数理解析研究所講究録 1288 巻 2002 年 52-63

大規模線形方程式を解くためのクリロフ部分空間法の前処理

東京工業大学　社会理工学研究科　　中田 和秀

1　はじめに

$\boldsymbol{A} \in \mathcal{C}^{m\times m}, \boldsymbol{b} \in \mathcal{C}^m$ が与えられているとき、線形方程式系

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

の解 $\boldsymbol{x} \in \mathcal{C}^m$ を求めたい。線形方程式系 (1) の解法として、LU 分解などの直接法が良く知られている。しかし直接法では、この線形方程式系 (1) のサイズ m に対し、一般に $\mathcal{O}(m^3)$flops の計算量と係数行列 $\boldsymbol{A}$ の記憶が必要となる。このため、大規模な線形方程式系を解くことは困難である。たとえ係数行列 $\boldsymbol{A}$ が疎行列であっても、LU 分解などの直接法では、計算途中で fill-in が起こりその疎性が崩れる事が多い。一方、クリロフ部分空間法では係数行列を直接に操作することは無く、ベクトルに対する線形変換にのみ利用するため、係数行列の疎性の維持が容易である。また、クリロフ部分空間法の収束が速ければ高速に近似解を得る可能性がある。このため、本論文ではクリロフ部分空間法により線形方程式系 (1) を解くことを試みる。2 章では、4 つのクリロフ部分空間法と前処理の一般的な枠組みについて述べる。3 章では、幾つかの前処理行列について説明を行う。4 章では半正定値計画問題を例に取り、係数行列が密である特殊な線形方程式系に対して、クリロフ部分空間法が有効であることを示す。

2　クリロフ部分空間法とその前処理

クリロフ部分空間 $\mathcal{K}_n(\boldsymbol{A};\boldsymbol{b})$ を、

$$\mathcal{K}_n(\boldsymbol{A};\boldsymbol{b}) = \mathrm{span}\{\boldsymbol{b}, \boldsymbol{A}\boldsymbol{b}, \boldsymbol{A}^2\boldsymbol{b}, \cdots, \boldsymbol{A}^{n-1}\boldsymbol{b}\}$$

で定義する。クリロフ部分空間法は非定常な反復解法であり、n 反復目の反復点 $\boldsymbol{x}_n$ がクリロフ部分空間 $\mathcal{K}_n(\boldsymbol{A};\boldsymbol{b})$ の中をなんらかの意味で探索する方法である。(本論文では、記述を簡略化するため、クリロフ部分空間法の初期点をとして $\boldsymbol{x}_0 = \boldsymbol{0}$ を仮定する。) これまで、多くのクリロフ部分空間法が提案されてきたが、歴史的には係数行列が正定値行列である線形方程式系に対する解法として、Hestenes と Stiefel が共役勾配法 (Conjugate Gradient method: CG) を発表したことに始まる [6]。その後、同じく係数行列が正定値な線形方程式系に対する解法である共役残差法 (Conjugate Residual method: CR) や、一般の線形方程式系に対する解法である双共役勾配法 (Biconjugate Gradient

method: Bi-CG)、2乗共役勾配法 (Conjugate Gradient Squared method: CGS)、安定化双共役勾配法 (Biconjugate Gradient Stabilized method: Bi-CGSTAB)、疑似最小残差法 (Quasi-Minimal Residual method: QMR)、一般化共役残差法 (Generalized Conjugate Residual method: GCR)、リスタート版一般化共役残差法 (Restarted Generalized Conjugate Redisual method: GCR(m))、一般化最小残差法 (Generalized Minimal Residual method: GMRES)、リスタート版一般化最小残差法 (Restarted Generalized Minimal Residual method: GMRES(m)) などのクリロフ部分空間法が提案されている [2, 3, 5]。

クリロフ部分空間法の収束性は係数行列 $\boldsymbol{A}$ の固有値の分布に依存し、特に係数行列 $\boldsymbol{A}$ の固有値が密集している場合には高速に収束することが知られている。このため、線形方程式系 (1) を解く変わりに、正則行列 $\boldsymbol{M}_1, \boldsymbol{M}_2 \in \mathcal{C}^{m\times m}$ に対し、同値な線形方程式系

$$(\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{M}_2)(\boldsymbol{M}_2^{-1}\boldsymbol{x}) = (\boldsymbol{M}_1\boldsymbol{b}) \tag{2}$$

を考え、この線形方程式系をクリロフ部分空間法で解くことが有効である。このとき、$\boldsymbol{M}_1\boldsymbol{A}\boldsymbol{M}_2$ の固有値が密集していれば、新しいクリロフ部分空間法は高速に収束することが期待できる。この技術を前処理と呼び、線形方程式系 (2) をクリロフ部分空間法で解くことを前処理付きクリロフ部分空間法という。元の線形方程式系 (1) の係数行列 $\boldsymbol{A}$ が正定値である場合、その正定値性を壊さない前処理が求められる。そのような場合、線形方程式系 (2) の代りに、正則行列 $\boldsymbol{M} \in \mathcal{C}^{m\times m}$ に対し、

$$(\boldsymbol{M}\boldsymbol{A}\boldsymbol{M}^*)(\boldsymbol{M}^{-*}\boldsymbol{x}) = (\boldsymbol{M}\boldsymbol{b}) \tag{3}$$

という線形方程式系を考え、これに共役勾配法や共役残差法を適用することが大変有効である。

本章では以下、典型的な4つのクリロフ部分空間法を例に取り、そららに対し前処理を適用することにより、前処理付きのクリロフ部分空間法の概要を説明する。

2.1 共役勾配法

本節では、クリロフ部分空間法の中で最も基本的かつ有名な方法である共役勾配法について述べる。共役勾配法は、係数行列が正定値行列である線形方程式系を解くクリロフ部分空間法である。前節で述べたように、クリロフ部分空間法は n 反復目の反復点 $\boldsymbol{x}_n$ がクリロフ部分空間 $\mathcal{K}_n(\boldsymbol{A};\boldsymbol{b})$ を探索する反復解法であるが、特に共役勾配法では反復点 $\boldsymbol{x}_n$ が

$$\min_{\boldsymbol{x}\in\mathcal{K}_n(\boldsymbol{A};\boldsymbol{b})} ||\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}||_{A^{-1}} \tag{4}$$

の解となっていることが知られている。この性質は、理論的には共役勾配法が高々 m 回での反復で線形方程式系の厳密解を得ることを保証する。しかし、一般にクリロフ部分空間法の計算は誤差が生じやすいことが知られており、共役勾配法において m 回の反復での終了は期待できない場合も多い。アルゴリズム1で線形方程式系 (1) を解く共役勾配法を示している。アルゴリズム中の α, β には、理論的には同じ値を導く幾つかの計算方法が存在することを付記しておく。係数行列 $\boldsymbol{A}$ はベクトルの線形変換 $\boldsymbol{A}\boldsymbol{p}_n$ にのみ使われている。ここで $\#(A)$ を行列 A の非ゼロ要素の数とすると、共役勾配法の1反復当りの計算量は、$2\,\#(A) + \mathcal{O}(m)$ flops となる。

アルゴリズム 1 :
共役勾配法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$\alpha_n = \dfrac{(\boldsymbol{r}_n, \boldsymbol{p}_n)}{(\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$ or $\dfrac{(\boldsymbol{r}_n, \boldsymbol{r}_n)}{(\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$
$\beta_n = -\dfrac{(\boldsymbol{r}_{n+1}, \boldsymbol{A}\boldsymbol{p}_n)}{(\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$ or $\dfrac{(\boldsymbol{r}_{n+1}, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_n, \boldsymbol{r}_n)}$
$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \beta_n \boldsymbol{p}_n$
end

アルゴリズム 2 :
前処理付き共役勾配法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$\alpha_n = \dfrac{(\boldsymbol{r}_n, \boldsymbol{p}_n)}{(\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$
$\widetilde{\boldsymbol{r}_{n+1}} = \boldsymbol{K}\boldsymbol{r}_{n+1}$
$\beta_n = -\dfrac{(\widetilde{\boldsymbol{r}_{n+1}}, \boldsymbol{A}\boldsymbol{p}_n)}{(\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$
$\boldsymbol{p}_{n+1} = \widetilde{\boldsymbol{r}_{n+1}} + \beta_n \boldsymbol{p}_n$
end

線形方程式系 (3) に対し共役勾配法を適用することを前処理付き共役勾配法という。この前処理付き共役勾配法を、さらに数学的に同値なアルゴリズムに変形したものをアルゴリズム 2 で示した (変形については [5] などを参照)。なお、アルゴリズム 2 中の $\boldsymbol{K}$ は $\boldsymbol{K} = \boldsymbol{M}^*\boldsymbol{M}$ で定義される行列である。ただし、この行列 $\boldsymbol{K}$ を実際にメモリ上に保持する必要はなく、$\boldsymbol{r}_{n+1}$ に対する線形変換 $\boldsymbol{K}\boldsymbol{r}_{n+1}$ が計算できればよい ($\boldsymbol{K}$ の正定値性は必要である)。前処理付き共役勾配法を用いた場合、1 反復の計算量は通常の共役勾配法に比べ、線形変換 $\boldsymbol{K}\boldsymbol{r}_{n+1}$ を行う分だけ増える。このため、前処理付き共役勾配法の 1 反復当たりの計算量を増やさないためには、線形変換 $\boldsymbol{K}\boldsymbol{r}_{n+1}$ を少ない計算時間とメモリ量で行う必要がある。また、前節で述べたように行列 $\boldsymbol{M}\boldsymbol{A}\boldsymbol{M}^*$ の固有値がより密集しているほうが、前処理付き共役勾配法は高速に収束するため、$\boldsymbol{K}$ が $\boldsymbol{A}^{-1}$ の近似行列であることが望ましい。これらの性質をふまえた上で、具体的な前処理法については次章で述べる。

2.2 共役残差法

共役残差法は、係数行列が正定値行列である線形方程式系を解くクリロフ部分空間法である。この方法の n 回目の反復点 $\boldsymbol{x}_n$ は、

$$\min_{\boldsymbol{x} \in \mathcal{K}_n(\boldsymbol{A}; \boldsymbol{b})} ||\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}||$$

の解となっていることが知られている。共役勾配法の場合と同様に、この性質は理論的には共役残差法の m 回での収束性を保証するものの、実際には誤差の影響を受けるため理論通りにはならない。アルゴリズム 3 に共役残差法を示した。

アルゴリズム 3 :
共役残差法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$$\alpha_n = \frac{(\boldsymbol{r}_n, \boldsymbol{A}\boldsymbol{p}_n)}{(\boldsymbol{A}\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$$
$$\beta_n = -\frac{(\boldsymbol{A}\boldsymbol{r}_{n+1}, \boldsymbol{A}\boldsymbol{p}_n)}{(\boldsymbol{A}\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$$
$$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \beta_n \boldsymbol{p}_n$$
end

アルゴリズム 4 :
前処理付き共役残差法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$$\alpha_n = \frac{(\widetilde{\boldsymbol{r}_n}, \boldsymbol{A}\boldsymbol{p}_n)}{(\boldsymbol{A}\boldsymbol{p}_n, \widetilde{\boldsymbol{A}\boldsymbol{p}_n})}$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$$
$$\widetilde{\boldsymbol{r}_{n+1}} = \boldsymbol{K}\boldsymbol{r}_{n+1}$$
$$\beta_n = -\frac{(\boldsymbol{A}\widetilde{\boldsymbol{r}_{n+1}}, \widetilde{\boldsymbol{A}\boldsymbol{p}_n})}{(\boldsymbol{A}\boldsymbol{p}_n, \widetilde{\boldsymbol{A}\boldsymbol{p}_n})}$$
$$\boldsymbol{p}_{n+1} = \widetilde{\boldsymbol{r}_{n+1}} + \beta_n \boldsymbol{p}_n$$
$$\widetilde{\boldsymbol{A}\boldsymbol{p}_{n+1}} = \boldsymbol{K}\boldsymbol{A}\boldsymbol{p}_{n+1}$$
end

アルゴリズム 5 :
前処理付き共役残差法（実用版）

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$$\alpha_n = \frac{(\widetilde{\boldsymbol{z}_n}, \boldsymbol{q}_n)}{(\boldsymbol{q}_n, \widetilde{\boldsymbol{q}_n})}$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{q}_n$$
$$\widetilde{\boldsymbol{r}_{n+1}} = \widetilde{\boldsymbol{r}_n} - \alpha_n \widetilde{\boldsymbol{q}_n}$$
$$\boldsymbol{z}_{n+1} = \boldsymbol{A}\widetilde{\boldsymbol{r}_{n+1}}$$
$$\beta_n = -\frac{(\boldsymbol{z}_{n+1}, \widetilde{\boldsymbol{q}_n})}{(\boldsymbol{q}_n, \widetilde{\boldsymbol{q}_n})}$$
$$\boldsymbol{p}_{n+1} = \widetilde{\boldsymbol{r}_{n+1}} + \beta_n \boldsymbol{p}_n$$
$$\boldsymbol{q}_{n+1} = \boldsymbol{z}_{n+1} + \beta_n \boldsymbol{q}_n$$
$$\widetilde{\boldsymbol{q}_{n+1}} = \boldsymbol{K}\boldsymbol{q}_{n+1}$$
end

線形方程式系 (3) に対し共役残差法を適用することを前処理付き共役残差法という。この方法をアルゴリズム 4 で示した。なお、アルゴリズム 4 中の $\boldsymbol{K}$ は $\boldsymbol{K} = \boldsymbol{M}^*\boldsymbol{M}$ で定義する。アルゴリズム 4 より、共役残差法の各反復では、係数行列 $\boldsymbol{A}$ の線形変換を 2 回と、前処理行列 $\boldsymbol{K}$ の線形変換を 2 回行うことになる。このため、アルゴリズム 4 の計算は前処理付き共役勾配法の計算の 2 倍かかることになる。しかし、[12] で述べたように、

$$\begin{aligned}\widetilde{\boldsymbol{r}_{n+1}} &= \boldsymbol{K}\boldsymbol{r}_{n+1} \\ &= \boldsymbol{K}(\boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n) \\ &= \widetilde{\boldsymbol{r}_n} - \alpha_n \boldsymbol{K}\boldsymbol{A}\boldsymbol{p}_n\end{aligned}$$

という関係を利用することにより、$\widetilde{\boldsymbol{r}_{n+1}}$ は $\widetilde{\boldsymbol{r}_n}$ と $\boldsymbol{K}\boldsymbol{A}\boldsymbol{p}_n$ から更新することができる。また、

$$\begin{aligned}\boldsymbol{A}\boldsymbol{p}_{n+1} &= \boldsymbol{A}(\widetilde{\boldsymbol{r}_{n+1}} + \beta_n \boldsymbol{p}_n) \\ &= \boldsymbol{A}\widetilde{\boldsymbol{r}_{n+1}} + \beta_n \boldsymbol{A}\boldsymbol{p}_n\end{aligned}$$

という関係を利用することにより、$\boldsymbol{A}\boldsymbol{p}_{n+1}$ は $\boldsymbol{A}\widetilde{\boldsymbol{r}_{n+1}}$ と $\boldsymbol{p}_n$ から更新することが可能である。この更新法で計算することにより、前処理付き共役残差法の 1 反復当たりの計算量は、共役勾配法の 1 反復当たりの計算量と同程度にすることができる。この方法を実用版の前処理付き共役残差法としてアルゴリズム 5 に示した。共役勾配法に対する前処理の場合と同様に、行列 $\boldsymbol{K}$ は正定値行列であることが必要であるが、アルゴリズムの実行において実際にメモリ上に保持する必要はなく、線形変換 $\boldsymbol{K}\boldsymbol{q}_{n+1}$ が行えれば良い。このとき、なるべく少ない計算量とメモリ量で計算できることが必要である。さらに、共役残差法の収束性を高めるためには、$\boldsymbol{K}$ が $\boldsymbol{A}^{-1}$ の近似行列であることが望ましい。

2.3 双共役勾配法

本節では、双共役勾配法について述べる。共役勾配法が係数行列が正定値である線形方程式系に対する解法であったのに対し、この方法は正定値性を仮定しない一般の係数行列に持つ線形方程式系に対する解法である。次に述べるアルゴリズムから想像出来るように、これは共役勾配法を拡張した方法である。また、この双共役勾配法から派生した方法として２乗共役勾配法、安定化共役勾配法、疑似最小残差法などがあるが、詳細については[2]などを参照されたい。アルゴリズム６で、線形方程式系 (1) を解く双共役勾配法を示した。

アルゴリズム６:
双共役勾配法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
$\boldsymbol{p}'_0 = \boldsymbol{b},\ \boldsymbol{r}'_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$\alpha_n = \dfrac{(\boldsymbol{r}'_n, \boldsymbol{r}_n)}{(\boldsymbol{p}'_n, \boldsymbol{A}\boldsymbol{p}_n)}$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$
$\boldsymbol{r}'_{n+1} = \boldsymbol{r}'_n - \overline{\alpha}_n \boldsymbol{A}^* \boldsymbol{p}'_n$
$\beta_n = \dfrac{(\boldsymbol{r}'_{n+1}, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}'_n, \boldsymbol{r}_n)}$
$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \beta_n \boldsymbol{p}_n$
$\boldsymbol{p}'_{n+1} = \boldsymbol{r}'_{n+1} + \overline{\beta}_n \boldsymbol{p}'_n$
end

アルゴリズム７:
前処理付き双共役勾配法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
$\boldsymbol{p}'_0 = \boldsymbol{b},\ \boldsymbol{r}'_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$\alpha_n = \dfrac{(\boldsymbol{r}'_n, \widetilde{\boldsymbol{r}_n})}{(\boldsymbol{p}'_n, \boldsymbol{A}\boldsymbol{p}_n)}$
$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$
$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$
$\boldsymbol{r}'_{n+1} = \boldsymbol{r}'_n - \overline{\alpha}_n \boldsymbol{A}^* \boldsymbol{p}'_n$
$\widetilde{\boldsymbol{r}_{n+1}} = \boldsymbol{K}\boldsymbol{r}_{n+1}$
$\widetilde{\boldsymbol{r}'_{n+1}} = \boldsymbol{K}^* \boldsymbol{r}'_{n+1}$
$\beta_n = \dfrac{(\boldsymbol{r}'_{n+1}, \widetilde{\boldsymbol{r}_{n+1}})}{(\boldsymbol{r}'_n, \widetilde{\boldsymbol{r}_n})}$
$\boldsymbol{p}_{n+1} = \widetilde{\boldsymbol{r}_{n+1}} + \beta_n \boldsymbol{p}_n$
$\boldsymbol{p}_{n+1} = \widetilde{\boldsymbol{r}'_{n+1}} + \overline{\beta}_n \boldsymbol{p}'_n$
end

前処理を施した線形方程式系 (2) に対する双共役勾配法と数学的に同値なアルゴリズムがアルゴリズム７である。ここで、アルゴリズム７中の $\boldsymbol{K}$ は、$\boldsymbol{K} = \boldsymbol{M}_2\boldsymbol{M}_1$ で定義された行列である。アルゴリズム７では、$\boldsymbol{K}$ のみが定まれば、前処理付き双共役勾配法のアルゴリズムは前処理行列 $\boldsymbol{M}_1$, $\boldsymbol{M}_2$ には依存しない。また、これまでと同様に、行列 $\boldsymbol{K}$ は実際にメモリ上に保持する必要はなく、あるベクトル $\boldsymbol{v}$ に対する線形変換 $\boldsymbol{K}\boldsymbol{v}$ が行えれば、アルゴリズム７を実行することができる。このとき、計算効率の観点から、なるべく少ない計算量とメモリ量で計算できることが必要である。さらに、前処理付き双共役勾配法の収束性を高めるためには、$\boldsymbol{K}$ が $\boldsymbol{A}^{-1}$ の近似行列であることが望ましい。

2.4 一般化共役残差法

本節では、一般化共役残差法について説明する。一般化共役残差法は共役残差法を拡張した方法であり、正定値とは限らない一般の行列を係数行列に持つ線形方程式系を解く。

アルゴリズムの形状は異なるものの、この一般化共役残差法と一般化最小残差法は等しい反復点 x_n を生成することが知られている。また、一般化共役残差法や一般化最小残差法から派生した方法として、途中でリスタートを行うリスタート版一般化共役残差法やリスタート版一般化最小残差法があるが、それらの詳細については [2] などを参照されたい。一般化共役残差法を、アルゴリズム 8 で示した。

アルゴリズム 8 :
一般共役残差法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$$\alpha_n = \frac{(\boldsymbol{r}_n, \boldsymbol{A}\boldsymbol{p}_n)}{(\boldsymbol{A}\boldsymbol{p}_n, \boldsymbol{A}\boldsymbol{p}_n)}$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$$
$$\beta_{kn} = -\frac{(\boldsymbol{A}\boldsymbol{r}_{n+1}, \boldsymbol{A}\boldsymbol{p}_k)}{(\boldsymbol{A}\boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k)}$$
$$\boldsymbol{p}_{n+1} = \boldsymbol{r}_{n+1} + \sum_{k=1}^{n} \beta_{kn} \boldsymbol{p}_k$$
end

アルゴリズム 9 :
前処理付き一般共役残差法

$\boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{p}_0 = \boldsymbol{b},\ \boldsymbol{r}_0 = \boldsymbol{b}$
for $n = 0, 1, \cdots$
$$\alpha_n = \frac{(\boldsymbol{r}_n, \boldsymbol{M}_1^* \boldsymbol{M}_1 \boldsymbol{A}\boldsymbol{p}_n)}{(\boldsymbol{A}\boldsymbol{p}_n, \boldsymbol{M}_1^* \boldsymbol{M}_1 \boldsymbol{A}\boldsymbol{p}_n)}$$
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n \boldsymbol{p}_n$$
$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$$
$$\widetilde{\boldsymbol{r}_{n+1}} = \boldsymbol{K}\boldsymbol{r}_{n+1}$$
$$\beta_{kn} = -\frac{(\boldsymbol{A}\widetilde{\boldsymbol{r}_{n+1}}, \boldsymbol{M}_1^* \boldsymbol{M}_1 \boldsymbol{A}\boldsymbol{p}_k)}{(\boldsymbol{A}\boldsymbol{p}_k, \boldsymbol{M}_1^* \boldsymbol{M}_1 \boldsymbol{A}\boldsymbol{p}_k)}$$
$$\boldsymbol{p}_{n+1} = \widetilde{\boldsymbol{r}_{n+1}} + \sum_{k=1}^{n} \beta_{kn} \boldsymbol{p}_k$$
end

これまでと同様に、前処理を施した線形方程式系 (2) に対し一般共役残差法を適用するものと数学的に同値となる解法をアルゴリズム 9 で示す。アルゴリズム 9 中の $\boldsymbol{K}$ は $\boldsymbol{K} = \boldsymbol{M}_2\boldsymbol{M}_1$ で定義される行列である。アルゴリズム 7 の前処理付き双共役勾配法の場合と違い、線形変換 $\boldsymbol{K}\boldsymbol{v}$ の他に線形変換 $\boldsymbol{M}_1^*\boldsymbol{M}_1\boldsymbol{v}$ の計算が必要であることに注意する。つまり、前処理付き一般化共役残差法では、$\boldsymbol{K}$ だけでなく $\boldsymbol{M}_1$, $\boldsymbol{M}_2$ に依存して、アルゴリズムは変化する。

3 様々な前処理行列

本章では、幾つかの前処理について述べる。前章で述べたように、前処理行列 $\boldsymbol{K} = \boldsymbol{M}_2\boldsymbol{M}_1$ に求められている性質として、なるべく少ない計算量とメモリ量で線形変換 $\boldsymbol{K}\boldsymbol{v}$ が計算できることと、$\boldsymbol{K}$ が $\boldsymbol{A}^{-1}$ の近似行列であることが挙げられる。このことを踏まえた上で、以下の節では対角スケーリング、不完全コレスキー分解、多項式前処理について説明する。一般的に、良い前処理とは係数行列の性質や使用できるメモリの量などに強く依存するため、決定的な前処理法は存在しない。

3.1 対角スケーリング

一番簡単な前処理として、対角スケーリングがある。これは、行列 $\boldsymbol{A}$ の対角要素を抜き出した行列を $\boldsymbol{D}$ とすると、前処理行列 $\boldsymbol{K}$ として $\boldsymbol{D}^{-1}$ とすることに相当する。$\boldsymbol{D}$ が $\boldsymbol{A}$ の近似行列であるとみなせば、$\boldsymbol{K}$ は $\boldsymbol{A}^{-1}$ の近似行列といえる。一般的にこの前処理行列 $\boldsymbol{K}$ は $\boldsymbol{A}^{-1}$ のそれほど良い近似ではないため、収束性に関して大きな効果が期待できない。しかし、線形変換 $\boldsymbol{Kv}$ には m flops しかかからず、計算量が少ないため良く利用される。さらに、$\boldsymbol{A}$ が正定値なら $\boldsymbol{K}$ も正定値となる性質があるため、共役勾配法や共役残差法の前処理として利用できる。

3.2 不完全コレスキー分解

係数行列 $\boldsymbol{A}$ が正定値である場合によく使われる前処理法として不完全コレスキー分解がある。これは、係数行列 $\boldsymbol{A}$ のゼロの部分が分解した下三角行列 $\boldsymbol{L}$ でもゼロとなるように、近似的にコレスキー分解を行う方法である。不完全コレスキー分解は以下のようにして計算できる。

```
for k = 1, 2, ..., n
    L_kk = sqrt(L_kk)
    for i = k+1, k+2, ..., n        A_ik ≠ 0 の場合
        L_ik = L_ik / L_kk
    end
    for j = k+1, k+2, ..., n
        for i = j, j+1, ..., n      A_ij ≠ 0 の場合
            L_ij = L_ij - L_ik L_jk
        end
    end
end
```

この不完全コレスキー分解 $\boldsymbol{A} \approx \boldsymbol{LL}^*$ を利用し、前処理行列として $\boldsymbol{K} = \boldsymbol{L}^{-*}\boldsymbol{L}^{-1}$ と定める。このとき、$\boldsymbol{K} \approx \boldsymbol{A}^{-1}$ となるため、$\boldsymbol{K}$ は $\boldsymbol{A}^{-1}$ の近似行列とみなせる。下三角行列 $\boldsymbol{L}$ の構成法により、$\boldsymbol{A}$ と $\boldsymbol{L}$ は同程度に疎である。しかし、$\boldsymbol{L}^{-*}\boldsymbol{L}^{-1}(=\boldsymbol{K})$ は疎であるとは限らない。このため実用上は、疎な三角行列 $\boldsymbol{L}$ を保持した上で、線形変換 $\boldsymbol{Kv}$ を線形方程式系 $\boldsymbol{LL}^*\boldsymbol{z} = \boldsymbol{v}$ を解くことにより求める。この線形方程式系の解は、$\boldsymbol{L}$ が三角行列であるため $\boldsymbol{L}$ の疎性を直接利用して計算できる。この場合、$\boldsymbol{A}$ と $\boldsymbol{L}$ の疎性が等しいことを考慮すると、線形変換 $\boldsymbol{Av}$ と同じくらいの計算量で前処理が可能である。また、$\boldsymbol{L}$ が生成できれば $\boldsymbol{K}$ は正定値行列となるため、共役勾配法や共役残差法の前処理として利用できる。しかし、$\boldsymbol{A}$ が正定値でも不完全コレスキー分解ができるとは限らない。その場合、$\boldsymbol{A} + \epsilon\boldsymbol{I}$ を不完全コレスキー分解する方法などが提案されている。また、係数行列が正定値では無い場合、不完全コレスキー分解の変わりに不完全 LU 分解が使われることが多い [3]。

3.3 多項式近似

多項式近似とは、前処理行列 $\boldsymbol{K}$ は $\boldsymbol{A}$ の多項式で近似する方法である。一般に、$\boldsymbol{M}$ を正則行列としたとき、$\boldsymbol{G}=\boldsymbol{I}-\boldsymbol{M}^{-1}\boldsymbol{A}$ と定義すると、$\rho(\boldsymbol{G})<1$ ならば、

$$\boldsymbol{A}^{-1}=\boldsymbol{M}^{-1}+\boldsymbol{G}\boldsymbol{M}^{-1}+\boldsymbol{G}^2\boldsymbol{M}^{-1}+\boldsymbol{G}^3\boldsymbol{M}^{-1}+\cdots$$

が成り立つことが知られている [5]。これは無限級数展開であるが、これを t まで展開したものを前処理行列として利用することを考える。そのような行列を $\boldsymbol{K}_t$ とすると、つまり

$$\boldsymbol{K}_t=\boldsymbol{M}^{-1}+\boldsymbol{G}\boldsymbol{M}^{-1}+\boldsymbol{G}^2\boldsymbol{M}^{-1}+\cdots+\boldsymbol{G}^{t-1}\boldsymbol{M}^{-1}$$

と書くことができる。クリロフ部分空間法では、あるベクトル $\boldsymbol{v}$ に対する線形変換 $\boldsymbol{K}_t\boldsymbol{v}$ ができればよく、$\boldsymbol{K}_t$ は陽に行列を生成する必要はない。ここで、

$$\begin{aligned}\boldsymbol{K}_t\boldsymbol{v} &= (\boldsymbol{G}\boldsymbol{K}_{t-1}+\boldsymbol{M}^{-1})\boldsymbol{v}\\ &= \boldsymbol{M}^{-1}(\boldsymbol{r}-\boldsymbol{A}\boldsymbol{K}_{t-1}\boldsymbol{v})+\boldsymbol{K}_{t-1}\boldsymbol{v}\end{aligned}$$

という関係が成り立つため、

$$\boldsymbol{z}_0=\boldsymbol{v},\quad \boldsymbol{z}_k=\boldsymbol{M}^{-1}(\boldsymbol{r}-\boldsymbol{A}\boldsymbol{z}_{k-1})+\boldsymbol{z}_{k-1} \tag{5}$$

という漸化式を添字が t まで行うことにより、線形変換 $\boldsymbol{K}_t\boldsymbol{v}$ を求めることができる。つまり、(5) は定常反復法と見なすことが可能である。この計算を効率良く行うためには、行列 $\boldsymbol{M}^{-1}$ とベクトルの積を少ない計算量で行う必要がある。ここで、$\boldsymbol{A}$ の対角部分を抜き出した行列を $\boldsymbol{D}$、狭義下三角部分を抜き出した行列を $\boldsymbol{L}$、狭義上三角部分を抜き出した行列を $\boldsymbol{U}$ とする。すると、$\boldsymbol{M}$ として $\boldsymbol{D}$ を取るとこれはヤコビ法となる。また、$\boldsymbol{D}+\boldsymbol{L}$ あるいは $\boldsymbol{D}+\boldsymbol{U}$ を取ると、これはガウス・ザイデル法となる。さらに、$\frac{1}{\omega}\boldsymbol{D}+\boldsymbol{L}$ あるいは $\frac{1}{\omega}\boldsymbol{D}+\boldsymbol{U}$ を取ると、これは逐次過剰緩和法 (SOR 法) となる。このような特別な場合には、より効率よく線形変換 $\boldsymbol{K}_t\boldsymbol{v}$ を計算することができる。さらに [1] では、一般化共役残差法に対する可変的前処理という枠組みを提案している。その中では、前処理として SOR 法を用いた数値実験が行われている。SOR 法などを前処理として利用した場合、たとえ係数行列 $\boldsymbol{A}$ が正定値の場合でも、前処理行列 $\boldsymbol{K}$ の正定値性は失われる。この問題を解決するため、対称逐次過剰緩和法 (SSOR 法) などの方法が提案されている [5]。

4 係数行列が密の場合

クリロフ部分空間法は係数行列の疎性を崩さない方法であるため、特に係数行列が疎である線形方程式系に対し利用されることが多い。一方、係数行列が特別な構造を持っている場合、その構造を利用することにより係数行列が密である線形方程式系に対し効率良く働く可能性がある。本章では、半正定値計画問題を主双対内点法で解く場合、その途中で現れる線形方程式系に対しクリロフ部分空間法が有効であることを述べる。

4.1 半正定値計画問題とは

まず、$\mathcal{S}^n$ を $n \times n$ の実対称行列の集合と定義する。また、$\mathcal{S}^n_+$ を $n \times n$ の半正定値対称行列の集合と定義する。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。

$\boldsymbol{F}_i \in \mathcal{S}^n$ $(i = 0, 1, \ldots, m)$, $f_i \in \Re$ $(i = 1, 2, \ldots, m)$ とする。このとき、半正定値計画 (Semidefinite Programming: SDP) の主問題と双対問題は以下のように与えられる。

主問題：

$$\left.\begin{array}{ll} \text{最小化} & \boldsymbol{F}_0 \bullet \boldsymbol{X} \\ \text{制約条件} & \boldsymbol{F}_i \bullet \boldsymbol{X} = f_i \ (i = 1, 2, \ldots, m), \\ & \boldsymbol{X} \in \mathcal{S}^n_+. \end{array}\right\} \tag{6}$$

双対問題：

$$\left.\begin{array}{ll} \text{最大化} & \displaystyle\sum_{i=1}^{m} f_i y_i \\ \text{制約条件} & \displaystyle\sum_{i=1}^{m} \boldsymbol{F}_i y_i + \boldsymbol{Z} = \boldsymbol{F}_0, \\ & \boldsymbol{Z} \in \mathcal{S}^n_+. \end{array}\right\} \tag{7}$$

この問題は線形計画問題の対称行列の空間への拡張として捉えることができ、線形計画問題の主双対内点法を拡張した内点法で理論的に多項式時間内で解くことが可能である [7, 10, 11]。この主双対内点法では、探索方向の計算過程である種の Newton 法を使用しており、このとき、サイズが m(主問題の線形制約の数) の線形方程式系

$$\boldsymbol{A}d\boldsymbol{y} = \boldsymbol{b} \tag{8}$$

を解く必要が生ずる。ここで、線形方程式系 (8) の係数行列 $\boldsymbol{A}$ の各要素は、

$$A_{ij} = \boldsymbol{F}_i \bullet \boldsymbol{X}\boldsymbol{F}_j\boldsymbol{Z}^{-1} \qquad (i, j = 1, 2, \ldots, m),$$

と定義される。この係数行列 $\boldsymbol{A}$ に関し、重要な 2 つの性質が知られている [8]。一つは、係数行列 $\boldsymbol{A}$ が正定値行列になっていることであり、もう一つは、一般に行列 $\boldsymbol{X}, \boldsymbol{Z}$ が密であるため $\boldsymbol{F}_i (i = 1, 2, \ldots, m)$ が疎行列であっても、係数行列 $\boldsymbol{A}$ は密行列となることである。このため、主問題の線形制約の数 m が非常に多い SDP 問題を解く場合、非常に大きな次元の係数行列が密な線形方程式系を解く必要が生ずる。このため、直接法では計算量やメモリ使用量の制約からこの大規模線形方程式系の解の計算が困難となる。そこで、主双対内点法の各反復で解く線形方程式系 (8) に共役勾配法を適用することを試みる。

4.2 係数行列とベクトルの計算

共役勾配法では、係数行列 $\boldsymbol{A}$ はベクトル $\boldsymbol{p}$ に対する線形変換にのみ利用する。この線形変換 $\boldsymbol{Ap}$ となるベクトルを $\boldsymbol{q}$ とすると、任意の i に対して、

$$\begin{aligned} q_i &= \sum_{j=1}^{m} A_{ij}p_j = \sum_{j=1}^{m} \boldsymbol{F}_i \bullet \boldsymbol{X}\boldsymbol{F}_j\boldsymbol{Z}^{-1}p_j \\ &= \sum_{j=1}^{m} \mathrm{Tr}\,(\boldsymbol{F}_i\boldsymbol{X}\boldsymbol{F}_j\boldsymbol{Z}^{-1})p_j \\ &= \mathrm{Tr}\,(\boldsymbol{F}_i\boldsymbol{X}\sum_{j=1}^{m}\boldsymbol{F}_jp_j\boldsymbol{Z}^{-1}) \\ &= \boldsymbol{F}_i \bullet \boldsymbol{X}\sum_{j=1}^{m}\boldsymbol{F}_jp_j\boldsymbol{Z}^{-1} \end{aligned}$$

が成り立つ。よって、以下のような手順で線形変換 $\boldsymbol{Ap}$ の計算を行うことができる。

$$\boldsymbol{H}_1 = \sum_{j=1}^{m} p_j\boldsymbol{F}_j,\ \boldsymbol{H}_2 = \boldsymbol{X}\boldsymbol{H}_1\boldsymbol{Z}^{-1},\ q_i = \boldsymbol{F}_i \bullet \boldsymbol{H}_2\ (i = 1, 2, \ldots, m) \tag{9}$$

この計算方法では、計算量としてほぼ大きさ $n \times n$ の行列同士のかけ算2回分となる。$m = \mathcal{O}(n^2)$ であることを考慮すると、この計算は $\mathcal{O}(m^{1.5})$flops と見積もることができる。係数行列 $\boldsymbol{A}$ は m^2 個の (ゼロでない) 要素を持つのに対し、係数行列 $\boldsymbol{A}$ の構造を利用することにより線形変換を $\mathcal{O}(m^{1.5})$flops で計算することが可能となった。また、$m \times m$ の密行列 $\boldsymbol{A}$ を保持せず線形変換 $\boldsymbol{Ap}$ を計算している。

4.3 多項式近似を利用した前処理

次に、共役勾配法に対する前処理について考察する。線形方程式 (8) の係数行列 $\boldsymbol{A}$ は密であるため、前処理として不完全コレスキー分解を利用することはできない。前節では $\boldsymbol{A}$ の構造を利用することにより線形変換 $\boldsymbol{Ap}$ を効率よく計算する方法を説明したが、前処理においても $\boldsymbol{A}$ の構造を利用したい。このため、3.3節で説明した多項式近似を利用することが適切であると思われる。前処理としてヤコビ法を利用する場合、前節で述べた計算法を直接利用することにより、高速に前処理を行うことができる。しかし、前処理としてSOR法やSSOR法を利用する場合、前節と同様の方法では計算できない。このため以下の工夫を行う。

SOR法の k 反復目の反復点 $\boldsymbol{x}$ の i 番目の要素を $x_i^{(k)}$ とする。このとき、

$$\begin{aligned} x_i^{(k+1)} &= \sum_{j=1}^{i-1} A_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{m} A_{ij}x_j^{(k)} \\ &= \sum_{j=1}^{i-1}(\boldsymbol{F}_i\boldsymbol{Z}^{-1} \bullet \boldsymbol{X}\boldsymbol{F}_j)x_j^{(k+1)} + \sum_{j=i+1}^{m}(\boldsymbol{F}_i\boldsymbol{Z}^{-1} \bullet \boldsymbol{X}\boldsymbol{F}_j)x_j^{(k)} \\ &= \boldsymbol{F}_i\boldsymbol{Z}^{-1} \bullet (\boldsymbol{X}(\sum_{j=1}^{i-1} x_j^{(k+1)}\boldsymbol{F}_j) + \boldsymbol{X}(\sum_{j=i+1}^{m} x_j^{(k)}\boldsymbol{F}_j)) \end{aligned}$$

が成り立つ。ここで、$\boldsymbol{H}_i = \boldsymbol{X}(\sum_{j=1}^{i-1} x_j^{(k+1)} \boldsymbol{F}_j) + \boldsymbol{X}(\sum_{j=i+1}^{m} x_j^{(k)} \boldsymbol{F}_j)$ と定義すると、この $\boldsymbol{H}_i$ は $\boldsymbol{H}_i = \boldsymbol{H}_{i-1} + x_{i-1}^{(k+1)} \boldsymbol{X}\boldsymbol{F}_{i-1} - x_i^{(k)} \boldsymbol{X}\boldsymbol{F}_i$ という漸化式で書き表せる。この事実から、以下のような手順により SOR 法を実行することができる。

$\boldsymbol{x} = \boldsymbol{0}, \boldsymbol{H} = \boldsymbol{O}$
for $k = 0, 1, \ldots$
  for $i = 1, 2, \ldots, m$
    $\boldsymbol{H} = \boldsymbol{H} + x_{i-1}^{(k+1)} \boldsymbol{X}\boldsymbol{F}_{i-1} - x_i^{(k)} \boldsymbol{X}\boldsymbol{F}_i$
      (もし $i = 1$ のとき $\boldsymbol{H} = \boldsymbol{H} + x_m^{(k)} \boldsymbol{X}\boldsymbol{F}_m - x_1^{(k)} \boldsymbol{X}\boldsymbol{F}_1$)
    $\boldsymbol{J} = \boldsymbol{F}_i \boldsymbol{Z}^{-1}$
    $x_i^{(k+1)} = \omega(b_i - \boldsymbol{J} \bullet \boldsymbol{H})/A_{ii} + (1 - \omega) x_i^{(k)}$
  end
end

このアルゴリズムの詳細については [9] を参照されたい。結果として、SOR 法の 1 反復を $\mathcal{O}(n^3)$ で計算することができる。これは、$\boldsymbol{Ap}$ の計算にかかる計算量と同等であり、実用的な前処理となる。

謝辞

本研究をするにあたり、東京大学工学系研究科の張紹良助教授と新田義寛さんの助力を得ました。ここに、感謝の意を表します。

参考文献

[1] 阿部邦美, 張紹良, 長谷川秀彦, 姫野龍太郎, SOR 法を用いた可変的前処理付き一般化共役残差法, 応用数理学会 論文誌, 第 1 1 巻, 4 号 (2001) 157–170.

[2] R. Barrett, M. Barry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Römine, and H. van der Vorst, *Templates for the Solution of Linear Systems:Building Blocks for Iterative Methods* (Society for Industrial and Applied Mathematice, 1994)

[3] 藤野 清次, 張 紹良, 反復法の数理, (朝倉書店, 1996)

[4] K. Fujisawa, M. Fukuda, M. Kojima and K. Nakata, Numerical evaluation of SDPA (SemiDefinite Programming Algorithm), in: H. Frenk, K. Roos, T. Terlaky and S. Zhang, eds., *High Performance Optimization* (Kluwer Academic Publishers, Dordrecht, 1999) 267–301.

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

[6] M. Hestenes and E. Stiefel, Methoeds of conjugate gradients for solving linear systems, *Journal Research of the National Bureau of Standards*, 49 (1952) 409–436.

[7] 小島政和, 半正定値計画とその組合せ最適化への応用, 離散構造とアルゴリズム 5, (近代科学社, 1998).

[8] 中田 和秀, 藤沢 克樹, 小島 政和, 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 (1998) 297–316.

[9] 新田 義寛, 半正定値計画で現れる線形方程式系に対する前処理付き共役残差法, 東京大学 物理工学専攻 修士論文, (2002).

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

[11] H. Wolkowicz, R. Saigal and L. Vandenberghe, eds., *Handbook of Semidefinite Programming, Theory, Algorithms, and Applications* (Kluwer Academic Publishers, Massachusetts, USA, 2000).

[12] S.-L. Zhang, K. Nakata and M. Kojima, Incomplete orthogonalization preconditioner for solving large and dense linear systems which arise from semidefinite programming, *Applied Numerical Mathematics*, 41 (2002) 235–245.