日本オペレーションズ・リサーチ学会
2023 年春季研究発表会

射影 - 再スケーリング法を用いた 対称錐計画問題に対する後処理アルゴリズム

05001390　筑波大学 システム情報工学研究群　*加納伸一　KANO Shin-ichi
01703540　筑波大学 システム情報系　吉瀬章子　YOSHISE Akiko

1. はじめに

対称錐計画は, 対称錐 $\mathcal{K} \subseteq \mathbb{R}^d$ とアフィン空間からなる実行可能領域上で線形の目的関数を最小化または最大化する問題である.

$$\begin{array}{ll} \text{(P)} & \inf \langle c, x\rangle \quad \text{s.t. } \mathcal{A}x = b,\ x \in \mathcal{K}. \\ \text{(D)} & \sup\ b^\top y \quad \text{s.t. } c - \mathcal{A}^* y \in \mathcal{K}^*. \end{array}$$

ここで, $\langle , \rangle$ は内積, $\mathcal{A}$ は $\mathbb{R}^d$ から $\mathbb{R}^m$ への線形写像, $\mathcal{A}^*$ は $\mathcal{A}$ の随伴作用素, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^d$ である. また, $\mathcal{K}^*$ は $\mathcal{K}$ の双対錐であり, $\mathcal{K}^* = \{s \in \mathbb{R}^d : \langle x, s\rangle \geq 0, \forall x \in \mathcal{K}\}$ である. 対称錐は自己双対性が成り立つため, $\mathcal{K} = \mathcal{K}^*$ であることに注意する. 対称錐計画は線形計画, 2 次錐計画, 半正定値計画を含み, 主双対内点法で効率よく求解できると知られている. 内点法は, 現在点から更新方向を計算し, 逐次的に最適解に近づけていく手法であるが, 現在点が最適解に近づくと更新方向の計算が数値的に不安定になるため, 高精度の解を求めることが難しいという実装面での課題がある.

2. 射影 - 再スケーリング法

[1] は, 同次線形計画問題の内点許容解を求める新しいアルゴリズムを提案した. このアルゴリズムは, 線形部分空間への射影とスケーリングで構成され, [2, 3, 4] により, 対称錐計画へ拡張された. [2, 3, 4] の手法が扱う問題は以下の通りである.

$$\text{find } x \in \mathbb{R}^d \text{ s.t. } x \in \text{int}\mathcal{K} \cap \mathcal{L}. \qquad (1)$$

ここで, $\mathcal{L} \subseteq \mathbb{R}^d$ は線形部分空間であり, int$\mathcal{K}$ は対称錐 $\mathcal{K}$ の内部を表す. 射影 - 再スケーリング法は, 1. int$\mathcal{K}\cap\mathcal{L}$ の点を探す, 2. (1) と等価な問題の作成をする, という 2 つのステップを繰り返す. ステップ 1 では, $\mathcal{L}$ と $\mathcal{K}$ を入力とする内部関数を使用する. この内部関数は $\mathcal{L}$ への射影を用いて, int$\mathcal{K}\cap\mathcal{L}$ の点, int$\mathcal{K} \cap \mathcal{L} = \emptyset$ であることを示す点 (つまり, $\mathcal{K} \cap \mathcal{L}^\perp$ を満たす非ゼロの点. ただし, $\mathcal{L}^\perp$ は $\mathcal{L}$ の直交補空間), または int$\mathcal{K} \cap \mathcal{L}$ の存在範囲を狭めるカットのいずれかを返す. もし, ステップ 1 でカットが見つかった場合は, ステップ 2 で, int$\mathcal{K} \cap \mathcal{L}$ の存在範囲が, より錐 $\mathcal{K}$ の中心に移るようなスケーリングを施し, 新たな線形部分空間 $\bar{\mathcal{L}}$ を作成する. ここで, (1) の $\mathcal{L}$ を $\bar{\mathcal{L}}$ に置き換えた問題は, (1) と等価な問題であることに注意する. そして, $\bar{\mathcal{L}}$ に対して再びステップ 1 を行う操作を繰り返す.

[4] では, [4] の射影-再スケーリング法が [2, 3] のものよりも高速であることを理論的に証明し, 計算機実験によりそのことを確かめた. また, int$\mathcal{K}\cap\mathcal{L}$ を満たす点が錐の境界付近にしか存在しない問題に対し, 射影-再スケーリング法は商用ソルバー (Mosek [5]) よりも実行時間がかかるものの, 線形制約の残差が小さい解が得られやすかったという実験結果を報告した.

3. 後処理アルゴリズム

[4] より, 射影-再スケーリング法を用いることで, 内点法 (ソルバー) が出力した近似最適解をより高精度の解へ更新できると思われる. 同手法だけでも問題を解くことはできるが, 実行時間の観点から, 本研究では後処理としての使用を提案する.

3.1. 基盤となる理論

射影-再スケーリング法を用いて (P) または (D) を解く時, まず任意の実数 $\theta \in \mathbb{R}$ に対し, 以下の拡大係数行列 $\bar{\mathcal{A}}$ を考える.

$$\bar{\mathcal{A}} = \begin{pmatrix} \mathcal{A} & -b & 0 \\ c^\top & -\theta & 1 \end{pmatrix}$$

$\bar{\mathcal{A}}$ の零空間を ker$\bar{\mathcal{A}}$ とする. ker$\bar{\mathcal{A}}$ の直交補空間は, $\bar{\mathcal{A}}$ の随伴作用素 $\bar{\mathcal{A}}^*$ の像空間 range$\bar{\mathcal{A}}^*$ で与えられる. ここで, 対称錐 $\bar{\mathcal{K}} = \mathcal{K} \times \mathbb{R}^2_+$ と線形部分空間 $\mathcal{L} = \text{ker}\bar{\mathcal{A}}$ を用いた問題 (2) を作成し, 射影-再スケーリング法で解くことを考える.

$$\text{find } (x, \tau, \omega) \in \mathbb{R}^{d+2} \text{ s.t. } x \in \text{int}\bar{\mathcal{K}} \cap \mathcal{L}. \qquad (2)$$

実は, θ に最適値 (最適値に十分近い値) を入力すれば, 射影-再スケーリング法は (P) または (D) の (近似的な) 最適解を出力できる. 仮に, 問題 (2) の実行可能解 (x, τ, ω) が得られたとする. すると, $\bar{x} = x/\tau$ は $\langle c, \bar{x}\rangle + \omega/\tau = \theta$ を満たす, つまり $\langle c, \bar{x}\rangle < \theta$ となる (P) の内点解となる. 問題 (2) が実行不可能の時は, $\bar{\mathcal{K}} \cap \mathcal{L}^{\perp}$ を満たす非ゼロの点, つまり以下を満たす $(y, \kappa) \in \mathbb{R}^{m+1}$ が得られる.

$$\kappa c - \mathcal{A}^*(-y) \in \mathcal{K},\ b^\top(-y) \geq \kappa\theta,\ \kappa \geq 0.$$

$\kappa > 0$ ならば, $\bar{y} = -y/\kappa$ は $b^\top \bar{y} \geq \theta$ を満たす (D) の解となり, $\kappa = 0$ の時, $b^\top(-y) > 0$ ならば (P) の実行不可能性, $b^\top(-y) = 0$ ならば (P) の内点解の非存在性がそれぞれ証明可能である. さらに, $\mathcal{L} = \mathrm{range}\bar{\mathcal{A}}^*$ として問題 (2) を考えれば, 同様にして $b^\top y > \theta$ を満たす (D) の内点解, $\langle c, x\rangle \leq \theta$ を満たす (P) の解, または (D) の実行不可能性 / 内点解の非存在性を導くことができる. (以下, $\mathcal{L} = \ker\bar{\mathcal{A}}$ の (2) を解く射影-再スケーリング法を P-Alg, $\mathcal{L} = \mathrm{range}\bar{\mathcal{A}}^*$ の時の同手法を D-Alg と呼ぶ.)

3.2. アルゴリズムの流れ

提案手法の説明を簡潔にするため, (P) と (D) に内点解が存在し最適値を θ^* と仮定する. 入力値 θ に対し, P-Alg と D-Alg の出力を P-Alg(θ), D-Alg(θ) で表すとすると, 出力は {(P) の解, (D) の解 } のどちらかとなる. 内点法の出力解を用いて最初の入力値 θ を決定した後, 提案手法の大まかな流れは以下の通りである. $\varepsilon > 0$ は十分小さい値とする.

1. 以下を満たす UB_d, LB_d が得られるまで, θ を調整しながら, 繰り返し D-Alg を行う.

   $\text{D-Alg}(UB_d) = \text{(P) の解},\ UB_d - LB_d \leq \varepsilon,$
   $\text{D-Alg}(LB_d) = \text{(D) の解}.$

2. 以下を満たす UB_p, LB_p が得られるまで, θ を調整しながら, 繰り返し P-Alg を行う.

   $\text{P-Alg}(UB_p) = \text{(P) の解},\ UB_p - LB_p \leq \varepsilon,$
   $\text{P-Alg}(LB_p) = \text{(D) の解}.$

3. (P) と (D) の解を返す.

ここで射影-再スケーリング法を実装した場合, 計算機が返すのは, (2) の近似解に過ぎないことに注意する. 例えば, $\|\mathcal{A}x - b\|_2 > 0$ となる解や僅かに錐制約を違反する解, 入力値 θ よりも目的関数値が改善されていない解が返ってくることがある.

よって, 本研究ではこれらのことを考慮してアルゴリズムを構築した. 具体的には, ステップ 1 と 2 で (D) の解が得られた場合は暫定解として保持し, (D) の近似解が得られた場合は, 暫定解とアフィン結合を考えることで解を更新している. また, 計算機上で $\mathcal{A}x = b$ を厳密に満たす x が得られることはまず無いので, ステップ 3 では, これまでに得られた (P) の近似解を用いて 1 つの (P) の近似解を作成し, 出力するようにしている.

4. おわりに

本発表では SDPLIB [6] の問題に対し, Mosek の出力解に後処理を適用した実験結果を紹介する. 詳細な結果は当日説明するが, 後処理アルゴリズムにより, 最適解としての精度を測る DIMACS error [7] の値を大きく改善することができた.

参考文献

[1] Chubanov, S. (2015). A polynomial projection algorithm for linear feasibility problems. Mathematical Programming, 153(2), 687-713.

[2] Pena, J., & Soheili, N. (2017). Solving conic systems via projection and rescaling. Mathematical Programming, 166(1-2), 87-111.

[3] Lourenço, B. F., Kitahara, T., Muramatsu, M., & Tsuchiya, T. (2019). An extension of Chubanov's algorithm to symmetric cones. Mathematical Programming, 173(1-2), 117-149.

[4] Kanoh, S. I., & Yoshise, A. (2021). A New Extension of Chubanov's Method to Symmetric Cones. arXiv preprint arXiv:2110.09854.

[5] MOSEK, A. Moset optimization toolbox for MATLAB (2019). Release, 9, 98.

[6] Borchers, B. (1999). SDPLIB 1.2, a library of semidefinite programming test problems. Optimization Methods and Software, 11(1-4), 683-690.

[7] Mittelmann, H. D. (2003). An independent benchmarking of SDP and SOCP solvers. Mathematical Programming, 95(2), 407-430.