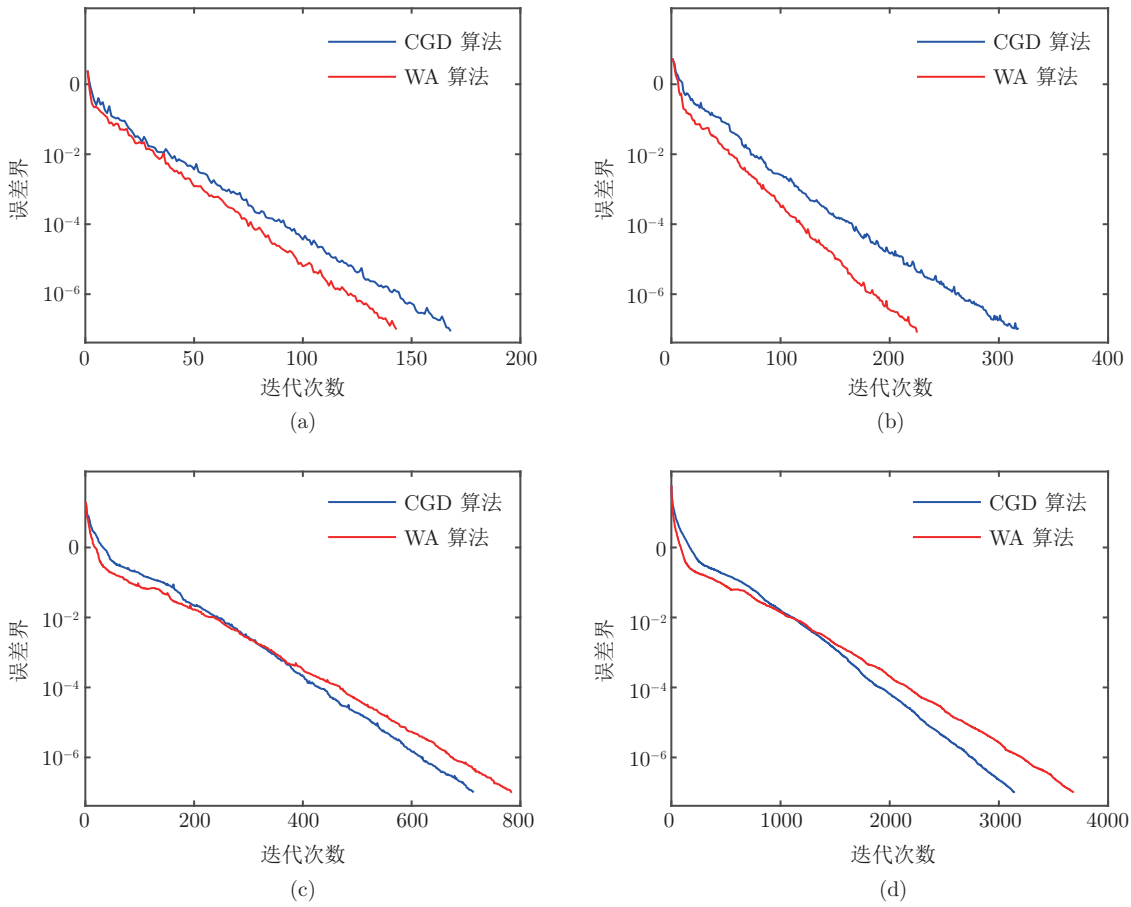

图 1　CGD 算法和 WA 算法数值效果对比图. (a) 小规模样本; (b) 中等规模样本; (c) 大规模样本; (d) 超大规模样本

本、中等规模样本、大规模样本和超大规模样本. 横坐标是迭代次数, 纵坐标是以对数为尺度的误差界 ϵ. 需要说明的是, "以对数为尺度的误差界" 指的是图中纵坐标的值是 $\ln\epsilon$, 对应的坐标的标签是 ϵ 的值, 因此, 每个子图可以反映取对数后的误差界与迭代次数之间的函数关系. 这么做的目的在于, 如果 CGD 算法收敛速度是线性的, 那么取对数之后的误差界 $\ln\epsilon$ 与迭代次数之间会呈现线性关系, 因而通过该图可以更清楚地反映出 CGD 算法的收敛性质. 从图 1 中可以看出, CGD 算法和 WA 算法呈现出全局线性收敛的特性, 这与文献 [19] 里观测到的结果相似. 另一方面, 从图 1(a)–1(d) 中可以很清楚地看出, 存在拐点使得在拐点之前 CGD 算法收敛速度次于 WA 算法, 但拐点之后 CGD 算法收敛速度更快. 这与第 4.1 小节给出的结论是一致的.

我们还进一步研究样本维数和点数量对两种算法计算效果的影响, 测试样本为高维度低数量样本和低维度高数量样本. 图 2 和 3 分别显示了两种算法在高维度低数量样本和低维度高数量样本下的测试效果, 每个算法在每个样本下测试了 10 次, 其中蓝色曲线代表 CGD 算法, 红色曲线代表 WA 算法. 图 2 中的 10 个子图 (横坐标是迭代次数, 纵坐标是以对数为尺度的误差界) 都显示了 CGD 算法比 WA 算法更高效, 而图 3 中有 6 张子图显示了 WA 算法收敛更快. 通过对比可以发现, 样本点的维数对 CGD 算法和 WA 算法收敛速度的影响很大, CGD 算法更适用于高维度样本.

鉴于 CGD 算法的迭代类型可以分为上升迭代、下降迭代和删除点迭代 3 类, 我们还研究了在不

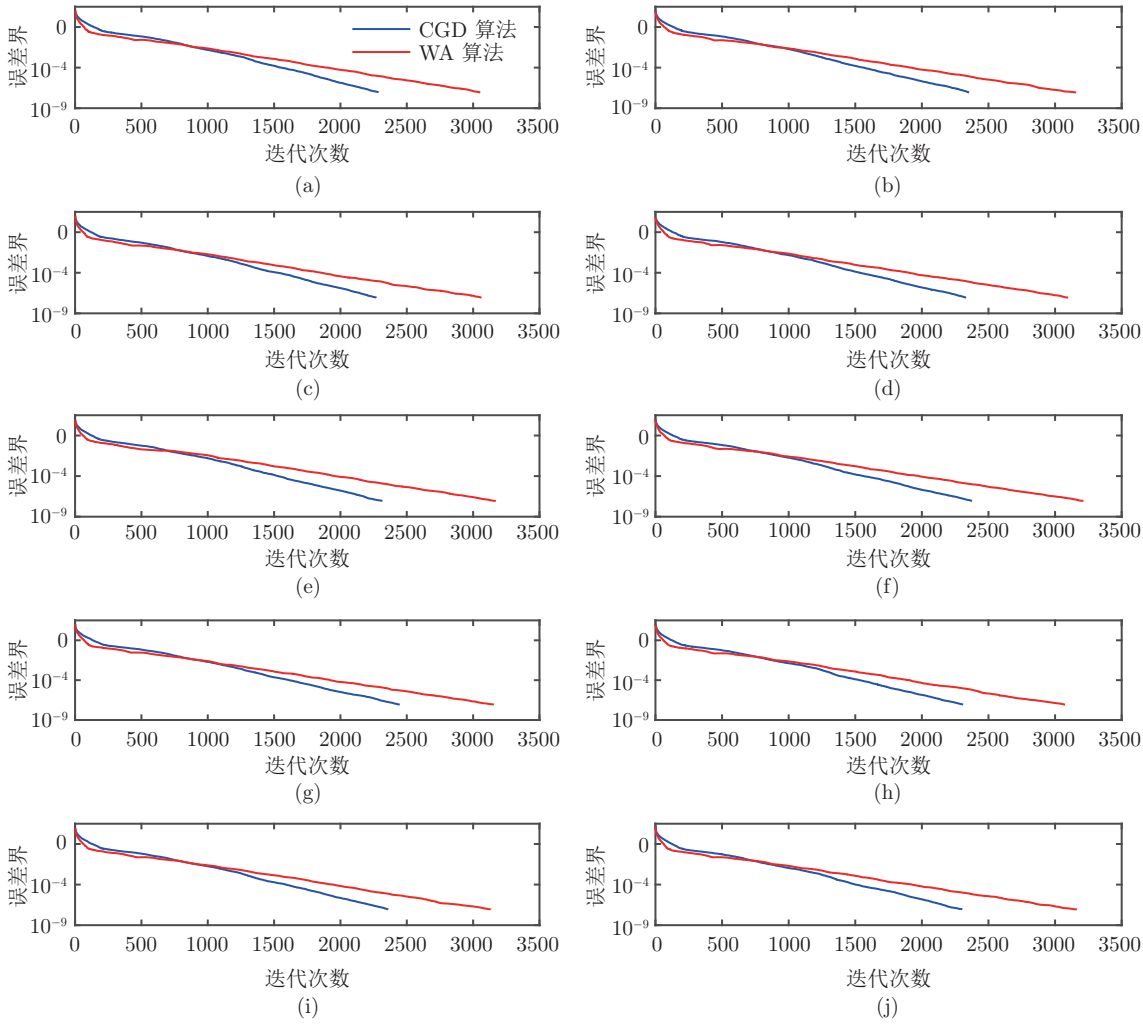

图 2 CGD 算法和 WA 算法在高维度低数量样本中收敛速度示意图

同类型迭代下目标函数值的下降量情形. 根据引理 4.1, 在上升迭代步中目标函数的下降量为

$$\Delta^+(\kappa_+) = h(u) - h(u^\dagger) = \ln(2\kappa_+ - n) - \ln(\kappa_+) + \frac{(n-\kappa_+)n}{\kappa_+^2},$$

其中 $\kappa_+ \geqslant n$. 在下降迭代步中目标函数的下降量为 $\Delta^-(\kappa_-) = h(u) - h(u^\dagger) = \ln\frac{\kappa_-}{n} + \frac{n}{\kappa_-} - 1$, 其中 $0 \leqslant \kappa_- \leqslant n$. 图 4 绘制了目标函数下降量与 κ 之间的函数关系: $\Delta^+(\kappa_+)$ 和 $\Delta^-(\kappa_-)$. 在图 4 中, 横轴表示 κ 的值, 纵轴表示目标函数 $h(u)$ 的下降量. 在图 4 中分别固定 n 为 $n=1$ (蓝色)、$n=2$ (绿色) 和 $n=3$ (红色). 实线显示的是上升迭代步情形下 $\Delta^+(\kappa_+)$ 与 κ_+ 之间的函数关系, 虚线显示的是下降迭代步情形下 $\Delta^-(\kappa_-)$ 与 κ_- 之间的函数关系. 从图 4 中可以看出:

(1) $\Delta^-(\kappa_-)$ 对于 κ_- 是单调非增的, 而 $\Delta^+(\kappa_+)$ 对于 κ_+ 是单调非减的, 这与在引理 4.2 中得到的结论是一致的;

(2) $\lim_{\kappa_+\to\infty}\Delta^+(\kappa_+) = \ln 2$;

(3) 当 $0 \leqslant \kappa_- \leqslant 0.3n$, $\Delta^-(\kappa_-) > \ln 2$, 即当 $0 \leqslant \kappa_- \leqslant 0.3n$ 时, $\Delta^-(\kappa_-)$ 大于 $\Delta^+(\kappa_+)$.

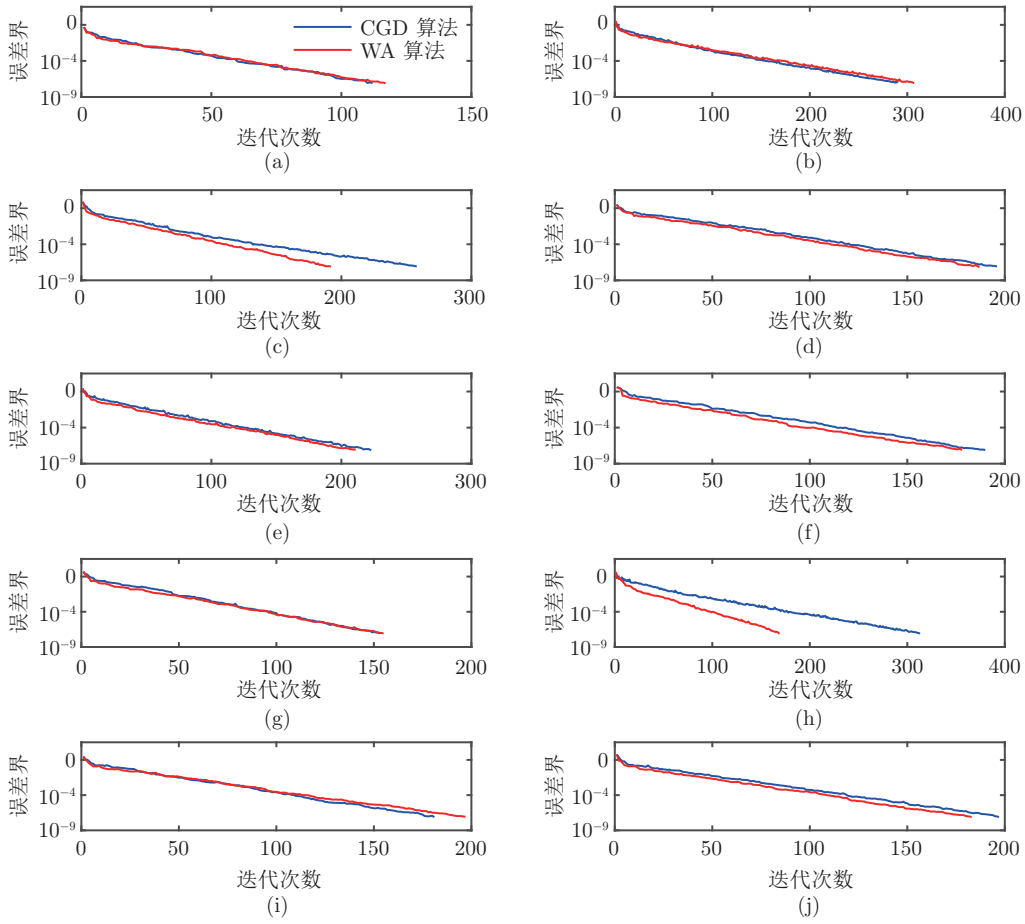

图 3　CGD 算法和 WA 算法在低维度高数量样本中收敛速度示意图

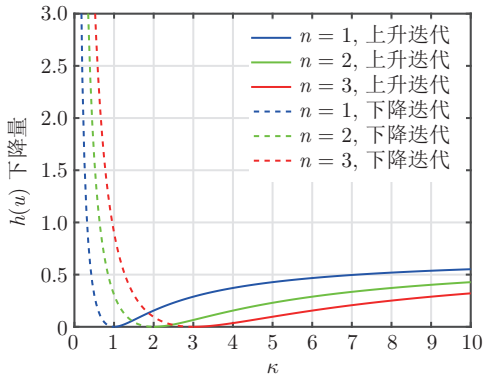

图 4　上升迭代和下降迭代情形下目标函数值下降量示意图

5.3　随机坐标轴下降算法

文献 [23] 指出, 坐标轴下降算法在每步选取坐标轴时需要计算目标函数梯度的所有分量, 因此, 当未知量维数较大时, 每步迭代的计算量也随之变大. 基于此, 文献 [23] 提出了随机坐标轴下降 (randomized coordinate descent, RCD) 算法. 本小节对比 RCD 算法与 CGD 算法在 MVEE 问题上的数值

效果. 需要注意的是, RCD 算法在每步迭代选取坐标轴时是基于以下准则: 光滑程度越差的坐标轴越有可能被选中. 然而, 由引理 3.1 可知, 模型 (2.3) 的目标函数 $h(u)$ 是对算法依坐标轴光滑的, 其光滑系数与 $\kappa(u)$ 在每个迭代点处的值有关. 因此, 在采用 RCD 算法选取坐标轴时, 依然需要在每步迭代计算出目标函数梯度的所有分量. 所以, 从每步迭代所耗费的计算量来说, RCD 算法并不比 CGD 算法少. 因此, 本小节主要对比 RCD 算法和 CGD 算法的收敛速度. 我们按照如下方式随机选择坐标轴: 在每步迭代 (假设当前迭代为第 k 次迭代) 中随机生成一个位于 1 至 m 之间的整数 i, 其中每个整数 i 被选取的概率为

$$\mathrm{Prob}_i = \kappa(u_k)_i \left(\sum_{j=1}^{m} \kappa(u_k)_i \right)^{-1}, \quad i = 1, \ldots, m.$$

根据文献 [23], 下面给出计算 MVEE 的 RCD 算法 (见算法 4).

算法 4 计算最小体积覆盖椭球的 RCD 算法

1: 输入: $\mathcal{X} = \{x_1, \ldots, x_m\}$, $\epsilon > 0$.
2: 第 0 步. 初始化: 采用 FW-KY 初始化策略设置 u_0.
3: 第 1 步. 若不收敛, 则执行循环.
4: 第 1.1 步. 随机产生 1 至 m 之间的整数 i.
5: 第 1.2 步. $u_{k+1} = P_{u \geqslant 0}(u_k + \frac{1}{L_i} \nabla h(u_k)_i)$.
6: 输出: $u^* = u_k$.

为了对比 CGD 算法和 RCD 算法的数值效果, 我们随机产生了两组样本, 分别为小规模样本 (样本维数为 10, 样本点数量为 500) 和中等规模样本 (样本维数为 30, 样本点数量为 1,800). 图 5 对比了

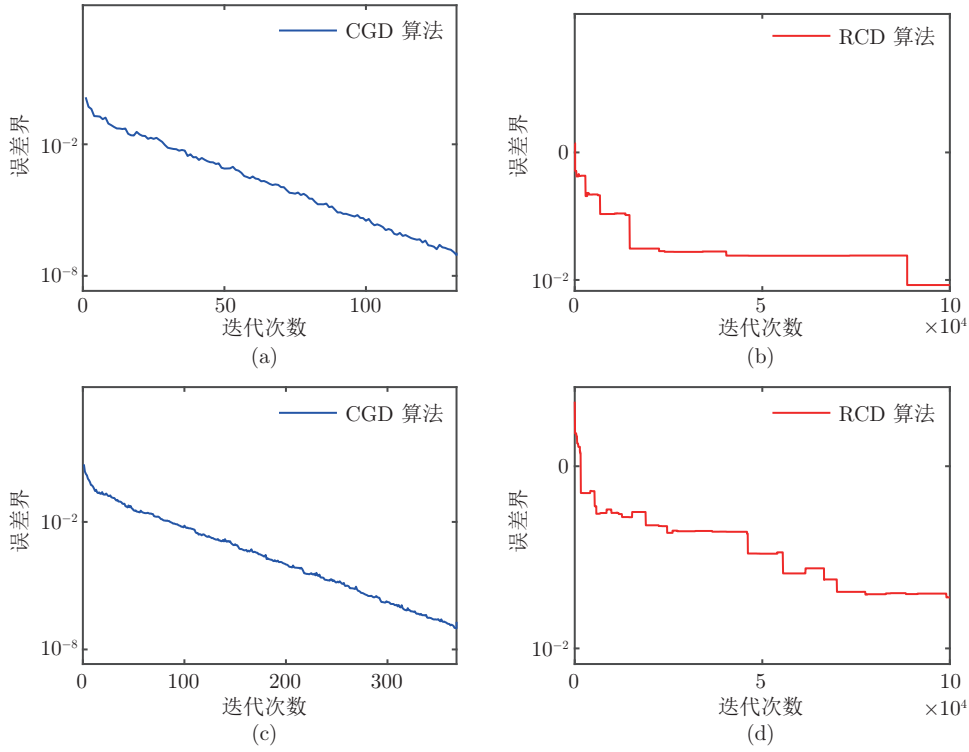

图 5 CGD 算法与 RCD 算法数值效果对比图. (a) 和 (b) 分别为 CGD 算法和 RCD 算法在小规模样本上的收敛性情形; (c) 和 (d) 分别为 CGD 算法和 RCD 算法在中等规模样本上的收敛性情形

CGD 算法与 RCD 算法的数值效果, 其中图 5(a) 和 5(b) 分别为 CGD 算法和 RCD 算法在小规模样本上的收敛性情形, 图 5(c) 和 5(d) 分别为 CGD 算法和 RCD 算法在中等规模样本上的收敛性情形, 横坐标表示迭代次数, 纵坐标表示以对数为尺度的误差界. 对于小规模样本, 图 5(a) 显示了 CGD 算法在大约 130 步后便收敛到 10^{-7}- 近似最优解, 而通过图 5(b) 可以看出 RCD 算法在迭代超过 1×10^4 次后只能收敛到 10^{-2}- 近似最优解. 对于中等规模样本, 图 5(c) 显示了 CGD 算法在大约 350 步后便收敛到 10^{-7}- 近似最优解, 而通过图 5(d) 可以看出 RCD 算法在迭代超过 1×10^4 次后只能收敛到 10^{-2}- 近似最优解. 由此可以看出, 在计算 MVEE 的问题上 CGD 算法比 RCD 算法更具有优势.

6 研究结论

本文提出了坐标轴下降算法用以计算空间点集的最小体积覆盖椭球, 采用 Gauss-Southwell 准则作为坐标轴选取方式. CGD 算法的计算效率与经典的 WA 算法计算效率相当, 具有全局次线性收敛性和局部线性收敛性. 值得一提的是, 在处理高维度数据集时, CGD 算法计算效率更高. 同时我们指出计算最小体积覆盖椭球模型的目标函数是对算法依坐标轴光滑的, 利用这个性质, FW 型算法的坐标轴选取方式可以纳入到统一框架中, 即 FW-K 算法按照目标函数在每个坐标轴上的光滑程度来选择坐标轴, WA 算法采用 Gauss-Southwell 坐标轴选取准则, 这也是 WA 算法比 FW-K 算法高效的原因之一. 最后, 我们通过数值实验对比了 CGD 算法与 RCD 算法, 测试结果说明, 在计算最小体积覆盖椭球问题上 CGD 算法效率更高.

关于今后在 MVEE 计算问题上的研究, 我们认为, 对偶削减 Newton 法由于每步迭代需要计算目标函数的 Hesse 矩阵导致内存不足, 因而无法处理超大规模数据集. 文献 [26] 提出了采用 Newton-Sketch 方法来解决存储空间不足的问题, 并且该方法具有超线性收敛速度. 因此将对偶削减 Newton 法与 Newton-Sketch 方法相结合有助于提升计算最小体积覆盖椭球的效率. 另外, 本文选取的步长是常数步长, 未来的研究可以考虑用其他的步长设置方法来提升 CGD 算法的计算效率, 如线搜索技术和 Barzilai-Borwein 方法 [27–29].

致谢 感谢三位审稿人提出的宝贵的建设性意见. 陶杰感谢卢晓敏博士给予的帮助和支持. 张威感谢 Singapore Maritime Institute 的经费支持.

参考文献

1 Kumar P, Yıldırım E A. A linearly convergent linear-time first-order algorithm for support vector classification with a core set result. INFORMS J Computing, 2011, 23: 377–391
2 Haraldsdóttir H S, Cousins B, Thiele I, et al. CHRR: Coordinate hit-and-run with rounding for uniform sampling of constraint-based models. Bioinformatics, 2017, 33: 1741–1743
3 Nikolov A, Talwar K, Zhang L. The geometry of differential privacy: The small database and approximate cases. SIAM J Comput, 2016, 45: 575–616
4 Li Y, Wang C, Shene C K. Extracting flow features via supervised streamline segmentation. Comput Graphics, 2015, 52: 79–92
5 Joshi S, Boyd S. Sensor selection via convex optimization. IEEE Trans Signal Process, 2009, 57: 451–462
6 Ahipaşaoğlu S D. Fast algorithms for the minimum volume estimator. J Global Optim, 2015, 62: 351–370
7 John F. Extremum problems with inequalities as subsidiary conditions. In: Giorgi G, Kjeldsen T, eds. Traces and Emergence of Nonlinear Programming. Basel: Birkhäuser, 2014, 197–215
8 Clarkson K L. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. ACM Trans Algorithms, 2010, 6: 1–30
9 Nesterov Y, Nemirovskii A. Interior-Point Polynomial Algorithms in Convex Programming. Philadelphia: SIAM, 1994

10 Vandenberghe L, Boyd S, Wu S P. Determinant maximization with linear matrix inequality constraints. SIAM J Matrix Anal Appl, 1998, 19: 499–533
11 Sun P, Freund R M. Computation of minimum-volume covering ellipsoids. Oper Res, 2004, 52: 690–706
12 Khachiyan L G. Rounding of polytopes in the real number model of computation. Math Oper Res, 1996, 21: 307–320
13 Kumar P, Yıldırım E A. Minimum-volume enclosing ellipsoids and core sets. J Optim Theory Appl, 2005, 126: 1–21
14 Yıldırım E A. On the minimum volume covering ellipsoid of ellipsoids. SIAM J Optim, 2006, 17: 621–641
15 Todd M J, Yıldırım E A. On Khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. Discrete Appl Math, 2007, 155: 1731–1744
16 Atwood C L. Sequences converging to D-optimal designs of experiments. Ann Statist, 1973, 1: 342–352
17 Ahipasaoglu S D, Sun P, Todd M J. Linear convergence of a modified Frank-Wolfe algorithm for computing minimum-volume enclosing ellipsoids. Optim Methods Softw, 2008, 23: 5–19
18 Beck A. First-Order Methods in Optimization. Philadelphia: SIAM, 2017
19 Todd M J. Minimum-Volume Ellipsoids: Theory and Algorithms. Philadelphia: SIAM, 2016
20 Boyd S, Vandenberghe L. Convex Optimization. Cambridge: Cambridge University Press, 2004
21 Cong W, Liu H, Ye F, et al. Rank-two update algorithms for the minimum volume enclosing ellipsoid problem. Comput Optim Appl, 2012, 51: 241–257
22 Cong W J, He L. Active-set algorithm for computing MVEE—Based on new initilization strategy (in Chinese). J Jilin Univ Sci, 2017, 55: 1141–1145 [丛伟杰, 何磊. 基于新的初始化策略计算 MVEE 的积极集算法. 吉林大学学报 (理学版), 2017, 55: 1141–1145]
23 Nesterov Y. Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM J Optim, 2012, 22: 341–362
24 Nutini J, Schmidt M, Laradji I. Coordinate descent converges faster with the Gauss-Southwell rule than random selection. In: Bach F, Blei D, eds. Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37. JMRL.org, 2015, 1632–1641
25 Ahipasaoglu S D. Solving ellipsoidal inclusion and optimal experimental design problems: Theory and algorithms. PhD Thesis. Ithaca: Cornell University, 2009
26 Pilanci M, Wainwright M J. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. SIAM J Optim, 2017, 27: 205–245
27 Dai Y H, Fletcher R. Projected Barzilai-Borwein methods for large-scale box-constrained quadratic programming. Numer Math, 2005, 100: 21–47
28 Dai Y H. A new analysis on the Barzilai-Borwein gradient method. J Oper Res Soc China, 2013, 1: 187–198
29 Tan C, Ma S, Dai Y H, et al. Barzilai-borwein step size for stochastic gradient descent. In: Lee D D, von Luxburg U, Garnett R, eds. Proceedings of the 30th International Conference on Neural Information Processing Systems. New York: Curran Associates Inc., 2016, 685–693

附录 A 关于定理 2.1 的一种新证明方法

假设从 $\{1,\ldots,m\}$ 中任意选取互不相同的 i 和 j 满足 $u_i^*>0$ 和 $u_j^*>0$. 定义向量 $u\in\Delta_m$ 为

$$u_k=\begin{cases}u_k^*, & k\notin\{i,j\},\\ u_i^*-\dfrac{u_i^*}{2}, & k=i,\\ u_j^*+\dfrac{u_i^*}{2}, & k=j.\end{cases}$$

令 $\kappa(u^*)_i=x_i^{\mathrm{T}}(XU^*X^{\mathrm{T}})^{-1}x_i$, $i=1,\ldots,m$. 由于 $\nabla g(u^*)=-(\kappa(u^*)_1,\ldots,\kappa(u^*)_m)$, 因此根据凸优化问题最优性条件 $\nabla g(u^*)^{\mathrm{T}}(u-u^*)\geqslant 0$ 可得

$$\frac{u_i^*}{2}\kappa(u^*)_i-\frac{u_i^*}{2}\kappa(u^*)_j\geqslant 0. \tag{A.1}$$

因为 $u_i^*>0$, 所以有 $\kappa(u^*)_i\geqslant\kappa(u^*)_j$. 调换 i 和 j 的位置, 可得 $\kappa(u^*)_j\geqslant\kappa(u^*)_i$, 进而可得 $\kappa(u^*)_i=\kappa(u^*)_j$. 因此, $\nabla g(u^*)$ 中对应 u^* 中大于 0 的分量具有相同的值 (不妨设为 t), 再由 (A.1) 可知剩下的

对应 u^* 中等于 0 的分量都小于 t. 于是存在 $t \in \mathbb{R}$ 使得

$$\begin{aligned} \kappa(u^*)_i = t, \quad u_i^* > 0, \\ \kappa(u^*)_i \leqslant t, \quad u_i^* = 0. \end{aligned}$$

另外, 由于 $e^{\mathrm{T}}u^* = 1$, 因此有 $\sum_{i=1}^m u_i^* \kappa(u^*)_i = t$, 即

$$t = \left(\sum_{i=1}^m u_i^* \mathrm{tr}(\kappa(u^*)_i) \right) = \mathrm{tr}(XU^*X^{\mathrm{T}}(XU^*X^{\mathrm{T}})^{-1}) = n.$$

A coordinate gradient descent algorithm for computing the minimum volume enclosing ellipsoid

Jie Tao, Wei Zhang, Chao Lu & Mark Goh

Abstract The minimum volume enclosing ellipsoid (MVEE) is a basic convex optimization problem. This paper describes some new properties such as "algorithmic coordinate-wise smoothness" of this model and proposes a steepest descent type algorithm, the coordinate gradient descent (CGD) algorithm, to address the MVEE problem. We prove that the CGD algorithm is sublinearly convergent. Moreover, it is shown to converge in a linear rate locally, and slightly faster than the FW type algorithms, especially in the cases of large dimensions. At last, we compare our algorithm with the random coordinate descent (RCD) method and show that the RCD algorithm is less efficient than the CGD algorithm in computing the MVEE. The numerical tests support our theoretical results.

Keywords **minimum volume enclosing ellipsoid, first-order algorithm, coordinate descent**

MSC(2020) **90C30**

doi: 10.1360/SCM-2018-0795