数理解析研究所講究録
1114巻 1999年 172-183

半正定値計画問題での大規模線形方程式系に対する前処理付き共役勾配法

東京大学 工学系研究科　中田 和秀 (Kazuhide Nakata)
東京大学 工学系研究科　張 紹良 (Shao-Liang Zhang)
東京工業大学 情報理工学研究科　小島 政和 (Masakazu Kojima)

1 はじめに

近年、半正定値計画問題（Semidefinite Programming, SDP と略）に対して、その解法である主双対内点法に関わる理論的な解析、組合せ最適化問題への応用等の様々な研究が行われている。また、著者等のグループが公開している SDPA (SemiDefinite Programming Algorithm)[4] をはじめ、SDP を主双対内点法により解くソフトウェアが幾つか開発され、それをもとにした数値実験等の研究が盛んに行われている。SDP を線形計画問題の対称行列の空間への拡張として捉えると、SDP の主双対内点法は線形計画問題に対する主双対内点法の拡張とみなすことが可能である。しかし、大規模な SDP 問題を解く場合、同規模の線形計画問題に比べ主双対内点法の各反復で求める探索方向の計算が非常に困難となる。そのため、大規模な問題によく現れる疎性を利用した計算法が提案され、探索方向の計算の効率化に大きな成果をあげている [5]。しかし、線形計画問題と違い SDP ではたとえ制約行列が疎であっても、一般に解く必要のある線形方程式系の係数行列は密になる。SDPA を含む多くの SDP を主双対内点法により解くソフトウェアでは、その線形方程式系の解法としてコレスキー分解や修正コレスキー分解等の直接法を使用している。よって、SDP の主問題の制約式数が非常に大きな問題では線形方程式系のサイズが大きくなるため、これらの行列分解にかかる計算時間や係数行列を保持するメモリ量が極めて大きくなるため最適解の計算が困難となり、これが実用上大きな制限となっている。本研究では、この問題点を克服するため、線形方程式系に対し共役勾配法を適用し、さらに計算効率を高める種々の工夫を組み入れたソフトウエアの開発を行った。SDP を主双対内点法で解く際に現れる線形方程式系の係数行列は一般に疎ではないが、この係数行列を保持せずに計算を行うことにより、従来に比べメモリ使用量を大幅に減らすことが可能となる。一方、共役勾配法において計算時間は反復回数に大きく依存しているが、数値実験により高精度の近似解を生成するにつれて、この反復回数は増大することを示す。このとき、共役勾配法の反復回数を減らし高精度の解を計算するには、前処理を行い収束性を高める必要がある。そのため、線形方程式系の係数行列の構造に基づき、前処理の統一的な枠組みを提案する。最後に、線形方程式系に共役勾配法を適用したソフトウェアにより大規模なグラフの最大クリーク問題の SDP 緩和を解き、これらの方法の有効性を検証する。

2 半正定値計画問題 (Semidefinite Programming)

$\Re^{n\times n}$ を $n\times n$ の実行列の集合、$\mathcal{S}^n$ を $n\times n$ の実対称行列の集合とする。任意の $\boldsymbol{X}, \boldsymbol{Z} \in \Re^{n\times n}$ に対して、$\boldsymbol{X}\bullet\boldsymbol{Z}$ は $\boldsymbol{X}$ と $\boldsymbol{Z}$ の内積、すなわち、Tr $\boldsymbol{X}^T\boldsymbol{Z}$（$\boldsymbol{X}^T\boldsymbol{Z}$のトレース）を表す。$\boldsymbol{X}\succ\boldsymbol{O}$ は$\boldsymbol{X}\in\mathcal{S}^n$ が正定値、つまりゼロでない任意の $\boldsymbol{u}\in\Re^n$に対し$\boldsymbol{u}^T\boldsymbol{X}\boldsymbol{u}>0$ であることを示す。$\boldsymbol{X}\succeq\boldsymbol{O}$ は$\boldsymbol{X}\in\mathcal{S}^n$ が半正定値、つまり任意の $\boldsymbol{u}\in\Re^n$ に対し $\boldsymbol{u}^T\boldsymbol{X}\boldsymbol{u}\geq 0$ であることを示す。

$\boldsymbol{C}\in\mathcal{S}^n$, $\boldsymbol{A}_i\in\mathcal{S}^n$ $(1\leq i\leq m)$, $b_i\in\Re$ $(1\leq i\leq m)$ とする。このとき、半正定値計画問題 (Semidefinite Programming, SDP と略) の主問題と双対問題は以下のように与えられる。

$$
\left.
\begin{array}{lll}
\textbf{主問題：} & \text{最小化} & \boldsymbol{C} \bullet \boldsymbol{X} \\
& \text{制約条件} & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i \ (1 \leq i \leq m),\ \boldsymbol{X} \succeq \boldsymbol{O}. \\
\textbf{双対問題：} & \text{最大化} & \displaystyle\sum_{i=1}^{m} b_i y_i \\
& \text{制約条件} & \displaystyle\sum_{i=1}^{m} \boldsymbol{A}_i y_i + \boldsymbol{Z} = \boldsymbol{C},\ \boldsymbol{Z} \succeq \boldsymbol{O}.
\end{array}
\right\} \tag{1}
$$

2.1 主双対内点法

SDP(1) に対する解法として、主双対内点法がよく知られている。以下に一般的な SDP に対する主双対内点法の概要を示す。

一般的な主双対内点法：

Step 0: $\boldsymbol{X} \succ \boldsymbol{O}$ と $\boldsymbol{Z} \succ \boldsymbol{O}$ を満たすように初期点 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ を決める。

Step 1: もし、現在の反復点 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ が終了条件を満たしていたならば、反復を終了する。

Step 2: 探索方向パラメータ μ を定め、探索方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ を計算する。

Step 3: 主問題のステップ長 α_p と双対問題のステップ長 α_d を定め

$$
\boldsymbol{X} = \boldsymbol{X} + \alpha_p d\boldsymbol{X} \succ \boldsymbol{O},\quad \boldsymbol{y} = \boldsymbol{y} + \alpha_d d\boldsymbol{y},\quad \boldsymbol{Z} = \boldsymbol{Z} + \alpha_d d\boldsymbol{Z} \succ \boldsymbol{O}.
$$

とする。

Step 4: Step 1 に戻る。

SDP の主双対内点法の詳細は 小島 [8], Todd [12], Vandenberghe and Boyd [14] などを参照されたい。ここでは、筆者らのグループが開発した SDPA での実装をもとに、本論で以下議論を行う計算部分についてのみ説明をする。主双対内点法の Step 2 で計算する探索方向として様々な方向が提案されているが、SDPA ではその中でも実用上よく使われている HRVW/KSH/M 方向を実装している。このとき、探索方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ の計算過程で、大きさ m の線型方程式系

$$
\boldsymbol{B} d\boldsymbol{y} = \boldsymbol{s} \tag{2}
$$

を解くことになる。ただし、係数行列$\boldsymbol{B}$の各要素は

$$
B_{ij} = \boldsymbol{A}_i \bullet \boldsymbol{X} \boldsymbol{A}_j \boldsymbol{Z}^{-1} \qquad (1 \leq i \leq m,\ 1 \leq j \leq m) \tag{3}
$$

で与えられる。このとき$\boldsymbol{X}, \boldsymbol{Z}^{-1}$は正定値行列であるので、係数行列$\boldsymbol{B}$は正定値行列となることに注意する。式 (3) では行列同士の積と内積により表現しているが、4 節で利用するためこれをベクトルとクロネッカー積の形でも表現しておく。任意の行列 $\boldsymbol{K} \in \Re^{n \times n}$ に対し、$\mathbf{vec}(\boldsymbol{K})$ を $\mathbf{vec}(\boldsymbol{K}) \equiv (K_{11}, K_{12}, \cdots, K_{nn})^T \in \Re^{n^2}$ と定義する。また、行列$\boldsymbol{M}, \boldsymbol{N} \in \Re^{n \times n}$のクロネッカー積$\boldsymbol{M} \otimes \boldsymbol{N}$は

大きさ $n^2 \times n^2$の行列であり、任意の行列$\boldsymbol{L} \in \Re^{n \times n}$に対し $(\boldsymbol{M} \otimes \boldsymbol{N})\mathbf{vec}(\boldsymbol{L}) = \mathbf{vec}(\boldsymbol{N}\boldsymbol{L}\boldsymbol{M}^T)$ を満たすものとする。このとき、係数行列$\boldsymbol{B}$の各要素は

$$\begin{aligned} B_{ij} &= \boldsymbol{A}_i \bullet \boldsymbol{X}\boldsymbol{A}_j\boldsymbol{Z}^{-1} \\ &= \mathbf{vec}(\boldsymbol{A}_i)^T\mathbf{vec}(\boldsymbol{X}\boldsymbol{A}_j\boldsymbol{Z}^{-1}) \\ &= \mathbf{vec}(\boldsymbol{A}_i)^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathbf{vec}(\boldsymbol{A}_j) \end{aligned} \tag{4}$$

と表現できる。また、$\mathcal{A}$ を $\mathcal{A} \equiv (\mathbf{vec}(\boldsymbol{A}_1), \mathbf{vec}(\boldsymbol{A}_2), \cdots, \mathbf{vec}(\boldsymbol{A}_m)) \in \Re^{n^2 \times m}$ と定義すると、係数行列$\boldsymbol{B}$は以下のように定式化される。

$$\boldsymbol{B} = \mathcal{A}^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathcal{A} \tag{5}$$

2.2 既存のソフトウェア

SDP を解くソフトウェアは、SDPA をはじめ SDPpack[1], CSDP[2], SDPHA[10], SeDuMi[11], SDPT3[13], SP[15] など幾つか存在している。これらのソフトウェアは、細部に違いはあるものの概ね前節で述べた主双対内点法を実装しており、探索方向の計算過程で大きさ m 程度の線形方程式系を解く必要がある。このとき、これらのソフトウェアでは線形方程式系の解法としてコレスキー分解や修正コレスキー分解等の直接法を使用しているため、SDP(1) の主問題の制約式の数である m が極めて大きくなる問題については解くことが困難であることが指摘されている [3]。表 1 はいくつかの疎で大規模な問題を SDPA によって解いたときの、計算時間とメモリ消費量を表している。2.1 節で述べたように、SDPA では探索方向の計算過程で大きさ m の線形方程式系 (2) を解く。この問題では、1 つを除いた制約行列の非ゼロ要素は 2 個であり、双対残差が 0.1 以下になるまで SDPA の反復を繰り返した。

表 1: 疎で大規模な問題を SDPA によって解いたときの計算時間と使用メモリ

n	100	100	100	100
m	539	1024	2029	3992
$\boldsymbol{B}\boldsymbol{dy} = \boldsymbol{s}$の計算時間	5.3 秒	72.3 秒	631.7 秒	6278.9 秒
その他の部分の計算時間	8.2 秒	16.2 秒	44.3 秒	209.4 秒
総計算時間	13.6 秒	88.5 秒	676.0 秒	6488.3 秒
係数行列$\boldsymbol{B}$の使用メモリ	2.2MB	8.0MB	31.4MB	121.6MB
その他の使用メモリ	6.3MB	7.1MB	9.9MB	20.4MB
総使用メモリ	8.5MB	15.1MB	41.3MB	142.0MB

この表より、計算時間の大部分は線形方程式系 $\boldsymbol{B}\boldsymbol{dy} = \boldsymbol{s}$ の計算に費やされていることが分かる。その部分の計算時間は、m が大きくなるにつれて非常に増大しており、それにともない総計算時間も増大している。また、使用メモリの大部分は係数行列 $\boldsymbol{B}$ の確保分に費やされている。そして、m が大きくなるにつれて、その部分の使用メモリは著しく増大しており、それにともない使用メモリも増大している。つまり、m が大きくなるにつれ、計算時間やメモリ使用量の観点から解くことが非常に困難になっている。

3 共役勾配法

本研究では、既存のソフトウェアでは解くことが困難である n に対し m が大きな SDP を解くため、SDPA を基礎として SDPA の各反復で解く線形方程式系 (2) に反復解法である共役勾配法を適用したソフトウェア SDPA-CG の開発を行った。

3.1 共役勾配法の実装

共役勾配法は、Hestenes and Stiefel [7] が提案した方法で、係数行列が対称でかつ正定値である線形方程式系に対する反復解法である。詳しくは、Golub and Van Loan[6] 等を参照。SDPA では、探索方向を HRVW/KSH/M 方向とした主双対内点法を実装しており、線形方程式系 (2) の係数行列 $\boldsymbol{B}$は対称で正定値であり、共役勾配法を適用することが可能である。

SDPA-CG の開発にあたり、共役勾配法の終了条件や実行可能性の補正の方法を提案した。また、より効率的に計算できるような計算方法を提案し実装を行った。SDPA-CG の詳細については中田, 藤沢, 小島 [9] を参照されたい。共役勾配法を効率よく実装した結果、SDPA-CG で線形方程式系の解を計算するのに必要な計算量は

$$6n^3 + \mathcal{O}(m+n^2) \text{ flop} \times \text{共役勾配法の反復回数}$$

となり、メモリ使用量は $n \times n$ の行列数個分となっている。

3.2 SDPA と SDPA-CG

まず最初に SDPA と SDPA-CG のメモリ使用量と計算量の理論的な比較を行い、次に数値実験により SDPA-CG の特徴を考察する。まずメモリ使用量の点では、SDPA-CG は SDPA と違い $m \times m$ の係数行列 $\boldsymbol{B}$ を保持せず、$n \times n$ の行列の保持により計算を行うことが出来る。よって SDPA-CG は、n に対し m が大きな SDP を解く場合メモリ使用量の点で非常に有利である。次に計算時間の観点から、SDPA-CG と SDPA を比較する。3.1 節で述べたように、共役勾配法の計算時間はその反復回数に比例する。つまり、少ない反復回数で線形方程式系の近似解を得ることが出来るならば、SDPA より高速に近似探索方向が求まることが予想される。幾つかの実験の結果、主双対内点法の反復が進み最適解に近づくに従い、共役勾配法の反復回数は指数的に増大する傾向があり、それに比例して計算時間も増大することがわかった。図 1 のグラフは、ある疎で大規模な問題を解いたときの実験結果である。縦軸に SDPA や SDPA-CG の各主反復に関しての双対残差の対数, 横軸に累積計算時間を取っている。図 1 のグラフでは、SDPA においては双対残差の対数と累積計算時間はほぼ比例しているが、SDPA-CG では双対残差の対数に対して累積計算時間は指数的に増えている。つまり SDPA ではその 1 主反復で双対残差はほぼ一定の割合で減り、計算時間もほぼ同じであるが、SDPA-CG では 1 主反復で双対残差はほぼ一定の割合で減るものの、各主反復での共役勾配法の反復回数は双対残差が小さくなるにつれて増えており、それにともない各反復での計算時間も増大していることが分かる。よって SDPA-CG は SDPA に比べ、双対残差がある程度大きな精度の悪い近似最適解を高速に求めることができる。一方、SDPA-CG で高精度の近似最適解を求めることは、計算時間の点から困難であるといえる。

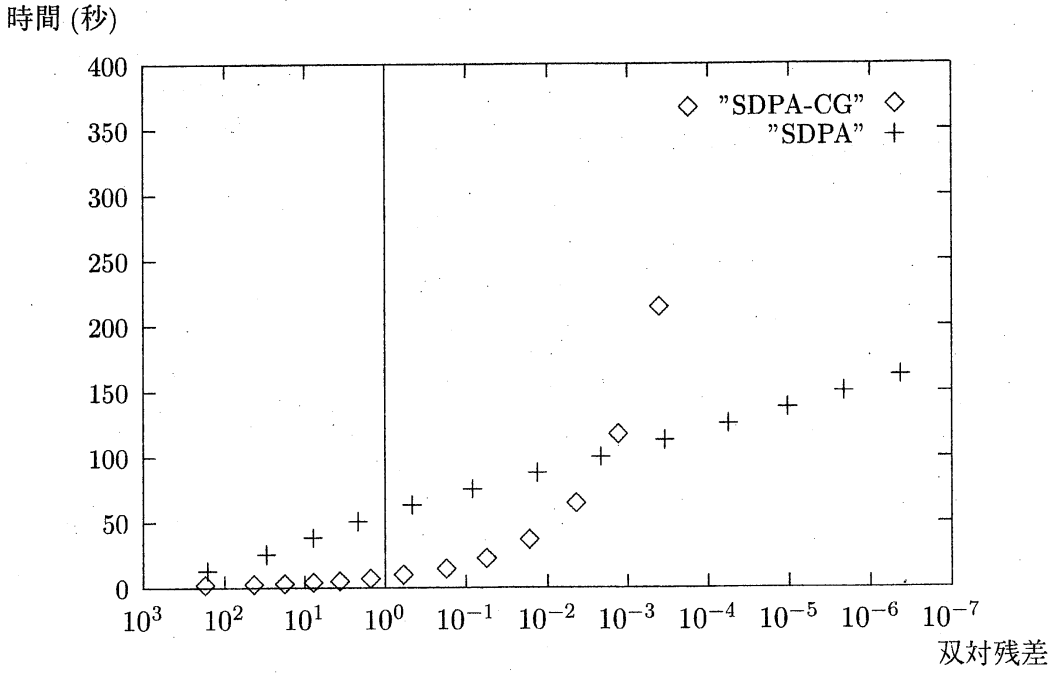

図 1: SDPA と SDPA-CG の双対残差 - 累積計算 グラフ

4 前処理つき共役勾配法

SDPA-CG でより厳密な最適解を求めるためには、共役勾配法の収束性を高める必要がある。一般に共役勾配法の収束性は線形方程式系の係数行列の固有値に依存するため、以下で説明する前処理付き共役勾配法が反復回数を減らすのに有効である。これは、ある正則行列$\boldsymbol{M}$を前処理行列とし、線形方程式系$\boldsymbol{B}\boldsymbol{dy} = \boldsymbol{b}$を解く代わりに $(\boldsymbol{M}^{-T}\boldsymbol{B}\boldsymbol{M}^{-1})(\boldsymbol{M}\boldsymbol{dy}) = (\boldsymbol{M}^{-T}\boldsymbol{b})$ という線形方程式を解くことに相当する。このとき行列 $\boldsymbol{M}^{-T}\boldsymbol{B}\boldsymbol{M}^{-1}$ が単位行列に近似しておれば、共役勾配法の収束が速くなる。ただし、$\boldsymbol{M}$はその行列を係数とする線形方程式系$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{h}$の解が高速に計算出来るものでなくてはならない。詳しくは、Golub and Van Loan[6] 等を参照。共役勾配法の前処理として、対角スケーリングや不完全コレスキー分解など、これまで様々なものが提案されてきた。しかし、すべての線形方程式系に有効である前処理行列は存在せず、個々の問題の特性に合わせた前処理行列を用意する必要がある。そのため、線形方程式系 (2) の$\boldsymbol{B}\boldsymbol{dy} = \boldsymbol{s}$に対し、係数行列$\boldsymbol{B}$の構造を解析することにより、この線形方程式系に対する前処理の統一的な枠組みを提案する。これは、制約行列$\boldsymbol{A}_i(i = 1, 2, \cdots, m)$ の集まりである $\mathcal{A}$ に対し、ある種の不完全な QR 分解を行うものと捉えることができ、この枠組みを Gram-Schmidt の直交化法から説明する。

4.1 Gram-Schmidt の直交化法

Gram-Schmidt の直交化法とは、複数のベクトルを正規直交化する方法である。このとき本論では、一般化したベクトルの内積による正規直交化を想定する。つまり、ある正定値行列$\boldsymbol{H}$を定め、ベクトルの内積を $(\boldsymbol{a}, \boldsymbol{b})_{\boldsymbol{H}} \equiv \boldsymbol{a}^T\boldsymbol{H}\boldsymbol{b}$と定義する。以下のアルゴリズムは修正 Gram-Schmidt の直交化法である [6]。このアルゴリズムにより、m 本のベクトル$\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_m$から、正規直交化された m 本のベクトル$\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_m$ が生成される。

Algorithm 4.1. 修正 Gram–Schmidt の直交化法

for $k = 1, \cdots, m$
$\quad R_{kk} = \sqrt{(\boldsymbol{a}_k, \boldsymbol{a}_k)_{\boldsymbol{H}}}$
$\quad \boldsymbol{q}_k = \boldsymbol{a}_k / R_{kk}$
$\quad$ **for** $j = k+1, \cdots, m$
$\quad\quad R_{kj} = (\boldsymbol{q}_k, \boldsymbol{a}_j)_{\boldsymbol{H}}$
$\quad\quad \boldsymbol{a}_j = \boldsymbol{a}_j - R_{kj}\boldsymbol{q}_k$
$\quad$ **end**
end

ここで、ベクトルの集合として$\boldsymbol{A} \equiv (\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_m)$, $\boldsymbol{Q} \equiv (\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_m)$ と定義する。このとき、行列$\boldsymbol{A}, \boldsymbol{Q}$と上記のアルゴリズムによって計算された上三角行列$\boldsymbol{R}$に対し、$\boldsymbol{A} = \boldsymbol{QR}$ という関係が成り立つ。また、行列$\boldsymbol{Q}$の縦ベクトル$\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_m$は正規直交系であるので、$\boldsymbol{Q}^T\boldsymbol{HQ} = \boldsymbol{I}$ である。

4.2 SDPA-CG の場合

Algorithm 4.1 の修正 Gram-Schmidt の直交化法を SDPA-CG の場合に適用する。2.1 節で表わした (4) に注目すると、

$$\begin{aligned} B_{ij} &= \mathbf{vec}(\boldsymbol{A}_i)^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathbf{vec}(\boldsymbol{A}_j) \\ &= (\mathbf{vec}(\boldsymbol{A}_i), \mathbf{vec}(\boldsymbol{A}_j))_{\boldsymbol{Z}^{-1}\otimes\boldsymbol{X}} \end{aligned}$$

となるので、これは $\mathbf{vec}(\boldsymbol{A}_i)$ と $\mathbf{vec}(\boldsymbol{A}_j)$ の$\boldsymbol{Z}^{-1}\otimes\boldsymbol{X}$の意味での内積と捉えることができる。なお、$\boldsymbol{Z}^{-1}$,$\boldsymbol{X}$は共に正定値行列であるので、$\boldsymbol{Z}^{-1}\otimes\boldsymbol{X}$も正定値行列である。つまり、$\boldsymbol{Z}^{-1}\otimes\boldsymbol{X}$での内積のもとで、$\boldsymbol{B}$の各要素は、SDP(2) の制約行列を表すのベクトル $\mathbf{vec}(\boldsymbol{A}_1), \mathbf{vec}(\boldsymbol{A}_2), \cdots, \mathbf{vec}(\boldsymbol{A}_m)$ の内積の集合とみなすことができる。よって、この内積で m 本のベクトル $\mathbf{vec}(\boldsymbol{A}_1), \mathbf{vec}(\boldsymbol{A}_2), \cdots, \mathbf{vec}(\boldsymbol{A}_m)$ の正規直交化を行う。このとき、**Algorithm 4.1** の 2 行目、4 行目は

$$\begin{aligned} R_{kk} &= \sqrt{(\mathbf{vec}(\boldsymbol{A}_k), \mathbf{vec}(\boldsymbol{A}_k))_{\boldsymbol{Z}^{-1}\otimes\boldsymbol{X}}} \\ &= \sqrt{\mathbf{vec}(\boldsymbol{A}_k)^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathbf{vec}(\boldsymbol{A}_k)} \\ &= \sqrt{\boldsymbol{A}_k \bullet \boldsymbol{X}\boldsymbol{A}_k\boldsymbol{Z}^{-1}} \end{aligned}$$

$$\begin{aligned} R_{kj} &= (\mathbf{vec}(\boldsymbol{Q}_k), \mathbf{vec}(\boldsymbol{A}_j))_{\boldsymbol{Z}^{-1}\otimes\boldsymbol{X}} \\ &= \mathbf{vec}(\boldsymbol{Q}_k)^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathbf{vec}(\boldsymbol{A}_j) \\ &= \boldsymbol{Q}_k \bullet \boldsymbol{X}\boldsymbol{A}_j\boldsymbol{Z}^{-1} \end{aligned}$$

となる。ここで、行列$\boldsymbol{A}_i$の構造にあわせ、ベクトル$\boldsymbol{q}_i$を行列$\boldsymbol{Q}_i$にする。以上のことを考慮すると、**Algorithm 4.1** の修正 Gram–Schmidt の直交化法は、以下のように書き直すことができる。

Algorithm 4.2. 修正 Gram–Schmidt の直交化法 (SDPA-CG 版)

for $k = 1, \cdots, m$
$\quad R_{kk} = \sqrt{\boldsymbol{A}_k \bullet \boldsymbol{X}\boldsymbol{A}_k\boldsymbol{Z}^{-1}}$
$\quad \boldsymbol{Q}_k = \boldsymbol{A}_k / R_{kk}$
$\quad$ **for** $j = k+1, \cdots, m$

$$R_{kj} = \boldsymbol{Q}_k \bullet \boldsymbol{X}\boldsymbol{A}_j\boldsymbol{Z}^{-1}$$
$$\boldsymbol{A}_j = \boldsymbol{A}_j - R_{kj}\boldsymbol{Q}_k$$

end

end

2.1節で定義したように、$\mathcal{A}$はm本のベクトルの集まり$\mathcal{A} \equiv (\mathbf{vec}(\boldsymbol{A}_1), \mathbf{vec}(\boldsymbol{A}_2), \cdots, \mathbf{vec}(\boldsymbol{A}_m))$であり、同様に$\mathcal{Q}$を$\mathcal{Q} \equiv (\mathbf{vec}(\boldsymbol{Q}_1), \mathbf{vec}(\boldsymbol{Q}_2), \cdots, \mathbf{vec}(\boldsymbol{Q}_m))$と定義する。すると、$\mathcal{A}$,$\mathcal{Q}$と、**Algorithm 4.2**により生成される上三角行列$\boldsymbol{R}$の間に、4.1節で示したように以下の関係が満たされる。

$$\mathcal{A} = \mathcal{Q}\boldsymbol{R}$$
$$\mathcal{Q}^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathcal{Q} = \boldsymbol{I}$$

これは**Algorithm 4.2**により、$\mathcal{A}$に対し特別な内積でのQR分解が行われたことを示す。このとき、$\mathcal{A}$,$\mathcal{Q}$,$\boldsymbol{R}$を線形方程式系の係数行列$\boldsymbol{B}$の定義式(5)に代入すると、

$$\begin{aligned}\boldsymbol{B} &\equiv \mathcal{A}^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathcal{A} \\ &= \boldsymbol{R}^T\mathcal{Q}^T(\boldsymbol{Z}^{-1} \otimes \boldsymbol{X})\mathcal{Q}\boldsymbol{R} \\ &= \boldsymbol{R}^T\boldsymbol{R}\end{aligned}$$

という関係が成り立つ。つまり、制約行列$\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_m$に対し**Algorithm 4.2**の修正Gram-Schmidtの直交化を行うことにより、線形方程式系(2)の係数行列$\boldsymbol{B}$のコレスキー分解が出来る。ここで、この上三角行列$\boldsymbol{R}$を共役勾配法の前処理として適用すると、$\boldsymbol{R}^{-T}\boldsymbol{B}\boldsymbol{R}^{-1} = \boldsymbol{I}$であるから、共役勾配法の収束性を高めるという点では非常に良い前処理行列である。しかし、mが大きなSDP問題を解く場合、**Algorithm 4.2**により上三角行列$\boldsymbol{R}$を計算することは、直接法で線形方程式系(2)で解くことと同等であり、依然計算時間や使用メモリ量の点で計算は困難であることに変わりはない。

4.3 前処理の統一的な枠組み

ここでは、4.2節で説明した$\mathcal{A}$によるQR分解をもとに、前処理の統一的な枠組みを提案する。**Algorithm 4.2**を完全に行い上三角行列$\boldsymbol{R}$を生成するのは困難であるため、Gram–Schmidtの直交化法において、すべてのベクトル同士の直交化を行わず一部のベクトル同士の直交化のみを行う。この不完全なGram–Schmidtの直交化法によって出来た上三角行列$\boldsymbol{R}'$を本来導かれる$\boldsymbol{R}$の近似行列とみなし、共役勾配法の前処理行列として利用する。すべての直交化の対象となるベクトルの組の集合を$S(\equiv \{(i,j) \mid i < j\})$と定義する。そして、この枠組みにより実際に直交化を行うベクトルの組の集合を$S'(\subset S)$と定義する。まず、直交化を行うベクトルの組の集合をS'を定め、以下で示される不完全な修正Gram–Schmidtの直交化法により$\boldsymbol{B}$のコレスキー分解行列$\boldsymbol{R}$の近似行列$\boldsymbol{R}'$を生成する。

Algorithm 4.3. 不完全な修正Gram–Schmidtの直交化法

for $k = 1, \cdots, m$

$$R'_{kk} = \sqrt{\boldsymbol{A}_k \bullet \boldsymbol{X}\boldsymbol{A}_k\boldsymbol{Z}^{-1}}$$
$$\boldsymbol{Q}_k = \boldsymbol{A}_k / R'_{kk}$$

for j **in** $(k,j) \in S'$

$$R'_{kj} = \boldsymbol{Q}_k \bullet \boldsymbol{X}\boldsymbol{A}_j\boldsymbol{Z}^{-1}$$
$$\boldsymbol{A}_j = \boldsymbol{A}_j - R'_{kj}\boldsymbol{Q}_k$$

end

end

ここで、S'としてSをとると、**Algorithm 4.3** で生成される上三角行列は、$\boldsymbol{B}$のコレスキー分解行列となるため、この枠組みは$\boldsymbol{B}$のコレスキー分解つまり直接法を含んでいるといえる。一般に、より多くの直交化を行ない生成された上三角行列$\boldsymbol{R}'$は$\boldsymbol{R}$との近似性が高く、前処理行列として利用すると収束性が良いことが期待できる。しかし、多くの直交化を行うと、計算時間やメモリ使用量が増えることが予想されるため、適当なS'を使い前処理行列を生成する必要がある。また、S'によっては、**Algorithm 4.3** によって生成できる前処理行列を、別の効率的な計算法により生成できる可能性があるが、ここでは前処理の実装の効率性にはふれない。

4.4 幾つかの前処理

4.3 節で提案した枠組みをもとに、以下の 3 つの前処理法を導出した。

(i) 正規化のみを行い直交化は全く行わない

(ii) k本ずつのグループに分け、グループ内のベクトルと直交化させる

(iii) 直前に計算したk本のベクトルと直交化させる

各々の前処理における直交させるベクトルの組の集合は、(i) では $S' = \emptyset$、(ii) では $S' = \{(i,j) \subset S \mid \lceil i/k \rceil = \lceil j/k \rceil\}$、(iii) では $S' = \{(i,j) \subset S \mid |i-j| \leq k\}$ となる。また、(i) は対角スケーリング、(ii) は特殊な不完全コレスキー分解と一致し、これらの前処理は共役勾配法の代表的な前処理である [6]。

これらの前処理の効果を調べるため、サイズが $n = 100, m = 2513$ である SDP で主双対内点法の 6 反復目での探索方向の計算で解く線形方程式系 (2) に対し、前処理付き共役勾配法の収束性を測定した。図 2 は前処理 (ii) で $k = 1, 10, 100$ としたときの実験結果であり、横軸に共役勾配法の反復回数、縦軸に線形方程式系の誤差 ($\|\boldsymbol{s} - \boldsymbol{B}\boldsymbol{dy}\|$) をとっている。また、図 3 は前処理 (iii) で $k = 1, 10, 100$ としたときの実験結果であり、同様に横軸に共役勾配法の反復回数、縦軸が線形方程式系の誤差をとっている。なお、前処理 (ii) と前処理 (iii) において $k = 1$ の場合は、共に前処理 (i) となっていることに注意する。図 2 と図 3 のグラフより、kが大きくなり各ベクトル同士の直交性が増すに連れ、共役勾配法の収束性が向上するのが確認できる。また、前処理を行わず共役勾配法を実行するのに比べ、前処理 (i)(グラフの $k = 1$) を適用し共役勾配法を実行すると、共役勾配法の収束はある程度良くなる。しかし、この実験のでは、前処理 (ii) や前処理 (iii) で $k = 10, 100$ としても前処理を行うコストに見合う程、共役勾配法の収束性は向上しないことが確認された。

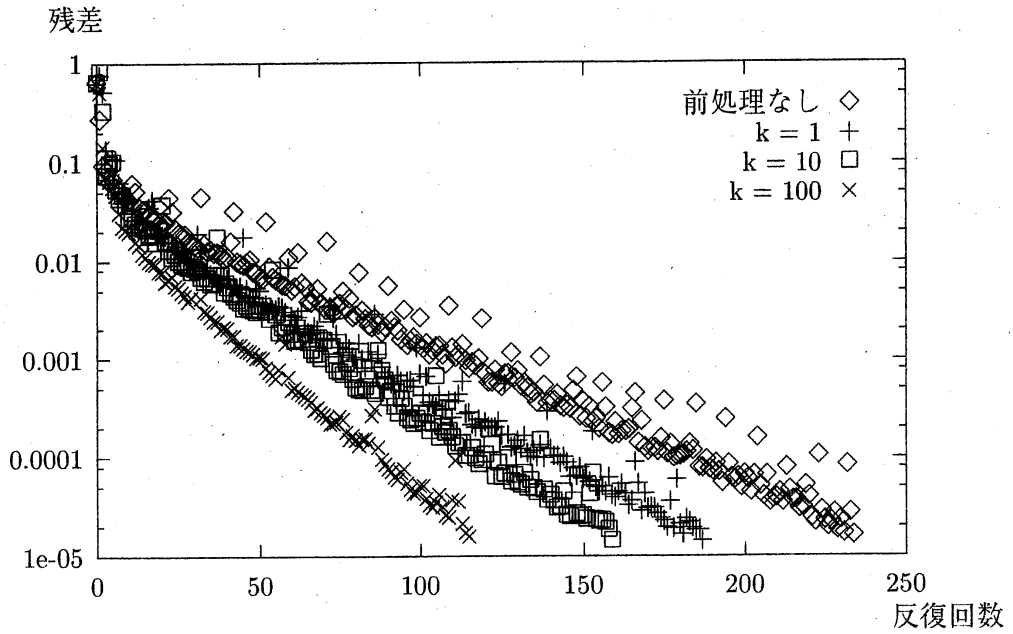

図 2: 前処理 (ii) での 反復回数 − 残差 グラフ

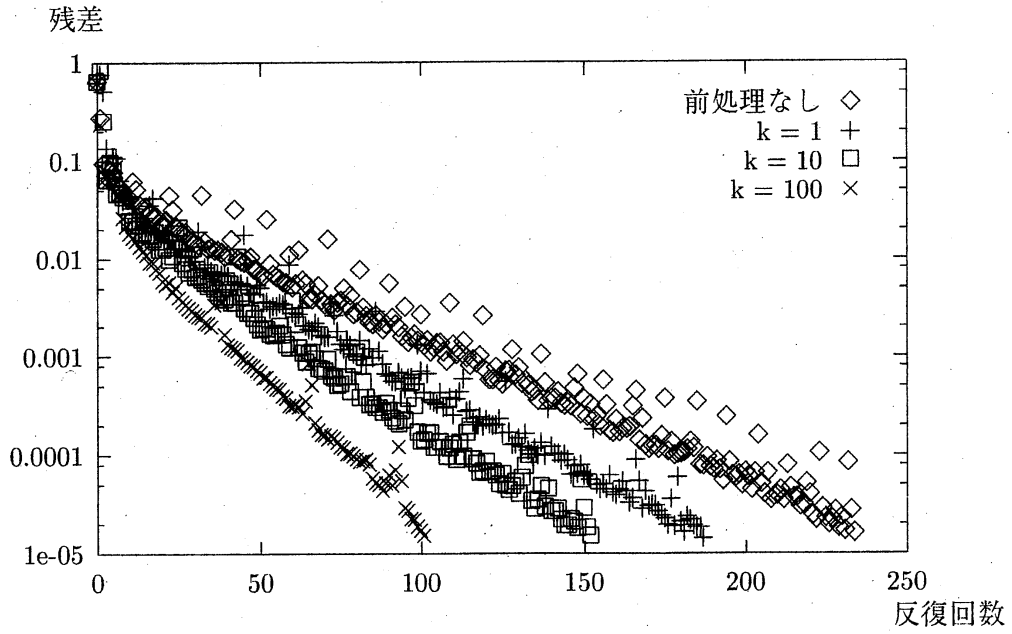

図 3: 前処理 (iii) での 反復回数 − 残差 グラフ

5 数値実験

本節では、SDPA と SDPA-CG を用いて最大クリーク問題の SDP 緩和の数値実験を行う。最大クリーク問題の SDP 緩和は、m が $O(n^2)$ であり、n に対して m が非常に大きな問題である。数値実験は、ランダムグラフに対する SDP 緩和の計算を行い、主双対内点法の終了条件はある整数が特定できるまでとした。なお計算実験は、DEC Alpha (CPU 21164) 300MHz, 256MB メモリで行った。また、SDPA-CG で使用する共役勾配法の前処理として対角スケーリングを利用した。実験結果は表 2 に示す。表 2 には、SDPA および SDPA-CG の主反復回数、計算時間、メモリ使用量が示してある。なお、表中の空白の部分は、主記憶上に全データを保持できず、解くことが不可能あったたことを表す。

表 2: 最大クリーク問題の SDP 緩和での実験結果

		SDPA			SDPA-CG		
n	m	反復回数	時間 (秒)	メモリ (MB)	反復回数	時間 (秒)	メモリ (MB)
100	492	6	9.2	8.0	9	15.8	5.1
100	926	5	28.7	13.4	7	9.8	5.3
100	2018	5	422.4	40.9	7	11.7	5.9
100	3918	6	3818.6	140.0	10	29.3	6.9
300	4519	7	6650.8	197.0	10	880.3	23.3
300	8997				8	384.5	25.5
300	17773				8	435.0	32.4
300	36075				8	368.9	38.4
500	12515				10	5014.3	65.8
500	24919				9	3466.3	72.5
500	49602				8	2439.8	85.3
500	99654				9	2508.5	112.0
700	24311				10	20626.9	115.0
700	49039				10	27356.6	139.0
700	98519				9	14656.7	166.0
700	196000				9	12826.1	217.0

表 2 より、SDPA-CG は SDPA に比べ計算時間とメモリ使用量が大幅に改善されており、特に n に対し m が大きくなるにつれその比が顕著になることが確認できる。また、SDPA-CG により従来計算が不可能であった超大規模 SDP ($n = 700, m = 196000$) を解くことができた。これらの実験結果より、SDPA-CG は最大クリーク問題の SDP 緩和を解く場合、非常に有効であるといえる。

6 おわりに

本研究では、SDP を主双対内点法により解くソフトウェア SDPA に共役勾配法を組み入れたソフトウェア SDPA-CG の開発を行った。SDPA-CG の大きな特徴は、主双対内点法の各反復で探索方向の計算の際に現れる線形方程式系の係数行列を保持していないことにある。これにより、メモリ使用量を少なく押さえることができ、n に対し m が大きな大規模な SDP を扱うこと可能になった。また、計算実験を通して、精度の良い解を求めようとすると共役勾配法の反復回数は指数的に増え、それに伴い計算時間が増大することが分かった。つまり、SDPA-CG でより厳密な最適解を求めるためには、共役勾配法の収束性を高める必要がある。そのため、線形方程式系の制約行列の構造を基に、不完全 Gram–Schmidt の直交化法という観点から、共役勾配法の前処理の統一的な枠組みを提案した。そして、この枠組みから幾つかの前処理法を導出し、各々の前処理が共役勾配法の収束性に与える影響を測定した。しかし、これらの前処理法の計算時間やメモリ使用量などついては今後の研究課題である。最後に、超大規模な最大クリーク問題の SDP 緩和を SDPA-CG と SDPA によって解くことにより、SDPA-CG が計算時間およびメモリ使用量を大幅に改善させることを実証した。今後は、他の超大規模 SDP への SDPA-CG の適用や、効率の良い前処理による高精度の解の計算などが求められる。

参考文献

[1] Alizadeh, F., Haeberly J.-P.A., Nayakkankuppam, M.V., Overton, M.L., and Schmieta, S. (1997). "SDPpack – User's Guide –," Comp. Sci. Dept., New York University, New York. Available at http://cs.nyu.edu/cs/faculty/overton/sdppack/sdppack.html.

[2] Borchers, B. (1997). "CSDP, a C library for semidefinite programming," Department of Mathematics, New Mexico Institute of Mining and Technology, 801 Leroy Place Socorro, New Mexico 87801. Available at http://www.nmt.edu/~borchers/csdp.html.

[3] Fujisawa, K., Fukuda, M., Kojima, M. and Nakata, K. (1997) "Numerical Evaluation of SDPA(SemiDefinite Programming Algorithm)," Research Report B-330, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, Japan.

[4] Fujisawa, K., Kojima, M. and Nakata, K. (1996). "SDPA(Semidefinite Programming Algorithm) – User's Manual –," Technical Report B-308, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, Japan. Available at ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA.

[5] Fujisawa, K., Kojima, M. and Nakata, K.(1997). "Exploiting Sparsity in Primal-Dual Interior-Point Methods for Semidefinite Programming," *Mathematical Programming* **79** 235–253.

[6] Golub, G.H. and Van Loan, C.F. (1983). *Matrix Computations,* (The John Hopkins University Press, Baltimore, Maryland).

[7] Hestenes, M. and Stiefel, E. (1952). "Methoeds of Conjugate Gradients for Solving Linear Systems," *J. Res. Nat. Bur. Standards*, **49** 409-436.

[8] 小島政和. (1996). "半正定値計画問題と内点法," 応用数理, Vol 6, 16—25.

[9] 中田和秀, 藤沢克樹, 小島政和. (1998). "半正定値計画問題に対する主双対内点法における共役勾配法の実装," 統計数理, 第 46 巻 第 2 号 297–316.

[10] Potra, F.A., Sheng, R. and Brixius, N. (1997). "SDPHA – a MATLAB implementation of homogeneous interior-point algorithms for semidefinite programming," Department of Mathematics, University of Iowa, Iowa City, IA 52242,
Available at http://www.math.uiowa.edu/~rsheng/SDPHA/sdpha.html.

[11] Sturm, J.F., (1998). "USING SEDUMI 1.0x, A MATLAB TOOLBOX FOR OPTIMIZATION OVER SYMMETRIC CONES," Department of Quantitative Economics, Maastricht University, Maastricht, The Netherlands,
Available at http://www.unimaas.nl/~sturm/software/sdpha.html.

[12] Todd, M. (1997). "On search directions in interior-point methods for semidefinite programming," Technical Report No.1205, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853-3801.

[13] Todd, M.J., Toh, K.C. and Tütüncü, R.H. (1996). "On the Nesterov-Todd direction in semidefinite programming," Technical Report, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York 14853-3801, USA.

[14] Vandenberghe, L. and Boyd, S. (1996). " Semidefinite Programming," *SIAM Review* **38** 49–95.

[15] Vandenberghe, L. and Boyd, S. (1994). "SP: Software for Semidefinite Programming, User's Guide, Beta Version," Stanford University, Stanford, CA.