半正定値計画問題に対する主双対内点法の実装と実験的解析

藤沢 克樹
東京工業大学 情報理工学研究科
数理・計算科学専攻

目 次

図 目 次

表 目 次

第 1 章

はじめに

コンピューターやネットワークを中心とした関連技術の発達は脱兎のごとく速く、質的にも量的にも大きく拡大しているので、一度その進歩に乗り遅れれば最先端の研究はもちろんのこと、各個人の必要限度の目的のためでさえ使いこなすのが困難である．しかし、多く使用者 (ユーザー) にとってコンピューターなどは単なる手段であって、それ自体が目的ではないのも事実である．使用目的、つまり社会からのコンピューターに対する要請とコンピューターの関連技術の進歩が密接に関係していることはいうまでもない．コンピューターを電子計算機と訳すこともあるように、最初は明らかに人間では手に負えないような高度な計算を代行させるのが目的であった．今ではコンピューターに必須の機能になりつつある動画や音声を表示したり、CG で映画を作成することまでは (すくなくとも短期的には) 念頭においていなかったと思われる．非常に高速に計算が実行できるようになれば、今度は人間の方でも様々な応用を念頭に置いた使用方法を発案していくのは当然の流れであって、線形計画問題 (LP) と最初の電子計算機 ENIAC が生み出されたのがほぼ同時期であったというのは決して無関係ではない．理論的には、多項式時間で終了するアルゴリズムは高速な部類に含まれるが、実際にコンピューター上に実装して、時代の要請に合致した大きさや難しさの問題を解くのは容易ではないことは、これまで多くの研究者が体験してきた通りである．しかし、コンピュータの能力の制約から場合によっては数十年も眠っていたような手法が再び脚光を浴びるようになったいう事例が最近では多くの工学的分野で起こっている．本論文で扱う半正定値計画問題 (SDP) は、LP の拡張から生まれた数理計画問題である．SDP も理論的には Yudin と Nemirovsky [56] が 1977 年に楕円体法によって多項式時間で解けることを示しているが、速度が遅く当時のコンピュータの能力からいっても実用に耐えるアルゴリズムではなかった．もっともこの手法は現在のコンピューターを持ってしても、お世辞にも高速に解けるとは言い難い．それでも多項式時間で解けることを示しただけでも、非常に画期的であったことは間違いない．それは SDP が LP や 凸二次計画問題 (QP) などの多くの凸計画問題を含んだ大きな枠組みであるからである．

しかし、SDP だけでなく凸計画問題全体にとっても大きな突破口 (breakthrough) が現れることになった．1984 年に Karmarker [22] によって提案された内点法である．内点法の思想やアイデア自体はさらに古くから存在していたが、数理計画や OR とは全く異なる分野でも報道され、数理計画の宣伝効果としても刮目すべきものがあった．そして LP の分野で、内点法 (特に主双対内点法) が理論的にも実用的にも成功を収めたあと、SDP にも発展

的に適用されて現在までに理論的には目覚ましい成果が挙げられている．その幾つかをここで紹介すると、線形計画問題の主双対内点法の SDP への拡張 ([1, 25, 26, 35, 38, 52])、組合せ最適化問題への応用 ([1, 16, 17, 29])、さらにシステムと制御への応用 ([6, 18, 52]) などである．特にさかんに研究されるようになったのは 1993 年頃からであるが、コンピューターに携わっていた者としては、まさに時期を得たものであったと言える．なぜならば、もしさらに数年前に研究が開始されていたとしても、当時のコンピューターの能力では小規模な問題しか解けずに、おそらく実験的に有益な評価することは無理であったと考えるからである．そうした CPU の高速化や記憶装置の大容量化といった追い風を受けているので、理論的有効性だけを示して研究を終わるのではなく、コンピューターに実装してなるべく現実に即した、あるいは応用を指向した形で数値実験等を行い、実用上の有効性まで調査するのが最近の研究の風潮になりつつあり、大変好ましいことである．最近では、SDP に対するソフトウェアの開発と数値実験を行う研究も幾つか存在するようになっている ([10, 11, 19, 50])．しかし大規模かつ系統的な数値実験はこれから行われるものと思われる．よって本論文の主目的は著者らが作成したソフトウェア SDPA ([10]) を用いて系統的な数値実験及び SDPA の実験的解析を行うことにある．

SDP に対する主双対内点法のアルゴリズムを実現したソフトウェア SDPA を作成するにあたって特に留意したことは、なるべく時代の要請に即したソフトウェアにするということである．次にそれらの点についてまとめてみたい．

1. 大規模な問題を高速に解くこと:
   当然、１番目にあげられるべき目標である．通常数値実験では大きく分けて二つの制約に常に悩まされる．一つ目は主記憶 (メモリ) の制約であり、二つ目は実行時間の制約である．しかも多くの場合には両方の対策を同時に行うのは困難な場合が多い．基本的には前者に対する対策は必要のないものはメモリに保持しない、また後者に対しては必要のないものは計算しないようにすることである．SDPA では、前者や後者に対して様々な工夫を組み入れている．本論文において工夫や対策は主に 3 章に、その効果の検証は 4 章に記述した．

2. ソフトウェアを広く公開すること:
   この公開には、様々な長所がある．SDP の知名度が上がるにつれて従来とは異なった分野、例えば建築や生物分野などでも SDP として定式化できる問題が発見されて研究がおこなわれている．理論的に SDP として定式化てきることがわかったとしても、SDP に対するソフトウェアが世に出ていなければ、研究はそこで中断してしまうことになる．また ユーザーが増えれば作成者にとっても非常に利益が大きい．また公開する別の理由は、最近では特に科学技術の進歩が加速しているので、大学や研究機関等で新しい研究を行っても、インターネット等を通じてソフトウェアを配布したり、論文等を公開したりしない限り、すぐに陳腐化してしまうことが多い。インターネットは、それが開放的で相互的な接続である以上セキュリティなどの問題は常に存在する．しかし積極的に活用するにしても、必要に迫られて使用するにしても、インターネットの使用によって利益を受けている人にはもはや不可欠なものになりつつある。そのため、ハッカーやコンピュータウィルスの問題などは、いたちごっこになるにしろ、前

向きな解決しかありえないのであって、研究に携わる人達は、こうしたコンピュータやネットワーク技術と積極的に付き合っていくのが懸命であると考える．なお SDPA などの SDP に対するソフトウェアの入手方法は、付録 B を参照していただきたい．

3. 使用しやすいインターフェイスやマニュアルの作成:
   最近では PC からワークステーションまで GUI を採用してインターフェイスの向上に努めている．しかし大学等の研究機関ではアルゴリズムの開発から高度なインターフェイスの作成まで一括して行うのは難しいので、研究機関とソフトウェアハウスとの提携なども能率の向上という面では好ましいと考えられる．SDPA のユーザーマニュアルを付録 A に載せてある．なおインターネット上から入手できるマニュアル [10] は英語で書かれているが両者の内容はほぼ同じである．

SDP などの最適化に関連する技術も今後さらに必要度が高まっていくことが予想される．それには、次のような時代背景がある．インターネットなどの情報通信技術革命によって、完全自由市場の必須条件の一つである完全情報性 (市場関係者 (いいかえれば市場への参加者) の誰もが、同時に、同等の情報を得ることが出来ること) はより高まったのである．結局、これらの情報通信革命は世界市場の完全情報性を高め、アダム・スミスが提唱した完全自由競争市場の実現へ近付けていくものといえる．全てを市場原理に委ねてよいかという論争は当然存在するとして、主に非製造業の部分で、アメリカが作成したグローバルスタンダードが事実上、世界標準となっていくのは、もはや止めようのない時代の流れである．しかし完全自由競争とは結局、弱肉強食、適者生存の厳しい現実であることも重く受け止めていくべきである．

そこでこの流れの当然の帰結として、市場への参加者、主に企業では経営、生産そして販売計画などにおいて最適化などを行い、効率の良い計画を作成することが求められる．つまり、先に述べた市場優先主義において、資源の最適配分や費用最小化の生産計画や利益最大化の販売計画などの最適化問題を解くことが、競争を生き抜く上で以前にも増して重要になることは言うまでもない．さらに、これまでの最適化問題は静的 (つまり時間軸を考慮しない) な場合が多かったが、コンピュータネットワークの発達によって、リアルタイムに条件が変化する問題も考慮しなければならなくなるであろう．時間の概念を納期や利子の概念と置き換えても構わない．旧ソ連は、世界最大の国土、豊富な天然資源や肥沃な穀倉地帯を持ち、いわゆる自給自足経済圏 (アウタルーキー) でありながら、その経済体制が崩壊した原因として、時間の概念が無かったことがその一つとして挙げられている．お金を何年も借りていても利子がかからない、また納期を破っても罰金も無い。こういった社会でまともに経済が動くはずもなく、ましてやカンバン方式や最適化の概念が出てくるはずもない．よって、最適化手法の研究も非常に時代の要請に合致したものと言えよう．本論文では SDP を扱ったが、この研究より得られた経験や知見等は他の最適化手法を研究する際にも十分役に立つものと思われる．

最後に本論文の各章の構成や内容について述べる．本論文では、半正定値計画法 (SemiDefinite Programming : SDP) に対する主双対内点法を C++ 言語を用いて記述したソフトウェア SDPA の実装方法や関連するアルゴリズムの説明、および実験的解析により SDPA の評価を行う．

1. SDP と 主双対内点法 (2 章)
   SDP の定義や過去の研究、主双対内点法の SDP への適用方法などについて述べる.

2. 主双対内点法の実装方法 (3 章)
   SDPA は入力データが疎行列を含む場合には、その構造を活用して省メモリ化や高速化を行う. この章ではその目的のためのデータ構造や最も実行時間を消費する探索方向の計算の高速化などについて述べる.

3. SDPA の実験的解析 (4 章)
   3 章で説明した計算手法の有効性を数値実験を通して検証する. また SDP の代表的な応用問題などを使用して SDPA の系統的な数値実験を行い、他の SDP のソフトウェアと比較実験を行うと共に、入力問題の大きさや行列の密度などによる SDPA の性能の変化を調査する.

4. SDPA のライブラリ (5 章)
   SDPA をライブラリ化して他のソフトウェアから繰り返して呼べるようにする計画が存在するので、この章では SDPA のライブラリについて述べる.

5. 結論と今後の課題 (6 章)

6. SDPA の使用方法 (付録 A)
   SDPA の使用方法 (入力ファイルの書式、SDPA のパラメータ及び出力ファイルの見方など) について説明する.

7. SDPA の変遷および入手方法 (付録 B)
   今までの SDPA の改善内容や追加さらた機能等を説明して、SDPA などの SDP に対するソフトウェアの入手方法について述べる.

第 2章

半正定値計画問題 (SDP) と主双対内点法

本章では SDP の問題の定義を行った後、SDP に対する主双対内点法アルゴリズムの説明を行う.

2.1 半正定値計画問題 (SDP) とその定義

数理計画において最も基本的なモデルであり、応用面においても頻繁に使用されるのが線形計画法 (Linear Programming : LP) である．LP とは、有限次元の変数ベクトルに関する線形の実数値関数 (目的関数) を、有限個の線形等式条件および線形不等式条件の下で最小化 (または、最大化) する問題である．そして SDP とは、LP の主問題及び双対問題を対称行列の空間へ拡張したものとみなすことができる．言い換えれば、対称行列の線形結合が半正定値であるという条件の下で、線形関数を最小化 (または最大化) する問題である．この制約は、非線形であるが凸制約である．つまり、SDP は凸計画法の一つであり LP や 凸二次計画法などを含んでいる．次に SDP の等式標準形を示す．この論文を通して 主問題 $\mathcal{P}$ を最小化問題、また双対問題 $\mathcal{D}$ を最大化問題として扱う.

$$
\left.
\begin{array}{lll}
\mathcal{P}: & \text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\
& \text{subject to} & \boldsymbol{A}_i \bullet \boldsymbol{X} = a_i \ (1 \leq i \leq m),\ \boldsymbol{X} \succeq \boldsymbol{O}. \\
& & \boldsymbol{X} \in \mathcal{S}. \\
\mathcal{D}: & \text{maximize} & \displaystyle\sum_{i=1}^{m} a_i y_i \\
& \text{subject to} & \displaystyle\sum_{i=1}^{m} \boldsymbol{A}_i y_i + \boldsymbol{Z} = \boldsymbol{A}_0,\ \boldsymbol{Z} \succeq \boldsymbol{O}. \\
& & \boldsymbol{Z} \in \mathcal{S}.
\end{array}
\right\} \tag{2.1}
$$

$\mathcal{S}$: $n \times n$ 対称行列の空間,

$\boldsymbol{A}_i \in \mathcal{S}$ $(i = 0, 1, 2, \ldots, m)$: 定数行列,

$\boldsymbol{O} \in \mathcal{S}$: 零行列,

$$\boldsymbol{a} = \begin{pmatrix} a_1 \\ a_2 \\ \cdot \\ a_m \end{pmatrix} \in R^m \ : \text{定数ベクトル},$$

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ y_m \end{pmatrix} \in R^m \ : \text{変数ベクトル},$$

$\boldsymbol{X} \in \mathcal{S},\ \boldsymbol{Z} \in \mathcal{S}$: 変数行列,

$\boldsymbol{U} \bullet \boldsymbol{V}$: $\boldsymbol{U},\ \boldsymbol{V}$ の内積,

$$\boldsymbol{U} \bullet \boldsymbol{V} = \mathrm{Tr}(\boldsymbol{U}^T\boldsymbol{V}) = \sum_{i=1}^{n}\sum_{j=1}^{n} U_{ij}V_{ij}$$

$\boldsymbol{U} \succeq \boldsymbol{O}, \boldsymbol{U} \in \mathcal{S} \iff \boldsymbol{U} \in \mathcal{S}$ が半正定値

$\boldsymbol{U} \succ \boldsymbol{O}, \boldsymbol{U} \in \mathcal{S} \iff \boldsymbol{U} \in \mathcal{S}$ が正定値

行列 $\boldsymbol{U}$ が正定値であるとは、つまり $\forall \boldsymbol{x} \in R^n$ に対して $\boldsymbol{x}^T\boldsymbol{U}\boldsymbol{x} \geq 0$ が成り立つことである．また $\boldsymbol{U}$ が半正定値であるということは、$\forall \boldsymbol{x} \in R^n, \boldsymbol{x} \neq 0$ に対して $\boldsymbol{x}^T\boldsymbol{U}\boldsymbol{x} > 0$ が成り立つことである．また、対称行列の場合には全ての固有値が非負 (半正定値の場合), または正 (正定値の場合) といった定義もある．よって最小固有値の値を調べることによって $\boldsymbol{U}$ が半正定値かどうか調べることができる．

$\boldsymbol{X}$ が主問題の実行可能解（または，最小解）であり，$(\boldsymbol{y}, \boldsymbol{Z})$ が双対問題の実行可能解（または，最大解）であるとき，$(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ は SDP の実行可能解（または，最適解）である．以下では，SDP の主問題と双対問題を一まとめにして扱うことが多い．$(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ が SDP のすべての制約条件を満たすとき，許容解であるという．さらに，許容解 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ が条件 $\boldsymbol{X} \succ \boldsymbol{O},\ \boldsymbol{Z} \succ \boldsymbol{O}$ を満たすとき，内点許容解と呼ぶ．また，$\boldsymbol{X}$ が 主問題の最小解で，$(\boldsymbol{y}, \boldsymbol{Z})$ が双対問題の最大解であるとき，$(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ は SDP の最適解であるという．$(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ を SDP の許容解とすると，主問題の目的関数値 $\boldsymbol{A}_0 \bullet \boldsymbol{X}$ と双対問題の目的関数値 $\sum_{i=1}^{m} a_i y_i$ の間に、不等式

$$\boldsymbol{X} \bullet \boldsymbol{Z} = \boldsymbol{A}_0 \bullet \boldsymbol{X} - \sum_{i=1}^{m} a_i y_i \geq 0 \tag{2.2}$$

が常に成り立つ．したがって，それらの目的関数値が一致すれば, あるいは，$\boldsymbol{X} \bullet \boldsymbol{Z} = 0$（相補性条件）が成り立てば，$(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ は SDP の最適解であることが分かる．以下の双対定理はこの逆を保証している．

定理 2.1.1 [37] SDP に内点許容解が存在すると仮定する（Slater の制約想定 [33]）．このとき，

1. SDP に最適解が存在する．
2. SDP の許容解 $(\boldsymbol{X}^*, \boldsymbol{y}^*, \boldsymbol{Z}^*)$ が最適解であるための必要十分条件は主問題と双対問題の目的関数値が一致することである．すなわち，

$$\boldsymbol{X}^* \bullet \boldsymbol{Z}^* = \boldsymbol{A}_0 \bullet \boldsymbol{X}^* - \sum_{i=1}^{m} a_i y_i^* = 0$$

この双対定理は SDP の理論の中核をなしている．

なおこの論文を通して、定数行列 $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ は線形独立であることを仮定する．このことはつまり $m \leq n(n+1)/2$ であることを示している．

2.2 過去の研究

SDP の理論的研究が始められたのは意外に古く、初期の論文の一つとして 1963 年の Bellman と Fan [4] がある．さらに SDP の最適性条件に関しては、1980 年代から研究が進められた ([8, 47, 54, etc.])．また対称行列の最大固有値の最小化問題は SDP として定式化して解くことができる．そしてこの固有値問題も多く研究がなされている ([15, 40, 41, etc.])．

また 1984 年に Karmarkar [22] によって多項式解法である内点法が LP に導入された (LP に対する多項式解法はさらに古く [9] などがある)．この内点法の最悪計算量が次数の低い多項式で押さえられるだけでなく、実用的にも優れた面を持っていた．この研究を機にして、LP に対する内点法の研究はさかんに行われるようになった．また内点法は LP だけでなく、その他の凸計画法 (例えば二次計画法や線形相補正問題 [24] など) にも適用された．

1988 年には Nesterov と Nemirovsky [36, etc.] が LP に対する内点法を全ての凸計画問題へと一般化できることを示し、さらに彼らは全ての凸集合に自己同一性 (self-concordance) を持つバリア関数が存在することを示した．SDP は、比較的容易に計算することができるバリア関数を持つ問題のクラスに属しているので、その結果内点法が適用されていった．

その後幾つかの独立したグループによって、LP の内点法は SDP へと拡張された ([1, 25, 26, 35, 38, 52])．また、SDP に対する主・双対内点法を記述したソフトウェアも幾つか存在する ([10, 11, 19, 50])．また SDP 関連のサーベイには ([53]) などが存在する．

また SDP の応用に関する理論的な研究も多く行われている．大きく分けて組合せ最適化への応用 ([1, 16, 17, 29]) とシステムと制御への応用 ([6, 18, 52]) である．

2.3 主双対内点法の SDP への拡張

主双対内点法（[23, 34, 46]）は，Karmarkar 法 [22] 以後に発展した内点法のなかで，理論的にも実用的にも最も優れた方法であるといえる．

非常に多くの研究がなされており，超大規模な線形計画問題を高速に解く計算手法として定着している．日本語の文献としては [55] 等がある．文献 [26] で主双対内点法の基本構造を SDP に拡張している．

この方法で主要な役割を果たしているのは内点許容領域の内部を通って最適解に収束する “中心パス（center path, central trajectory）” である．

SDP に対しては,

$$F_{cen} = \{(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) \in F_{++} \ : \ \boldsymbol{X}\boldsymbol{Z} = \mu \boldsymbol{I} \ \ \exists \mu > 0\}$$

で定義される．ここで,

$$F_{++} = \left\{ (\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) : \begin{array}{l} \boldsymbol{A}_i \bullet \boldsymbol{X} = a_i \ (i = 1, 2, \ldots, m), \boldsymbol{X} \succ \boldsymbol{O}, \\ \boldsymbol{Z} = \boldsymbol{A}_0 - \sum_{i=1}^{m} \boldsymbol{A}_i y_i, \boldsymbol{Z} \succ \boldsymbol{O}, \end{array} \right\}$$

は SDP の内点許容領域を表す.

以下に中心パスの性質を示す.

- $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) \in F_{cen}$ であれば，$\boldsymbol{X}\boldsymbol{Z} = \mu \boldsymbol{I}$ なるパラメータ $\mu > 0$ は $\mu = \boldsymbol{X} \bullet \boldsymbol{Z}/n$ で与えられる.
- $F_{++} \neq \emptyset$ のもとで，中心パス F_{cen} は内点許容領域内の滑らかな曲線となり，$\mu \to 0$ のとき SDP の最適解に収束する.

主双対内点法ではパラメータ $\mu > 0$ を減少させながら，中心パス F_{cen} を数値的に追跡し，SDP の近似最適解に到達する.

次に、SDP に対する主双対内点法アルゴリズムの概要を示す.

手順０． まず、条件 $\boldsymbol{X}^0 \succ \boldsymbol{O}, \ \boldsymbol{Z}^0 \succ \boldsymbol{O}$ を満たす初期点 $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ を選ぶ (許容解でなくてもよいことに注意).

手順１． 方向探索パラメータ $\beta \in [0, \ 1)$ を選んで（例えば $\beta = 0.1$），$\mu = \beta \boldsymbol{X}^k \bullet \boldsymbol{Z}^k/n$ とおく.

手順２． パラメータ $\mu > 0$ をもつ中心パス上の点

$$(\boldsymbol{X}^k + d\boldsymbol{X}^k, \boldsymbol{y}^k + d\boldsymbol{y}^k, \boldsymbol{Z}^k + d\boldsymbol{Z}^k) \in F_{cen}$$

を近似するためのニュートン方程式をたてる.

$$\left. \begin{array}{l} \boldsymbol{A}_i \bullet (\boldsymbol{X}^k + d\boldsymbol{X}^k) = a_i \ (i = 1, 2, \ldots, m), \\ (\boldsymbol{Z}^k + d\boldsymbol{Z}^k) = \boldsymbol{A}_0 - \sum_{i=1}^{m} \boldsymbol{A}_i (y_i^k + dy_i^k), \\ \boldsymbol{X}^k \boldsymbol{Z}^k + \boldsymbol{X}^k d\boldsymbol{Z}^k + d\boldsymbol{X}^k \boldsymbol{Z}^k = \mu \boldsymbol{I}, \\ d\boldsymbol{X}^k \in \mathcal{S}, \ d\boldsymbol{Z}^k \in \mathcal{S}, \ d\boldsymbol{y}^k \in R^m. \end{array} \right\} \tag{2.3}$$

手順３． ニュートン方程式を解き，探索方向ベクトル $(d\boldsymbol{X}^k, d\boldsymbol{y}^k, d\boldsymbol{Z}^k)$ を計算する．探索方向を求める方法は複数存在するが、本論文で考慮する探索方向は 2.4 節にて説明する．(手順１で探索方向パラメータ $\beta \in [0, \ 1)$ を 1 に近く取るほど，探索方向ベクトル $(d\boldsymbol{X}^k, d\boldsymbol{y}^k, d\boldsymbol{Z}^k)$ は中心パス方向を向き，0 に近く取るほど SDP の最適解の方向を向く.）

手順４． 新しい反復点 $(\boldsymbol{X}^{k+1}, \boldsymbol{y}^{k+1}, \boldsymbol{Z}^{k+1})$ を条件

$$\begin{aligned} \boldsymbol{X}^{k+1} &= \boldsymbol{X}^k + \alpha_p d\boldsymbol{X}^k \succ \boldsymbol{O}, \\ \boldsymbol{Z}^{k+1} &= \boldsymbol{Z}^k + \alpha_d d\boldsymbol{Z}^k \succ \boldsymbol{O}, \end{aligned}$$

を満たすように定める．ただし，$\alpha_p, \alpha_d > 0$ はステップ長.（ステップ長をあまり大きく取ると新しい反復点が境界に近づきすぎて，以後の反復で困難を生ずる．また，小さすぎると収束までに多くの反復回数を要する.）

方向探索パラメータ β とステップ長 α_p, α_d を適当に選ぶことにより，大域的な収束性を保証することができる．また，ポテンシャル関数を評価関数として組み合わせて使うことも可能である．

2.4 探索方向について

2.3 節において、等式 2.3 を解いて探索方向を求める方法が複数存在すると述べたが、現在では 20 以上の探索方向が知られている．本論文では多くの 主双対内点法に使用されている探索方向 [1, 2, 19, 27, 26, 28, 30, 35, 38, 39, 44, 49, 57, etc.] の中から主として３つを取り上げる．一つ目は HRVW/KSH/M 方向であり、この方向は２つの研究グループ (Helmberg-Rendl-Vanderbei-Wolkowicz [19] と Kojima-Shindoh-Hara [26]) によって独立に提案された．そして後に Monteiro [35] よって再発見されている．二つ目は Nesterov と Todd [38, 39] によって提案された NT 方向である．さらに最後の探索方向が Alizadeh, Haeberly と Overton [1] によって提案された AHO 方向である．またこれらの HRVW/KSH/M, NT そして AHO 方向を使用した 主双対内点法の大域的かつ局所的収束性が多くの論文で研究されている [1, 2, 19, 27, 26, 28, 30, 35, 38, 39, 44, 57, etc.]．これらの３つの探索方向を使用した数値実験結果も幾つか発表されており [1, 2, 11, 12, 19, 49, etc.], また、これらの探索方向を組み込んだ SDP を解くためのソフトウェアも現在インターネットより公開されている [7, 10, 50]．この節では、本論文の中心となるソフトウェア SDPA[10] に組み込まれている三つの探索方向 (HRVW/KSH/M,NT,AHO) について解説する．

最初に、HRVW/KSH/M 方向について述べる．HRVW/KSH/M 方向は次の３つの等式の解 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ である．

$$\boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i \ (1 \leq i \leq m), \ d\boldsymbol{X} \in \mathcal{S}, \tag{2.4}$$

$$\sum_{i=1}^{m} \boldsymbol{A}_i dy_i + d\boldsymbol{Z} = \boldsymbol{D}, \ d\boldsymbol{Z} \in \mathcal{S}, \tag{2.5}$$

$$\widehat{d\boldsymbol{X}}\,\boldsymbol{Z} + \boldsymbol{X}\,d\boldsymbol{Z} = \boldsymbol{K}', \ \widehat{d\boldsymbol{X}} \in R^{n\times n}, \ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^T)/2. \tag{2.6}$$

ここで、$\widehat{d\boldsymbol{X}} \in R^{n\times n}$ は線形方程式を求めるための補助行列である．また、$p_i \in R$, $\boldsymbol{D} \in \mathcal{S}$ と $\boldsymbol{K}' \in R^{n\times n}$ はそれぞれ定数スカラー、$n \times n$ の定数対称行列、そして $n \times n$ の定数行列であり、現在の点 (解)$(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ などの幾つかの要因によって決定する．これらの要因

は、探索方向によって異なる。例えば、HRVW/KSH/M 方向を用いた 実行可能パス追跡法 (feasible-path-following) においては以下のような値を選択する．

$$p_i = 0 \ (1 \leq i \leq m),\ \boldsymbol{D} = \boldsymbol{O},\ \boldsymbol{K}' = \beta \frac{\boldsymbol{X} \bullet \boldsymbol{Z}}{n} \boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z} \ \text{ ただし } \beta \in [0, 1)$$

HRVW/KSH/M 方向を用いた種々の 主双対内点法については以下の論文を参照されたい [1, 2, 19, 27, 26, 28, 30, 35, 44, 49, 57, etc.]. 先ほど述べた定数行列 $\{\boldsymbol{A}_i : 1 \leq i \leq m\}$ の線形独立の仮定より、$\boldsymbol{X} \succ \boldsymbol{O}$, $\boldsymbol{Z} \succ \boldsymbol{O}$ となるような全ての行列 $\boldsymbol{X}, \boldsymbol{Z}$, $p_i \in R$ $(1 \leq i \leq m)$, $\boldsymbol{D} \in \mathcal{S}$, そして $\boldsymbol{K}' \in R^{n \times n}$ に対して、連立方程式 (2.4), (2.5) と (2.6) は唯一の解 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ を持つことがいえる [26].

NT 方向は、HRVW/KSH/M 方向と等式 (2.4) と (2.5) を共有している．ただし等式 (2.6) は、

$$d\boldsymbol{X}\boldsymbol{W}^{-1} + \boldsymbol{W}d\boldsymbol{Z} = \boldsymbol{K}", \tag{2.7}$$

と置き換える．このとき $\boldsymbol{W} = \boldsymbol{X}^{1/2}(\boldsymbol{X}^{1/2}\boldsymbol{Z}\boldsymbol{X}^{1/2})^{-1/2}\boldsymbol{X}^{1/2} \in \mathcal{S}$ であり, $\boldsymbol{K}" \in R^{n \times n}$ は HRVW/KSH/M 方向と同様に、現在の点 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ などの幾つかの要因で決定される．NT 方向も連立方程式 (2.4), (2.5) と (2.7) の解 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ よって得られる．HRVW/KSH/M 方向と同様に $p_i \in R$, $\boldsymbol{D} \in \mathcal{S}$, $\boldsymbol{K}" \in R^{n \times n}$ であり、NT 方向においても解の一意性が示されている [49].

最後に AHO 方向 [1] について記す．AHO 方向も、HRVW/KSH/M 方向と等式 (2.4) と (2.5) を共有している．ただし等式 (2.6) は、

$$\frac{(d\boldsymbol{X}\boldsymbol{Z} + \boldsymbol{X}d\boldsymbol{Z}) + (d\boldsymbol{X}\boldsymbol{Z} + \boldsymbol{X}d\boldsymbol{Z})^T}{2} = \boldsymbol{K}"', \tag{2.8}$$

と置き換える．AHO 方向においても他の方向と同様に $p_i \in R$, $\boldsymbol{D} \in \mathcal{S}$, $\boldsymbol{K}"' \in \mathcal{S}$ である．これら 3 つの探索方向の SDPA への実装方法については 3 章において説明する．

第 3章

主双対内点法の実装方法

この章では、主双対内点法の実装方法について述べる．SDPA は基本的に 2.3 節で示した主双対内点法の枠組みに基づいている．しかし、大規模な問題を数値的に安定しながら高速に解くには様々な工夫が必要になる．SDPA を高速化させる方針は、基本的にはボトルネックとなる部分を探し出して、データ構造を変更したり、計算方法を工夫して高速化、省メモリ化を行う．そして、高速化に成功すれば今度は他の部分が新しいボトルネック部分になるので同様に改善を行う．最終的には、これ以上の改善が難しいと思われるか、各部分の実行時間が平準化されるまで改善を行う．

2.3 節で示した主双対内点法の枠組みの中では、手順２のニュートン方程式を構成する部分、つまり探索方向を計算する部分は後述するように最も実行時間を消費するので、これを高速化する方法を 3.2 節 に示す．また、探索方向を高速化すると手順４などで固有値を求める部分の実行時間が相対的に大きくなって新しいボトルネックになるのでこれも高速化を行う (3.3 節)．また、探索方向を高速化するには入力データの疎行列構造を利用しなければならないのでこの目的のためのデータ構造を 3.1 節で示す．

よって、この章は以下の構成を取る．

1. SDPA で採用する入力行列のデータ構造とその操作 (3.1 節)
2. ３つの探索方向 (HRVW/KSH/M, NT, AHO) の効率的な計算方法 (3.2 節)
3. 行列の最小固有値の効率的な近似法 (3.3 節)

SDPA は オブジェクト指向言語 である C++ で作成されている．よって C++ 言語の大きな特徴である、クラスや多重定義などの利点を活かしてアルゴリズムの記述が行われている．

3.1 SDPA で採用する入力行列のデータ構造とその操作

3.1.1 ブロック対角行列

ここで行列を区別する概念あるいは指標として、密 (dense) および 疎 (sparse) を導入する．一般的には、ある行列が密であるとは、その行列の非零要素の占める割合が多い場合を

示す．また反対に行列が疎であるとは、その行列の非零要素の数が非常に少ない場合を示している．しかし、非零要素の数が多くも少なくもない場合には、密とも疎とも判断しにくいので、その場合にはやや密 (mildly dense) と呼ぶことにする．SDP の前身とも言える LP のソフトウェアにおいても入力データの疎構造を活用して高速化あるいは主記憶上のメモリ量を削減するといった方法が用いられてきた．もはや数理計画等のソフトウェアにおいてはこういった工夫は当然備えるべき機能であるとされる．もちろん SDPA においても、疎行列などの入力行列の特殊構造を巧みに活用する工夫を組み入れている．次にこれらの工夫について説明する．

SDPA ではブロック対角なデータ行列の入力およびそれらの内部演算を組み入れている．次にブロック対角行列の一般形を示す．ブロック数とブロック構造ベクトルを用いて，定数行列 $\boldsymbol{A}_0,\ \boldsymbol{A}_1,\ \ldots,\ \boldsymbol{A}_m$ に共通なブロックデータ構造を表わす．

$$\left.\begin{array}{rcl} \boldsymbol{A} & = & \begin{pmatrix} \boldsymbol{B}_0 & \boldsymbol{O} & \boldsymbol{O} & \ldots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{B}_1 & \boldsymbol{O} & \ldots & \boldsymbol{O} \\ \cdot & \cdot & \cdot & \ldots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \ldots & \boldsymbol{B}_\ell \end{pmatrix}, \\ \boldsymbol{B}_i & : & p_i \times p_i \text{ 対称行列 } (i = 0, 1, \ldots, \ell - 1) \end{array}\right\} \tag{3.1}$$

とすると，この対称行列のブロック数 nBLOCK およびブロック構造ベクトル bLOCKsTRUCT は以下のように定義することができる．

$$\begin{aligned} \text{nBLOCK} &= \ell, \\ \text{bLOCKsTRUCT} &= (\beta_0,\ \beta_1,\ \ldots,\ \beta_{\ell-1}), \\ \beta_i &= \begin{cases} p_i & \boldsymbol{B}_i \text{ が通常の対称行列のとき} \\ -p_i & \boldsymbol{B}_i \text{ が対角行列のとき} \end{cases} \end{aligned}$$

例えば,

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 & 0 & 0 \\ 3 & 5 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix} \tag{3.2}$$

では

$$\begin{aligned} \text{nBLOCK} &= 3, \\ \text{bLOCKsTRUCT} &= (3,\ 2,\ -2) \end{aligned}$$

となる．また、

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 5 & 6 \\ 3 & 5 & 6 & 7 \\ 4 & 6 & 7 & 8 \end{pmatrix} \tag{3.3}$$

では

$$\begin{aligned} \mathrm{nBLOCK} &= 1, \\ \mathrm{bLOCKsTRUCT} &= (4) \end{aligned}$$

となる．このブロック対角行列構造の利点および特徴を以下に挙げる．

1. 行列を保持するための主記憶 (メモリ) 量の節約になる．
2. 乗算や逆行列等の行列対する演算は各ブロックごとに独立なので、各ブロックごとに演算を行えばよい．
3. ブロック対角構造を持たない行列も容易に扱うことができる．
4. 状況に応じて、各行列 $\boldsymbol{B}_i$ ごとにデータ構造を選択できる．

なお、行列 $\boldsymbol{B}_i$ は、密行列 (通常の 2 次元配列) と疎行列の 2 種類のデータ構造に対応することができるが、SDPA の最新バージョンでは入力データ ($\boldsymbol{A}_i$) を扱う場合には疎行列のみに限定している．よって次節以降でブロック対角行列の C++ での実装方法の説明を行うが、疎行列の場合のみに限定する．しかし一般的には入力データ (定数行列 $\boldsymbol{A}_i$) の各 $\boldsymbol{B}_i$ が疎行列であったとしても、変数行列 ($\boldsymbol{X}, \boldsymbol{Z}$) 等の $\boldsymbol{B}_i$ は密行列になる．よってこれらの行列は最初から $\boldsymbol{B}_i$ は密行列で保持することにする．

3.1.2 ブロック対角疎行列クラス：SparseBlockMatrix

この節においては、3.1.1 節で説明したブロック対角行列を C++ のクラスを用いて記述する方法を示す．次に、ブロック対角疎行列クラスの概要を示す．なお説明を簡潔にするため重要度が低いと判断される項目を幾つか削除している．

```
class SparseBlockMatrix
{
public:
    int     nBlock;
    int*    blockStruct;
    SparseMatrix    *block;
    SparseBlockMatrix ();                    // コンストラクタ
    SparseBlockMatrix (int n, int *bVect);  // コンストラクタ
    ~SparseBlockMatrix ();                   // デストラクタ
    boolean SparseBlockMatrixDelete ();     // SparseBlockMatrix クラスの
動的にメモリを確保した部分を解放する．
};
```

図 3.1: ブロック対角疎行列クラス：SparseBlockMatrix

整数変数 (int) nBlock は (3.1) でのブロック数 nBLOCK, また整数へのポインタである blockStruct は、ブロック構造ベクトルである bLOCKsTRUCT にそれぞれ対応している．block は 次節で説明する SparseBlock クラスのオブジェクトへのポインタである．簡単に言えば SparseBlockMatrix クラスで定義するのは、(3.1) の $\boldsymbol{A}$ 全体であり、各 $\boldsymbol{B}_i$ に相当するのが SparseMatrix クラスである．

3.1.3 疎行列クラス：SparseMatrix

```
class SparseMatrix
{
public:
    int    mRow;
    int    nCol;
    unsigned        short   *Row_index;
    unsigned        short   *Col_index;
    double                     *element;
    int                          NonzeroNumber;
    SparseMatrix ();                          // コンストラクタ
    SparseMatrix (int m, int n, int max);   // コンストラクタ
    ~SparseMatrix ();                          // デストラクタ
};
```

図 3.2: 疎行列クラス：SparseMatrix

(3.1) の各 $\boldsymbol{B}_i$ に相当するのがこの SparseMatrix クラスである．mRow と nCol には、$\boldsymbol{B}_i$ の行数と列数がそれぞれ入るが、SDPA で扱う行列は全て対称行列なので mRow = nCol となる．また NonzeroNumber には $\boldsymbol{B}_i$ の上三角部分に含まれる非零要素の数を入力する．

図 3.3 は SparseBlockMatrix と SparseMatrix クラスの構成例を示している．SparseMatrix クラスでは、行列 $\boldsymbol{B}_i$ の大きさ、つまり n の大きさに unsigned short (通常は 2 byte) を仮定しているので、$n \leq 2^{16} - 1 = 65535$ でなければならない．しかし現在の計算機の能力で扱うことの出来る n の大きさはたかだか数千なので、実用上はまったく問題はない．SDPA において n の大きさを表すのに unsigned short を使用するには以下のような理由がある．例えば $\boldsymbol{B}_i$ を二次元配列で保持するとこの場合 double (通常は 8 byte) であるので、二次元配列の場合には $8n^2$ byte である。また図 3.3 のデータ構造のときこの行列が n^2 個の要素を持つと仮定しても unsigned short は 2 byte なので $(2+2+8)\frac{n(n+1)}{2}$ byte になる。よって $8n^2 - 6n(n+1) = 2n^2 - 6n = 2n(n-3) > 0$ より $n \geq 4$ ならば疎データ構造の方がメモリの消費量が少ない。しかも、この疎データ構造は要素へのアクセスのコストが高くないという利点も持っている。

$$A_1 \;=\; \left(\begin{array}{rrrrr|rrr} 11.0 & 0 & -13.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 25.9 & 0 & 0 & 0 \\ -13.1 & 0 & 33.2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 25.9 & 0 & 0 & 6.0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 3.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2.5 \end{array}\right)$$

ここで SparseBlockMatrix ∗Amat として定数行列 $\boldsymbol{A}_1$ を構成する SparseBlockMatrix クラスへのオブジェクトを Amat[1] とすると

$$\begin{gathered} \text{Amat[1].nBlock} = 2; \\ \text{Amat[1].blockStruct[0]} = 5; \\ \text{Amat[1].blockStruct[1]} = -3; \end{gathered}$$

よって、$\boldsymbol{B}_0$ は大きさ 5×5 の対称行列であり、$\boldsymbol{B}_1$ は大きさ 3 の対角行列になる.

$$\Downarrow$$

Amat[1].block[0] ($\boldsymbol{B}_0$) の構成例 : 非零要素は上三角部分だけを保持する. この場合は Amat[1].block[0].NonzeroNumber = 5 になる.

Row_index	→	0	0	$1^{\dagger}$	2	4
Col_index	→	0	2	$4^{\dagger}$	2	4
element	→	11.0	−13.1	$25.9^{\dagger}$	33.2	6.0

一例を示すと † の場合には、
Amat[1].block[0].Row_index[2] = 1;
Amat[1].block[0].Col_index[2] = 4 ;
Amat[1].block[0].element[2] = 25.9; となる.

$$\Downarrow$$

Amat[1].block[1] ($\boldsymbol{B}_1$) の構成例 : 対角行列では Col_index のみを使用する. また Amat[1].block[1].NonzeroNumber = 3 である.

Col_index	→	0	1	2
element	→	3.0	−1.0	−2.5

図 3.3: SparseBlockMatrix と SparseMatrix クラスの構成例

3.2 疎行列を利用した探索方向の効率的な計算方法

この節では、SDPA が現在組み込んでいる 3 つの探索方向 (HRVW/KSH/M, NT, AHO) を効率良く計算する方法を紹介する．特に HRVW/KSH/M と NT 方向に関しては定数行列 $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ の全てまたは幾つかが疎行列であるときに効率的にそれらの方向を計算することができる．しかし $\boldsymbol{A}_i$ の全てが密行列であっても通常の方法と比較して効率が悪くなるわけではない．この計算方法の特徴を簡単にまとめると以下のようになる．

1. SDPA は $\boldsymbol{A}_i$ が密 (dense)、やや密 (mildly dense)、疎 (sparse) の 3 つの場合に備えて、それぞれ専用の計算方法を用意している．
2. SDPA は $\boldsymbol{A}_i$ の非零要素の数から自動的に計算方法を判断する．
3. 例えば $\boldsymbol{B}_0$ は密行列、$\boldsymbol{B}_1$ は疎行列といったような場合には、各ブロックごとに異なる計算方法を自動的に選択することができる．

3.2.1 疎行列を利用した HRVW/KSH/M と NT 方向の計算法

HRVW/KSH/M 方向の計算は (2.4 節の (2.4), (2.5), (2.6))、次の連立方程式を解くことに還元することができる．

$$\left.\begin{array}{l} \boldsymbol{B}' d\boldsymbol{y} = \boldsymbol{b}', \\ d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j, \\ \widehat{d\boldsymbol{X}} = (\boldsymbol{K}' - \boldsymbol{X} d\boldsymbol{Z})\, \boldsymbol{Z}^{-1},\ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^{T})/2, \end{array}\right\} \tag{3.4}$$

このとき

$$\boldsymbol{B}'_{ij} = \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_j \ (1 \leq i \leq m,\ 1 \leq j \leq m), \tag{3.6'}$$

$$\boldsymbol{b}'_i = (\boldsymbol{X}\boldsymbol{D} - \boldsymbol{K}')\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i + p_i \ (1 \leq i \leq m).$$

である．

ここで $n \times n$ 行列の $\boldsymbol{X}$, $n \times n$ 行列の $\boldsymbol{Z}^{-1}$ さらに $m \times m$ 行列の $\boldsymbol{B}'$ は全て対称行列であり, 一般的には $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ の全てが疎行列であったとしても全て密行列になる．ここで (3.4) を解いて探索方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ を求めるにはコレスキー分解 (Cholesky factorization) や LDL 分解 (LDL factorization) などを使用して $O(m^3 + n^3)$ の計算量が必要になる．一方, 係数行列 $\boldsymbol{B}'$ および右辺項 $\boldsymbol{b}'$ を計算するにはそれぞれ $O(mn^3 + m^2n^2)$, $O(n^3 + mn^2)$ の計算量が必要になる．それゆえ係数行列 $\boldsymbol{B}'$ を計算する部分は、$\boldsymbol{b}'$ の計算や 線形方程式の計算 $\boldsymbol{B}' d\boldsymbol{y} = \boldsymbol{b}'$ よりも HRVW/KSH/M 方向の計算時間全体の中で大きな割合を占める．それゆえ、$\boldsymbol{A}_i$ $(1 \leq i \leq m)$ の疎行列性を利用して $\boldsymbol{B}$ の計算量を下げることが全体の計算量の軽減に大きく貢献することになる．

また NT 方向についても (3.4) と同様に (2.4), (2.5), (2.7) は次の連立方程式を解くことに還元することができる.

$$\left.\begin{aligned} &\boldsymbol{B}''d\boldsymbol{y} = \boldsymbol{b}'', \\ &d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j, \\ &d\boldsymbol{X} = (\boldsymbol{K}'' - \boldsymbol{W}\,d\boldsymbol{Z})\,\boldsymbol{W}, \end{aligned}\right\} \tag{3.5}$$

このとき

$$\boldsymbol{B}''_{ij} = \boldsymbol{W}\boldsymbol{A}_i\boldsymbol{W} \bullet \boldsymbol{A}_j \ (1 \le i \le m,\ 1 \le j \le m), \tag{3.6"}$$

$$\boldsymbol{b}''_i = (\boldsymbol{W}\boldsymbol{D} - \boldsymbol{K}")\boldsymbol{W} \bullet \boldsymbol{A}_i + p_i \ (1 \le i \le m).$$

である.

NT 方向の場合においても HRVW/KSH/M 方向のときと同様に $\boldsymbol{B}''$ を計算する部分が最も計算量 ($O(mn^3 + m^2n^2)$) を必要とする.

そこで、この節の残りの部分では大きさ $m \times m$ の対称行列である $\boldsymbol{B}'$ と $\boldsymbol{B}''$ の効率的な計算方法を解説する. 2 つの方向に対する計算を同時に考慮するために、

$$\boldsymbol{B}_{ij} = \boldsymbol{T}\boldsymbol{A}_i\boldsymbol{U} \bullet \boldsymbol{A}_j \tag{3.6}$$

$(1 \le i \le m,\ 1 \le j \le m)$ を導入する. このとき $\boldsymbol{T} \in \mathcal{S}, \boldsymbol{U} \in \mathcal{S}$ である. ここで $\boldsymbol{T} = \boldsymbol{X}$, $\boldsymbol{U} = \boldsymbol{Z}^{-1}$ とすれば $\boldsymbol{B} = \boldsymbol{B}'$ となる. また $\boldsymbol{T} = \boldsymbol{U} = \boldsymbol{W}$ とすれば $\boldsymbol{B} = \boldsymbol{B}''$ となる. $\boldsymbol{B}$ は対称行列なので、行列 $\boldsymbol{B}$ の上三角部分、つまり $\boldsymbol{B}_{ij}$ $(1 \le i \le j \le m)$ のみ考慮すればよい.

行列 $\boldsymbol{B}$ の計算時間には浮動小数点の乗算や加算、また行列 $\boldsymbol{A}_i$ $(1 \le i \le m)$ の各要素を取り出すための時間など多くの要因が含まれている. この場合加算の回数は乗算の回数とほぼ同じオーダーであるが (ただし一般には浮動小数点の乗算は加算よりも時間を必要とする)、行列 $\boldsymbol{A}_i$ から要素を取り出す時間は、データ構造の選択に大きく影響される. そこで、最初に乗算の回数に実行時間の解析の焦点をあてて、その後 $\boldsymbol{A}_i$ $(1 \le i \le m)$ の各要素を取り出す時間 (いわゆるオーバーヘッド) も考慮に入れて行くことにする.

ここで、f_i を $\boldsymbol{A}_i$ の非零要素の数とする. また $\boldsymbol{\Sigma}$ は索引 (index) $1, 2, \ldots, m$ の順列の集合とする. (つまり, $\{1, 2, \ldots, m\}$ からそれ自身への一対一写像である). 各 $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ は次のように $\boldsymbol{B}_{ij}$ $(1 \le i \le m,\ 1 \le j \le m)$ 要素を計算する順序を決定する.

$$\left.\begin{aligned} &\overrightarrow{\boldsymbol{B}_{\sigma(1)\sigma(1)},\ \boldsymbol{B}_{\sigma(1)\sigma(2)},\ \ldots\ \boldsymbol{B}_{\sigma(1)\sigma(m)},} \\ &\qquad\qquad \overrightarrow{\boldsymbol{B}_{\sigma(2)\sigma(2)},\ \ldots\ \boldsymbol{B}_{\sigma(2)\sigma(m)},} \\ &\qquad\qquad\qquad \overrightarrow{\ldots\ \ldots} \\ &\qquad\qquad\qquad\qquad \boldsymbol{B}_{\sigma(m)\sigma(m)}. \end{aligned}\right\} \tag{3.7}$$

既に述べたように $\boldsymbol{B}$ は対称行列なので $\boldsymbol{B}_{\sigma(j)\sigma(i)} = \boldsymbol{B}_{\sigma(i)\sigma(j)}$ $(1 \le i < j \le m)$ である.

次に $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ と $i \in \{1, 2, \ldots, m\}$ を固定し、$\boldsymbol{B}_{\sigma(i)\sigma(j)}$ $(i \le j \le m)$ を計算する 3 つの式を提案する.

$\mathcal{F}$-1: 始めに $\boldsymbol{F}_i = \boldsymbol{A}_{\sigma(i)}\boldsymbol{U}$ ($nf_{\sigma(i)}$ 回の乗算を必要とする) を計算して、次に $\boldsymbol{G}_i = \boldsymbol{T}\boldsymbol{F}_i$ (n^3 回の乗算が必要) の計算を行う．その後各 $j = i, i+1, \ldots, m$ に対して $\boldsymbol{B}_{\sigma(i)\sigma(j)} = \boldsymbol{G}_i \bullet \boldsymbol{A}_{\sigma(j)}$ ($f_{\sigma(j)}$ 回の乗算が必要) の計算を行う．

よって、全ての $\boldsymbol{B}_{\sigma(i)\sigma(j)}$ $(i \leq j \leq m)$ を計算するための乗算の回数の合計は次の式から得ることができる．

$$nf_{\sigma(i)} + n^3 + \sum_{i \leq j \leq m} f_{\sigma(j)}. \tag{3.8}$$

$\mathcal{F}$-2: 始めに $\boldsymbol{F}_i = \boldsymbol{A}_{\sigma(i)}\boldsymbol{U}$ ($nf_{\sigma(i)}$ 回の乗算が必要) の計算を行う。次に各 $j = i, i+1, \ldots, m$ に対して、

$$\boldsymbol{B}_{\sigma(i)\sigma(j)} = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_{\sigma(j)}]_{\alpha\beta}\left(\sum_{\gamma=1}^{n} T_{\alpha\gamma}[\boldsymbol{F}_i]_{\gamma\beta}\right),$$

の計算を行う ($(n+1)f_{\sigma(j)}$ 回の乗算が必要)．よって、全ての $\boldsymbol{B}_{\sigma(i)\sigma(j)}$ $(i \leq j \leq m)$ を計算するための乗算の回数の合計は次の式から得ることができる．

$$nf_{\sigma(i)} + (n+1)\sum_{i \leq j \leq m} f_{\sigma(j)}. \tag{3.9}$$

$\mathcal{F}$-3: 各 $j = i, i+1, \ldots, m$ に対して、

$$\boldsymbol{B}_{\sigma(i)\sigma(j)} = \sum_{\gamma=1}^{n}\sum_{\epsilon=1}^{n}\left(\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_{\sigma(i)}]_{\alpha\beta}T_{\alpha\gamma}U_{\beta\epsilon}\right)[\boldsymbol{A}_{\sigma(j)}]_{\gamma\epsilon},$$

を計算する ($(2f_{\sigma(i)}+1)f_{\sigma(j)}$ 回の乗算が必要)．よって、全ての $\boldsymbol{B}_{\sigma(i)\sigma(j)}$ $(i \leq j \leq m)$ を計算するための乗算の回数の合計は次の式から得ることができる．

$$(2f_{\sigma(i)}+1)\sum_{i \leq j \leq m} f_{\sigma(j)} \tag{3.10}$$

ここで行列 $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ の要素を取り出すための時間 (オーバーヘッド) を考慮に入れることにする．最初に次の仮定を置く．

(i) 全ての $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ は 3.1.2 節で導入した SparseBlockMatrix クラスで定義して、図 3.3 のように疎行列用のデータ構造で保持するものとする．

(ii) 行列 $\boldsymbol{T}$ と $\boldsymbol{U}$ の $\boldsymbol{B}_i$ 部分は通常の二次元配列で保持する (多くの場合密であるから)．

(iii) 疎行列のデータ構造から要素を取り出す操作の方が、通常の二次元配列から要素を取り出すよりも実行時間がかかるものとする．

これらの仮定より、(3.8), (3.9), (3.10) 式に対して、以下のように修正を行う．ここで式 $\mathcal{F}$-k $(k = 1, 2, 3)$ を用いて、全ての $\boldsymbol{B}_{\sigma(i)\sigma(j)}$ $(i \leq j \leq m)$ を計算するための乗算の重み付き合計数 $d_{ki}(\boldsymbol{\sigma})$ を定義する．

$$d_{1i}(\boldsymbol{\sigma}) = \kappa nf_{\sigma(i)} + n^3 + \kappa\sum_{i \leq j \leq m} f_{\sigma(j)}, \tag{3.8'}$$

$$d_{2i}(\boldsymbol{\sigma}) = \kappa n f_{\sigma(i)} + \kappa(n+1) \sum_{i \le j \le m} f_{\sigma(j)}, \tag{3.9)'}$$

$$d_{3i}(\boldsymbol{\sigma}) = \kappa(2\kappa f_{\sigma(i)} + 1) \sum_{i \le j \le m} f_{\sigma(j)}. \tag{3.10)'}$$

ここで $\kappa \ge 1$ は定数であり、κ を疎行列データ構造を持つ $\boldsymbol{A}_i$ $(1 \le i \le m)$ の各要素を取り出すためのオーバーヘッドとみなすことにする．もし $\kappa = 1$ ならば (3.8)', (3.9)', (3.10)' は元の定義である (3.8), (3.9), (3.10) と一致する．SDPA においては、 ある $\boldsymbol{A}_i$ $(1 \le i \le m)$ が与えられたとき、式 $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3 の実際のパフォーマンスを推定するための指標として $d_{ki}(\boldsymbol{\sigma})$ を使用している．κ の値の適正値は採用するデータ構造などの要因で異なってくるが、幾つかの予備実験の結果から $\kappa = 1.5$ を採用して数値実験を行う．

ここで $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ として、

$$d_{*i}(\boldsymbol{\sigma}) = \min\{d_{1i}(\boldsymbol{\sigma}),\ d_{2i}(\boldsymbol{\sigma}),\ d_{3i}(\boldsymbol{\sigma})\}\ (1 \le i \le m), \tag{3.11}$$

$$d_{*}(\boldsymbol{\sigma}) = \sum_{1 \le i \le m} d_{*i}(\boldsymbol{\sigma}). \tag{3.12}$$

を定義する．そのとき各 $(i = 1, 2, \ldots, m)$ に対して、$d_{*i}(\boldsymbol{\sigma})$ は $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3 の何れかの式を用いて $\boldsymbol{B}_{\sigma(i)\sigma(j)}$ $(i \le j \le m)$ を計算するための乗算の回数の最小重み付き数を示している．そして、それらの合計 $d_*(\boldsymbol{\sigma})$ は行列 $\boldsymbol{B}$ の全ての要素を (3.7) の順序で計算したときの乗算の回数の最小重み付き数を示している．$d_*(\boldsymbol{\sigma})$ の数は $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ に依存する, つまり $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3 の式を $\boldsymbol{B}$ の計算のために適用する以前に、どのようにして、索引 (index : $1, 2, \ldots, m$) の順列を決定するかに依存している．そこで、$d_*(\boldsymbol{\sigma})$ を最小するような順列を選択するのが望ましい．以下に示す定理は、$f_1, f_2, \ldots, f_m$ の数を指標として索引 (index) を降順に並び替えたときに すべての $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ に対して $d_*(\boldsymbol{\sigma})$ が最小になることを示している．

定理 3.2.1 *(i)* $\boldsymbol{\sigma}^*$ が全ての $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ の中で、 $d_*(\boldsymbol{\sigma})$ を最小にすることと、以下の関係が成り立つことは必要十分である．

$$f_{\sigma^*(1)} \ge f_{\sigma^*(2)} \ge \ \cdots\ \ge f_{\sigma^*(m)}. \tag{3.13}$$

(ii) $\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ が *(3.13)* を満たすと仮定すると、そのとき次の条件を満たすような $q_1 \in \{0, 1, 2, \ldots, m\}$ と $q_2 \in \{q_1, q_1 + 1, \ldots, m\}$ が存在する．

$$\left.\begin{array}{ll} d_{1i}(\boldsymbol{\sigma}^*) \le d_{2i}(\boldsymbol{\sigma}^*), & d_{1i}(\boldsymbol{\sigma}^*) \le d_{3i}(\boldsymbol{\sigma}^*) \quad 0 < i \le q_1 \text{ のとき}, \\ d_{2i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*), & d_{2i}(\boldsymbol{\sigma}^*) \le d_{3i}(\boldsymbol{\sigma}^*) \quad q_1 < i \le q_2 \text{ のとき}, \\ d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*), & d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \quad q_2 < i \le m \text{ のとき}. \end{array}\right\} \tag{3.14}$$

証明: 始めに２つの補題を証明する．

補題 1. $\boldsymbol{\sigma}$ を順列 $\{1, 2, \ldots, m\}$ とする．ある $\ell \in \{1, 2, \ldots, m-1\}$ に対して、$f_{\sigma(\ell)} < f_{\sigma(\ell+1)}$ が成り立つと仮定する．また順列 $\boldsymbol{\tau}$ を次のように定義する．

$$\tau(i) = \begin{cases} \sigma(i) & 1 \le i < \ell \text{ または } \ell + 1 < i \le m \\ \sigma(\ell+1) & i = \ell \\ \sigma(\ell) & i = \ell + 1. \end{cases}$$

このとき $d_*(\boldsymbol{\sigma}) > d_*(\boldsymbol{\tau})$ である.

証明: 表記を簡単にするために、$\sigma(i) = i$ $(i = 1, 2, \ldots, m)$ を仮定する. この仮定から次のことが言える.

$$\begin{aligned} &\tau(i) = \begin{cases} i & 1 \le i < \ell \text{ または } \ell + 1 < i \le m, \\ \ell + 1 & i = \ell, \\ \ell & i = \ell + 1, \end{cases} \\ &f_\ell < f_{\ell+1}, \\ &d_{*i}(\boldsymbol{\sigma}) = d_{*i}(\boldsymbol{\tau}) \quad 1 \le i < \ell \text{ または } \ell + 1 < i \le m. \end{aligned} \tag{3.15}$$

よって、仮定 (3.15) もとでは次の不等式が満たされることが言える.

$$d_{*\ell}(\boldsymbol{\sigma}) + d_{*(\ell+1)}(\boldsymbol{\sigma}) > d_{*\ell}(\boldsymbol{\tau}) + d_{*(\ell+1)}(\boldsymbol{\tau}) \tag{3.16}$$

さらに、定義より次の関係が成り立つことも言える.

$$\begin{aligned} d_{q\ell}(\boldsymbol{\tau}) &\ge d_{*\ell}(\boldsymbol{\tau}) \text{ (どの } q \in \{1, 2, 3\} \text{ に対しても)}, \\ d_{r(\ell+1)}(\boldsymbol{\tau}) &\ge d_{*(\ell+1)}(\boldsymbol{\tau}) \text{ (どの } r \in \{1, 2, 3\} \text{ に対しても)}. \end{aligned}$$

それゆえに、もしある $q,\ r \in \{1, 2, 3\}$ に対して、次の不等式が成立すれば、

$$d_{*\ell}(\boldsymbol{\sigma}) + d_{*(\ell+1)}(\boldsymbol{\sigma}) \ge d_{q\ell}(\boldsymbol{\tau}) + d_{r(\ell+1)}(\boldsymbol{\tau}) \tag{3.17}$$

不等式 (3.16) が成り立つことも言える. ある $q,\ r \in \{1, 2, 3\}$ が与えられたときに、(3.17) 式が導きだせることを言うために、q と r を次のように取る.

(a) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{1\ell}(\boldsymbol{\sigma})$ かつ $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{1(\ell+1)}(\boldsymbol{\sigma})$ ならば $q = 1$ かつ $r = 1$ とする.

(b) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{1\ell}(\boldsymbol{\sigma})$ かつ $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{2(\ell+1)}(\boldsymbol{\sigma})$ ならば $q = 1$ かつ $r = 2$ とする.

(c-1) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{1\ell}(\boldsymbol{\sigma})$, $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{3(\ell+1)}(\boldsymbol{\sigma})$ かつ $2\kappa \sum_{\ell \le j \le m} f_j \ge n$ ならば $q = 1$ かつ $r = 3$ とする.

(c-2) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{1\ell}(\boldsymbol{\sigma})$, $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{3(\ell+1)}(\boldsymbol{\sigma})$ かつ $2\kappa \sum_{\ell \le j \le m} f_j < n$ ならば $q = 3$ かつ $r = 3$ とする.

(d) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{2\ell}(\boldsymbol{\sigma})$ かつ $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{1(\ell+1)}(\boldsymbol{\sigma})$ ならば $q = 1$ かつ $r = 2$ とする.

(e) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{2\ell}(\boldsymbol{\sigma})$ かつ $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{2(\ell+1)}(\boldsymbol{\sigma})$ ならば $q = 2$ かつ $r = 2$ とする.

(f-1) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{2\ell}(\boldsymbol{\sigma})$, $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{3(\ell+1)}(\boldsymbol{\sigma})$ かつ $2\kappa \sum_{\ell \le j \le m} f_j \ge n$ ならば $q = 2$ かつ $r = 3$ とする.

(f-2) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{2\ell}(\boldsymbol{\sigma})$, $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{3(\ell+1)}(\boldsymbol{\sigma})$ かつ $2\kappa \sum_{\ell \le j \le m} f_j < n$ ならば $q = 3$ かつ $r = 3$ とする.

(g) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{3\ell}(\boldsymbol{\sigma})$ かつ $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{1(\ell+1)}(\boldsymbol{\sigma})$ ならば $q = 1$ かつ $r = 3$ とする.

(h-1) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{3\ell}(\boldsymbol{\sigma})$, $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{2(\ell+1)}(\boldsymbol{\sigma})$ かつ $2\kappa f_{\ell+1} \geq n$ ならば $q = 2$ かつ $r = 3$ とする.

(h-2) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{3\ell}(\boldsymbol{\sigma})$, $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{2(\ell+1)}(\boldsymbol{\sigma})$ かつ $2\kappa f_{\ell+1} < n$ ならば $q = 3$ かつ $r = 3$ とする.

(i) もし $d_{*\ell}(\boldsymbol{\sigma}) = d_{3\ell}(\boldsymbol{\sigma})$ かつ $d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{3(\ell+1)}(\boldsymbol{\sigma})$ ならば $q = 3$ かつ $r = 3$ とする.

各場合において、不等式 (3.17) は容易に導きだせるので、ここでは、(c-1) と (c-2) の場合のみ扱う.

$$d_{*\ell}(\boldsymbol{\sigma}) = d_{1\ell}(\boldsymbol{\sigma}),\ d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{3(\ell+1)}(\boldsymbol{\sigma})\ \text{ かつ } 2\kappa \sum_{\ell \leq j \leq m} f_j \geq n.$$

を仮定すると、

$$\begin{aligned} &\Big(d_{1\ell}(\boldsymbol{\sigma}) + d_{3(\ell+1)}(\boldsymbol{\sigma})\Big) - \Big(d_{1\ell}(\boldsymbol{\tau}) + d_{3(\ell+1)}(\boldsymbol{\tau})\Big) \\ &= \ \kappa(f_{\ell+1} - f_\ell)\left(2\kappa \sum_{\ell \leq j \leq m} f_j - n\right) + \kappa(f_{\ell+1} - f_\ell)\ > \ 0. \end{aligned}$$

となる. よって (c-1) の場合において不等式 (3.17) が成立することを示すことができた.

$$d_{*\ell}(\boldsymbol{\sigma}) = d_{1\ell}(\boldsymbol{\sigma}),\ d_{*(\ell+1)}(\boldsymbol{\sigma}) = d_{3(\ell+1)}(\boldsymbol{\sigma})\ \text{ かつ } 2\kappa \sum_{\ell \leq j \leq m} f_j < n.$$

を仮定すると、

$$\begin{aligned} &\Big(d_{1\ell}(\boldsymbol{\sigma}) + d_{3(\ell+1)}(\boldsymbol{\sigma})\Big) - \Big(d_{3\ell}(\boldsymbol{\tau}) + d_{3(\ell+1)}(\boldsymbol{\tau})\Big) \\ &= \ \kappa f_\ell \left(n - 2\kappa \sum_{\ell \leq j \leq m} f_j\right) + n^3 + \kappa(f_{\ell+1} - f_\ell)\ > \ 0. \end{aligned}$$

である. よって (c-2) の場合においても不等式 (3.17) が成立することを示すことができた. 他の場合においても同様に導きだすことができる.

補題 2. $\boldsymbol{\sigma}$ を順列 $\{1, 2, \ldots, m\}$ また $i \in \{1, 2, \ldots, m-1\}$ とする. ある $i \in \{1, 2, \ldots, m-1\}$ に対して、$f_{\sigma(i)} \geq f_{\sigma(i+1)}$ を仮定する.

(a) もし $d_{2i}(\boldsymbol{\sigma}) \leq d_{1i}(\boldsymbol{\sigma})$ ならば $d_{2(i+1)}(\boldsymbol{\sigma}) < d_{1(i+1)}(\boldsymbol{\sigma})$.

(b) もし $d_{3i}(\boldsymbol{\sigma}) \leq d_{1i}(\boldsymbol{\sigma})$ ならば $d_{3(i+1)}(\boldsymbol{\sigma}) < d_{1(i+1)}(\boldsymbol{\sigma})$.

(c) もし $d_{3i}(\boldsymbol{\sigma}) \leq d_{2i}(\boldsymbol{\sigma})$ ならば $d_{3(i+1)}(\boldsymbol{\sigma}) < d_{2(i+1)}(\boldsymbol{\sigma})$.

証明: 表記を簡単にするため、$\sigma(j) = j$ $(j = 1, 2, \ldots, m)$ を仮定する, また d_{1j}, d_{2j} また d_{3j} をそれぞれ、$d_{1j}(\boldsymbol{\sigma})$, $d_{2j}(\boldsymbol{\sigma})$ そして $d_{3j}(\boldsymbol{\sigma})$ と表記する,

(a) (3.8)' と (3.9)' より、

$$\Big(d_{1(i+1)} - d_{2(i+1)}\Big) - (d_{1i} - d_{2i}) \ = \ n\kappa f_i > 0.$$

である．それゆえ $(d_{1i} - d_{2i}) \geq 0$ は $\left(d_{1(i+1)} - d_{2(i+1)}\right) > 0$ を意味する．

(b) By (3.8)' かつ (3.10)' より、

$$\left(d_{1(i+1)} - d_{3(i+1)})\right) \;=\; n^3 + \kappa f_{i+1} \left(n - 2\kappa \sum_{i+1 \leq j \leq m} f_j\right).$$

である．それゆえ, もし $\left(n - 2\kappa \sum_{i+1 \leq j \leq m} f_j\right) \geq 0$ ならば、$\left(d_{1(i+1)} - d_{3(i+1)}\right) > 0$ である．ここで、
$\left(n - 2\kappa \sum_{i+1 \leq i \leq m} f_j\right) < 0.$ を仮定すると、

$$\begin{aligned} &\left(d_{1(i+1)} - d_{3(i+1)}\right) - (d_{1i} - d_{3i}) \\ &\quad = \; 2(\kappa f_i)^2 + \kappa(f_i - f_{i+1}) \left(2\kappa \sum_{i+1 \leq j \leq m} f_j - n\right) \;>\; 0. \end{aligned}$$

それゆえ $(d_{1i} - d_{3i}) \geq 0$ は $\left(d_{1(i+1)} - d_{3(i+1)}\right) > 0$ を意味する．

(c) By (3.9)' かつ (3.10)' より, もし $n \geq 2\kappa f_{i+1}$ または $n \geq 2\kappa \sum_{i+1 \leq j \leq m} f_j$ ならば、

$$\begin{aligned} \left(d_{2(i+1)} - d_{3(i+1)}\right) \;&=\; n\kappa f_{i+1} + (n - 2\kappa f_{i+1})\, \kappa \sum_{i+1 \leq j \leq m} f_j \\ &=\; \left(n - 2\kappa \sum_{i+1 \leq j \leq m} f_j\right) \kappa f_{i+1} + n\kappa \sum_{i+1 \leq j \leq m} f_j \;>\; 0. \end{aligned}$$

ここで $n < 2\kappa f_{i+1}$ かつ $n < 2\kappa \sum_{i+1 \leq j \leq m} f_j.$ を仮定すると、

$$\begin{aligned} &\left(d_{2(i+1)} - d_{3(i+1)}\right) - (d_{2i} - d_{3i}) \\ &\quad =\; \kappa(f_i - f_{i+1}) \left(2\kappa \sum_{i+1 \leq j \leq m} f_j - n\right) + \kappa f_i \,(2\kappa f_i - n) \\ &\quad \geq\; f_i \,(2f_{i+1} - n) \;>\; 0. \end{aligned}$$

それゆえ $(d_{2i} - d_{3i}) \geq 0$ は $\left(d_{2(i+1)} - d_{3(i+1)}\right) > 0$ を意味する．

これより定理 3.2.1 を証明する．$\boldsymbol{\sigma}^*$ が $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ 中で、$d_*(\boldsymbol{\sigma})$ を最小にするとものとする．ここで、$\boldsymbol{\sigma}^*$ が (3.13) の関係を満たさないと仮定する．このとき、$f_{\sigma^*(\ell)} < f_{\sigma^*(\ell+1)}$ となるような $\ell \in \{1, 2, \ldots, m-1\}$ が存在する．それゆえ、補題 C.1 を適用することによって, $d_*(\boldsymbol{\tau}) < d_*(\boldsymbol{\sigma}^*)$ となるような $\boldsymbol{\tau} \in \boldsymbol{\Sigma}$ を見つけることができるので矛盾が生じる．よって、$\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ が (3.13) を満足することを示すことができた．

ここで、$\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ が (3.13) を満足すると仮定する．$\boldsymbol{\Sigma}$ の要素は有限なので, $d_* : \boldsymbol{\Sigma} \to R$ はある $\boldsymbol{\tau} \in \boldsymbol{\Sigma}$ に対して最小を達成する．また、この $\boldsymbol{\tau}$ は次の関係を満たさなければならな

い.

$$f_{\tau(1)} \geq f_{\tau(2)} \geq \ \cdots \ \geq f_{\tau(m)} \tag{3.13)'}$$

Since 関係式 (3.13) と (3.13)' が全ての $i \in \{1, 2, \ldots, m\}$ に対して $f_{\sigma(i)} = f_{\tau(i)}$ を意味することになるので、$d_*(\boldsymbol{\sigma}) = d_*(\boldsymbol{\tau})$ である. それゆえ、$\boldsymbol{\sigma}^* \in \boldsymbol{\Sigma}$ は、全ての $\boldsymbol{\sigma} \in \boldsymbol{\Sigma}$ の中で $d_*(\boldsymbol{\sigma})$ を最小にすることが言える.

また定理の (ii) は補題 C.2. から示すことができる. (証明終わり)

この定理に基づいて、ここで以下の計算方法を提案する.

Combined Formula $\mathcal{F}$-*(κ):

Step A: 行列 $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ の非零要素の数 f_i を数える.

Step B: $f_1, f_2, \ldots, f_m$ の数を指標として索引 (index : $1, 2, \ldots, m$) を降順に並び替える (3.13). ここで得られる $\boldsymbol{\sigma}^*$ は索引 (index : $\{1, 2, \ldots, m\}$) 集合の順列である. 各 $i = 1, 2, \ldots, m$ に対して、$d_{1i}(\boldsymbol{\sigma}^*)$ (3.8)' と $d_{2i}(\boldsymbol{\sigma}^*)$ (3.9)' と $d_{3i}(\boldsymbol{\sigma}^*)$ (3.10)' を計算する. そして (3.14) を満たすような $q_1 \in \{0, 1, 2, \ldots, m\}$ と $q_2 \in \{q_1, q_1{+}1, \ldots, m\}$ を求める.

Step C: 全ての $i \in \{1, 2, \ldots, m\}$ に対して、

- もし $0 < i \leq q_1$ ならば、 $\mathcal{F}$-1 を用いる.
- もし $q_1 < i \leq q_2$ ならば、$\mathcal{F}$-2 を用いる.
- 残りの部分、つまり $q_2 < i \leq m$ ならば、 $\mathcal{F}$-3 を用いる.

$\boldsymbol{B}_{\sigma^*(i)\sigma^*(j)}$ $(i \leq j \leq m)$ の計算を行う.

これらの手順を、HRVW/KSH/M 方向と NT 方向を用いた主双対内点法に容易に組み込むことができる. まず、$\boldsymbol{A}_i$ $(1 \leq i \leq m)$ を読み込んだときに、Step A を実行してその後すぐに Step B も実行する. つまり、Step A と Step B は最初に一回実行すればよい. 一方 Step C は各反復においてそれぞれ実行しなくてはならない.

3.2.2 HRVW/KSH/M 方向の実装方法

以下の 3 つの節では 2.4 節で計算式を簡単に説明した 3 つの探索方向の SDPA への実装法について述べる. HRVW/KSH/M 方向と NT 方向については、前節で提案した疎行列性を利用した効率の良い計算方法を組み入れる. 最初に 2.4 節で定義した HRVW/KSH/M 方向の式を示す.

$$\boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i \ (1 \leq i \leq m), \ d\boldsymbol{X} \in \mathcal{S}, \tag{3.18}$$

$$\sum_{i=1}^{m} \boldsymbol{A}_i dy_i + d\boldsymbol{Z} = \boldsymbol{D},\ d\boldsymbol{Z} \in \mathcal{S}, \tag{3.19}$$

$$\widehat{d\boldsymbol{X}}\,\boldsymbol{Z} + \boldsymbol{X}\,d\boldsymbol{Z} = \boldsymbol{K}',\ \widehat{d\boldsymbol{X}} \in R^{n\times n},\ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^{T})/2. \tag{3.20}$$

既に説明したように、$\widehat{d\boldsymbol{X}} \in R^{n\times n}$ は線形方程式を求めるための補助行列である．(3.20) 式は、本来

$$d\boldsymbol{X}\,\boldsymbol{Z} + \boldsymbol{X}\,d\boldsymbol{Z} = \boldsymbol{K}',\ d\boldsymbol{X} \in \mathcal{S}.$$

となるべきであるが、この場合には線形方程式が唯一の解を持たないので、$d\boldsymbol{X}$ を一般の実数行列に拡張して線形方程式を解く．最終的には、$\widehat{d\boldsymbol{X}}$ を対称化して $d\boldsymbol{X}$ としている．

また、$p_i \in R$ と $\boldsymbol{D} \in \mathcal{S}$ はそれぞれ主問題の残差 (定数スカラー)、$n \times n$ の双対問題の残差 (定数対称行列) である．よって、主問題と双対問題においてそれぞれ実行可能ならば、

$$p_i = 0\ (1 \le i \le m),\ \boldsymbol{D} = \boldsymbol{O}.$$

となる．また、(3.20) 式は、中心パスの定義である $\boldsymbol{X}\boldsymbol{Z} = \mu\boldsymbol{I}$ から導かれる．つまり探索方向を $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ 、また $\boldsymbol{K}' = \beta\dfrac{\boldsymbol{X}\bullet\boldsymbol{Z}}{n}\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z} : \beta \in [0,1)$ とすると、

$$\begin{aligned}
(\boldsymbol{X} + d\boldsymbol{X})(\boldsymbol{Z} + d\boldsymbol{Z}) &= \mu'\boldsymbol{I}\ :\ \mu' < \mu \\
d\boldsymbol{X}\,\boldsymbol{Z} + \boldsymbol{X}\,d\boldsymbol{Z} &= \mu'\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z}\ \ (\text{二次の } d\boldsymbol{X}d\boldsymbol{Z} \text{ 項を無視する.}) \\
d\boldsymbol{X}\,\boldsymbol{Z} + \boldsymbol{X}\,d\boldsymbol{Z} &= \beta\frac{\boldsymbol{X}\bullet\boldsymbol{Z}}{n}\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z}\ :\ \beta \in [0,1) \\
d\boldsymbol{X}\,\boldsymbol{Z} + \boldsymbol{X}\,d\boldsymbol{Z} &= \boldsymbol{K}' \\
\widehat{d\boldsymbol{X}}\,\boldsymbol{Z} + \boldsymbol{X}\,d\boldsymbol{Z} &= \boldsymbol{K}',\ \ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^{T})/2.
\end{aligned}$$

である．ここで β は、2.3 節で定義したように探索方向を決定するパラメータである．β が 1 に近いほど探索方向は中心パスの方を向き、0 に近いほど最適解の方向を向く．また $\dfrac{\boldsymbol{X}\bullet\boldsymbol{Z}}{n}$ は、$\boldsymbol{X}\boldsymbol{Z}$ の固有値の平均値を示している．

また次の連立方程式を解くことによって、HRVW/KSH/M 方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ が求められることは既に述べた．

$$\left.\begin{aligned}
&\boldsymbol{B}'d\boldsymbol{y} = \boldsymbol{b}', \\
&d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j, \\
&\widehat{d\boldsymbol{X}} = (\boldsymbol{K}' - \boldsymbol{X}\,d\boldsymbol{Z})\,\boldsymbol{Z}^{-1},\ d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^{T})/2,
\end{aligned}\right\} \tag{3.21}$$

このとき

$$\boldsymbol{B}'_{ij} = \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_j\ (1 \le i \le m,\ 1 \le j \le m), \tag{3.6)'}$$

$$\boldsymbol{b}'_i = (\boldsymbol{X}\boldsymbol{D} - \boldsymbol{K}')\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i + p_i\ (1 \le i \le m).$$

である．

この連立方程式は、(3.18), (3.19), (3.20) から次のようにして導き出すことができる．最初に、(3.20) を変形して次の式を得る．

$$\begin{aligned} \widehat{d\boldsymbol{X}} + \boldsymbol{X} d\boldsymbol{Z} \boldsymbol{Z}^{-1} &= \boldsymbol{K}' \boldsymbol{Z}^{-1} \\ \widehat{d\boldsymbol{X}} &= (\boldsymbol{K}' - \boldsymbol{X} d\boldsymbol{Z})\, \boldsymbol{Z}^{-1} \end{aligned}$$

また、同様に (3.19) を変形して次の式を得る．

$$d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j$$

続いて、(3.18) に $\widehat{d\boldsymbol{X}}$ と $d\boldsymbol{Z}$ を代入する．

$$\begin{aligned} \boldsymbol{A}_i \bullet d\boldsymbol{X} &= p_i \\ \boldsymbol{A}_i \bullet (\boldsymbol{K}' - \boldsymbol{X} d\boldsymbol{Z}) \boldsymbol{Z}^{-1} &= p_i \quad (\widehat{d\boldsymbol{X}} \text{ を代入}) \\ \boldsymbol{A}_i \bullet (\boldsymbol{K}' - \boldsymbol{X} (\boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j)) \boldsymbol{Z}^{-1} &= p_i \quad (d\boldsymbol{Z} \text{ を代入}) \\ \boldsymbol{A}_i \bullet (\boldsymbol{X} (\sum_{j=1}^{m} \boldsymbol{A}_j dy_j) \boldsymbol{Z}^{-1}) &= \boldsymbol{A}_i \bullet (\boldsymbol{X} \boldsymbol{D} \boldsymbol{Z}^{-1} - \boldsymbol{K}' \boldsymbol{Z}^{-1}) + p_i \\ \boldsymbol{A}_i \bullet (\boldsymbol{X} (\boldsymbol{A}_1 dy_1 + \cdots + \boldsymbol{A}_m dy_m) \boldsymbol{Z}^{-1}) &= (\boldsymbol{X} \boldsymbol{D} - \boldsymbol{K}') \boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i + p_i \\ \boldsymbol{A}_i \bullet ((\boldsymbol{X} \boldsymbol{A}_1 \boldsymbol{Z}^{-1}) dy_1 + \cdots + (\boldsymbol{X} \boldsymbol{A}_m \boldsymbol{Z}^{-1}) dy_m) &= (\boldsymbol{X} \boldsymbol{D} - \boldsymbol{K}') \boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i + p_i \end{aligned}$$

よって $\boldsymbol{B}'_{ij} = \boldsymbol{A}_i \bullet (\boldsymbol{X} \boldsymbol{A}_j \boldsymbol{Z}^{-1})$, $\boldsymbol{b}'_i = (\boldsymbol{X} \boldsymbol{D} - \boldsymbol{K}') \boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i + p_i$ とおくと $\boldsymbol{B}' d\boldsymbol{y} = \boldsymbol{b}'$ である．また、

$$\begin{aligned} \boldsymbol{B}'_{ij} &= \boldsymbol{A}_i \bullet (\boldsymbol{X} \boldsymbol{A}_j \boldsymbol{Z}^{-1}) \\ &= \mathrm{Tr}(\boldsymbol{A}_i \boldsymbol{X} \boldsymbol{A}_j \boldsymbol{Z}^{-1}) \quad : \quad \boldsymbol{U} \bullet \boldsymbol{V} = \mathrm{Tr}(\boldsymbol{U}^T \boldsymbol{V}) \text{ より} \\ &= \mathrm{Tr}(\boldsymbol{Z}^{-1} \boldsymbol{A}_i \boldsymbol{X} \boldsymbol{A}_j) \quad : \quad \mathrm{Tr}(\boldsymbol{U}\boldsymbol{V}) = \mathrm{Tr}(\boldsymbol{V}\boldsymbol{U}) \text{ より} \\ &= \mathrm{Tr}(\boldsymbol{A}_j \boldsymbol{X} \boldsymbol{A}_i \boldsymbol{Z}^{-1}) \quad : \quad \mathrm{Tr}(\boldsymbol{U}) = \mathrm{Tr}(\boldsymbol{U}^T),\ (\boldsymbol{U}\boldsymbol{V})^T = \boldsymbol{V}^T \boldsymbol{U}^T \text{ より} \\ &= \boldsymbol{A}_j \bullet \boldsymbol{X} \boldsymbol{A}_i \boldsymbol{Z}^{-1} = \boldsymbol{X} \boldsymbol{A}_i \boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_j \end{aligned}$$

と変形することができる．つまり行列 $\boldsymbol{B}'$ は対称行列である．

以下にこれらの計算式を用いて HRVW/KSH/M 方向の計算を SDPA に実装した手順を示す．

SDPA における HRVW/KSH/M 方向の計算方法.

1. $\boldsymbol{Z}^{-1}$ を計算する.
2. $\boldsymbol{K}' = \beta\dfrac{\boldsymbol{X} \bullet \boldsymbol{Z}}{n}\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z}$ を計算する.
3. $(\boldsymbol{X}\boldsymbol{D} - \boldsymbol{K}')\boldsymbol{Z}^{-1}$ を計算する.
4. $\boldsymbol{b}'_i = (\boldsymbol{X}\boldsymbol{D} - \boldsymbol{K}')\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_i + p_i \ (i = 1, \ldots, m)$ を計算する.
5. Combined Formula $\mathcal{F}$-$*$ を使用して $\boldsymbol{B}'$ を計算する.
   (a) Step A と Step B を実行して、(3.14) を満たすような $q_1 \in \{0, 1, 2, \ldots, m\}$ と $q_2 \in \{q_1, q_1 + 1, \ldots, m\}$ を求める (ただしこの実行は開始時に 1 回のみ).
   (b) $\boldsymbol{B}'_{ij} = \boldsymbol{X}\boldsymbol{A}_i\boldsymbol{Z}^{-1} \bullet \boldsymbol{A}_j \ \ (0 < i \leq q_1)$ を計算する.
   (c) $\boldsymbol{B}'_{ij} = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_j]_{\alpha\beta}\left(\sum_{\gamma=1}^{n}\boldsymbol{X}_{\alpha\gamma}[\boldsymbol{A}_i\boldsymbol{Z}^{-1}]_{\gamma\beta}\right) \ \ (q_1 < i \leq q_2)$ を計算する.
   (d) $\boldsymbol{B}'_{ij} = \sum_{\gamma=1}^{n}\sum_{\epsilon=1}^{n}\left(\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_i]_{\alpha\beta}\boldsymbol{X}_{\alpha\gamma}\boldsymbol{Z}^{-1}_{\beta\epsilon}\right)[\boldsymbol{A}_j]_{\gamma\epsilon} \ \ (q_2 < i \leq m)$ を計算する.
6. LDL 分解等を利用して $\boldsymbol{B}'d\boldsymbol{y} = \boldsymbol{b}'$ を解く.
7. $d\boldsymbol{y}$ を代入して $d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m}\boldsymbol{A}_j dy_j$ を計算する.
8. $\widehat{d\boldsymbol{X}} = (\boldsymbol{K}' - \boldsymbol{X}d\boldsymbol{Z})\,\boldsymbol{Z}^{-1}, d\boldsymbol{X} = (\widehat{d\boldsymbol{X}} + \widehat{d\boldsymbol{X}}^{T})/2$ を計算する.
9. HRVW/KSH/M 方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ を得る.

3.2.3 NT 方向の実装方法

この節では、NT 方向の計算を SDPA に実装する方法を説明する.

$$\boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i \ (1 \leq i \leq m), \ d\boldsymbol{X} \in \mathcal{S}, \tag{3.22}$$
$$\sum_{i=1}^{m}\boldsymbol{A}_i dy_i + d\boldsymbol{Z} = \boldsymbol{D}, \ d\boldsymbol{Z} \in \mathcal{S}, \tag{3.23}$$
$$d\boldsymbol{X}\boldsymbol{W}^{-1} + \boldsymbol{W}d\boldsymbol{Z} = \boldsymbol{K}", \boldsymbol{W} = \boldsymbol{X}^{1/2}(\boldsymbol{X}^{1/2}\boldsymbol{Z}\boldsymbol{X}^{1/2})^{-1/2}\boldsymbol{X}^{1/2} \in \mathcal{S}. \tag{3.24}$$

また、2.4 節で述べたように、NT 方向を求めるためは次の連立方程式を解かなければならない.

$$\left.\begin{aligned} &\boldsymbol{B}''d\boldsymbol{y} = \boldsymbol{b}'', \\ &d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m}\boldsymbol{A}_j dy_j, \\ &d\boldsymbol{X} = (\boldsymbol{K}'' - \boldsymbol{W}\,d\boldsymbol{Z})\,\boldsymbol{W}, \end{aligned}\right\} \tag{3.25}$$

このとき

$$\boldsymbol{B}''_{ij} = \boldsymbol{W}\boldsymbol{A}_i\boldsymbol{W} \bullet \boldsymbol{A}_j \ (1 \leq i \leq m,\ 1 \leq j \leq m), \qquad (3.6)''$$

$$\boldsymbol{b}''_i = (\boldsymbol{W}\boldsymbol{D} - \boldsymbol{K}'')\boldsymbol{W} \bullet \boldsymbol{A}_i + p_i \ (1 \leq i \leq m).$$

である．この連立方程式への変形も、$\boldsymbol{X} = \boldsymbol{W}, \boldsymbol{Z}^{-1} = \boldsymbol{W}$ と考えれば HRVW/KSH/M 方向のときと全く同じようにして行うことができる．

以下にこれらの計算式を用いて NT 方向の計算を SDPA に実装した手順を示す．

SDPA における NT 方向の計算方法.

1. $\boldsymbol{W} = \boldsymbol{X}^{1/2}(\boldsymbol{X}^{1/2}\boldsymbol{Z}\boldsymbol{X}^{1/2})^{-1/2}\boldsymbol{X}^{1/2}$ を計算する.
   (a) $\boldsymbol{X} = \boldsymbol{L}_X\boldsymbol{L}_X^T$ のコレスキー分解を行う.
   (b) $\boldsymbol{Z} = \boldsymbol{L}_Z\boldsymbol{L}_Z^T$ のコレスキー分解を行う.
   (c) $\boldsymbol{L}_Z^T\boldsymbol{L}_X$ を計算をする.
   (d) $\boldsymbol{L}_Z^T\boldsymbol{L}_X = \boldsymbol{U}^T\boldsymbol{D}'\boldsymbol{V}$ の特異値分解を行う.
   (e) $\boldsymbol{D}'^{-\frac{1}{2}}$ を計算する.
   (f) $\boldsymbol{G} = \boldsymbol{L}_X\boldsymbol{V}^T\boldsymbol{D}'^{-\frac{1}{2}}$ を計算する.
   (g) $\boldsymbol{W} = \boldsymbol{G}\boldsymbol{G}^T$ を計算する.
2. $\boldsymbol{K}'' = \beta\dfrac{\boldsymbol{X}\bullet\boldsymbol{Z}}{n}\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z}$ を計算する.
3. $(\boldsymbol{W}\boldsymbol{D} - \boldsymbol{K}'')\boldsymbol{W}$ を計算する.
4. $\boldsymbol{b}_i'' = (\boldsymbol{W}\boldsymbol{D} - \boldsymbol{K}'')\boldsymbol{W}\bullet\boldsymbol{A}_i + p_i$ $(i = 1,\ldots,m)$ を計算する.
5. Combined Formula $\mathcal{F}$-$*$ を使用して $\boldsymbol{B}''$ を計算する.
   (a) Step A と Step B を実行して、(3.14) を満たすような $q_1 \in \{0,1,2,\ldots,m\}$ と $q_2 \in \{q_1, q_1+1,\ldots,m\}$ を求める (ただしこの実行は開始時に 1 回のみ).
   (b) $\boldsymbol{B}''_{ij} = \boldsymbol{W}\boldsymbol{A}_i\boldsymbol{W}\bullet\boldsymbol{A}_j$ $(0 < i \le q_1)$ を計算する.
   (c) $\boldsymbol{B}''_{ij} = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_j]_{\alpha\beta}\left(\sum_{\gamma=1}^{n}\boldsymbol{W}_{\alpha\gamma}[\boldsymbol{A}_i\boldsymbol{W}]_{\gamma\beta}\right)$ $(q_1 < i \le q_2)$ を計算する.
   (d) $\boldsymbol{B}''_{ij} = \sum_{\gamma=1}^{n}\sum_{\epsilon=1}^{n}\left(\sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}[\boldsymbol{A}_i]_{\alpha\beta}\boldsymbol{W}_{\alpha\gamma}\boldsymbol{W}_{\beta\epsilon}\right)[\boldsymbol{A}_j]_{\gamma\epsilon}$ $(q_2 < i \le m)$ を計算する.
6. LDL 分解等を利用して $\boldsymbol{B}''d\boldsymbol{y} = \boldsymbol{b}''$ を解く.
7. $d\boldsymbol{y}$ を代入して $d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m}\boldsymbol{A}_j dy_j$ を計算する.
8. $d\boldsymbol{X} = (\boldsymbol{K}'' - \boldsymbol{W}\,d\boldsymbol{Z})\,\boldsymbol{W}$ を計算する.
9. NT 方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ を得る.

3.2.4 AHO 方向の実装方法

この節では、AHO 方向の計算を SDPA に実装する方法を説明する.

$$\boldsymbol{A}_i \bullet d\boldsymbol{X} = p_i \ (1 \le i \le m),\ d\boldsymbol{X} \in \mathcal{S}, \tag{3.26}$$

$$\sum_{i=1}^{m} \boldsymbol{A}_i dy_i + d\boldsymbol{Z} = \boldsymbol{D},\ d\boldsymbol{Z} \in \mathcal{S}, \tag{3.27}$$

$$\frac{(d\boldsymbol{X}\boldsymbol{Z} + \boldsymbol{X}d\boldsymbol{Z}) + (d\boldsymbol{X}\boldsymbol{Z} + \boldsymbol{X}d\boldsymbol{Z})^T}{2} = \boldsymbol{K}''', \tag{3.28}$$

次に、AHO 方向を SDPA に実装する方法を示す.

SDPA における AHO 方向の計算方法.

1. $\boldsymbol{R} = \beta\dfrac{\boldsymbol{X} \bullet \boldsymbol{Z}}{n}\boldsymbol{I} - \boldsymbol{X}\boldsymbol{Z}$ を計算する.
2. $\boldsymbol{K}''' = \dfrac{\boldsymbol{R} + \boldsymbol{R}^T}{2}$ を計算する.
3. $\dfrac{\boldsymbol{X}\boldsymbol{D} + \boldsymbol{D}\boldsymbol{X}}{2} - \boldsymbol{K}'''$ を計算する.
4. $\dfrac{\boldsymbol{V}\boldsymbol{Z} + \boldsymbol{Z}\boldsymbol{V}}{2} = \boldsymbol{A}_i$ を $\boldsymbol{V}$について解く (Lyapunov 方程式).
5. $\boldsymbol{b}_i''' = \boldsymbol{V} \bullet \left(\dfrac{\boldsymbol{X}\boldsymbol{D} + \boldsymbol{D}\boldsymbol{X}}{2} - \boldsymbol{K}'''\right) + p_i\ (i = 1, \ldots, m)$ を計算する.
6. $\boldsymbol{B}_{ij}''' = \boldsymbol{A}_j \bullet (\boldsymbol{V}\boldsymbol{X})$ を計算する.
7. LU 分解を用いて $\boldsymbol{B}'''d\boldsymbol{y} = \boldsymbol{b}'''$ を解く.
8. $d\boldsymbol{y}$ を代入して $d\boldsymbol{Z} = \boldsymbol{D} - \sum_{j=1}^{m} \boldsymbol{A}_j dy_j$ を計算する.
9. $\dfrac{d\boldsymbol{X}\boldsymbol{Z} + \boldsymbol{Z}d\boldsymbol{X}}{2} = \boldsymbol{K}''' - \dfrac{d\boldsymbol{Z}\boldsymbol{X} + \boldsymbol{X}d\boldsymbol{Z}}{2}$ を $d\boldsymbol{X}$ について解く (Lyapunov 方程式).
10. AHO 方向 $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ を得る.

なお、SDPA では Lyapunov 方程式は次のように解いている.

1. $\dfrac{\boldsymbol{V}\boldsymbol{U} + \boldsymbol{U}\boldsymbol{V}}{2} = \boldsymbol{C}$ を $\boldsymbol{V}$ について解くとする. ただし、$\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ と固有値分解しておく.
2. $\boldsymbol{F} = \boldsymbol{Q}^T\boldsymbol{C}\boldsymbol{Q}$ を計算する.
3. $\dfrac{\boldsymbol{\Lambda}\boldsymbol{V}' + \boldsymbol{V}'\boldsymbol{\Lambda}}{2} = \boldsymbol{F}$ を $\boldsymbol{V}'$ について解く.
   $\left(\boldsymbol{V}'_{ij} = 2\boldsymbol{F}_{ij}/(\boldsymbol{\Lambda}_{ii} + \boldsymbol{\Lambda}_{jj})\right)$
4. $\boldsymbol{V} = \boldsymbol{Q}\boldsymbol{V}'\boldsymbol{Q}^T$ を計算する.

3.3 対称行列の最小固有値の高速な近似法について

SDPA を実行する際には、各反復において対称行列の最小固有値を計算しなければならない場所が数箇所存在する. 一つは、主問題及び双対問題のステップ長を計算するところで

ある．ここで $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ が現在の反復において $\boldsymbol{X} \succ \boldsymbol{O}$ と $\boldsymbol{Z} \succ \boldsymbol{O}$ を満たしていて、さらに $(d\boldsymbol{X}, d\boldsymbol{y}, d\boldsymbol{Z})$ をそのとき計算して得られた探索方向とする．この場合には、主問題のステップ長 α_p は次のように計算することができる．

$$\bar{\alpha}_p = \max\{\alpha \in [0,1] : \boldsymbol{X} + \alpha d\boldsymbol{X} \succeq \boldsymbol{O}\},\ \alpha_p = \gamma\bar{\alpha}_p,$$

このとき $\gamma \in (0,1)$ は前もって設定されているパラメーターである．コレスキー分解 (Cholesky factorization) を正定値行列である $\boldsymbol{X}$ に対して行うことによって, $\boldsymbol{X} = \boldsymbol{L}\boldsymbol{L}^T$ となるような $n \times n$ の下三角行列 $\boldsymbol{L}$ を得ることができる．このとき $\bar{\alpha}_p$ 次のように書き直すことができる．

$$\begin{aligned}
\bar{\alpha}_p &= \max\{\alpha \in [0,1] : \boldsymbol{I} + \alpha\boldsymbol{L}^{-1}d\boldsymbol{X}\boldsymbol{L}^{-T} \succeq \boldsymbol{O}\}, \\
&= \begin{cases} \min\{-1/\xi_{\min},\ 1\} & \xi_{\min} < 0, \\ 1 & \text{その他}, \end{cases}
\end{aligned}$$

ここで $\xi_{\min}$ は行列 $\boldsymbol{L}^{-1}d\boldsymbol{X}\boldsymbol{L}^{-T}$ の最小固有値を表している．また同時に $\boldsymbol{X}^{-1}d\boldsymbol{X}$ の最小固有値でもある．双対問題のステップ長 α_d も同様に計算することができる．

$$\begin{aligned}
\bar{\alpha}_d &= \begin{cases} \min\{-1/\eta_{\min},\ 1\} & \eta_{\min} < 0, \\ 1 & \text{その他}, \end{cases} \\
\alpha_d &= \gamma\bar{\alpha}_d,
\end{aligned},$$

ここで $\eta_{\min}$ は $\boldsymbol{M}^{-1}d\boldsymbol{X}\boldsymbol{M}^{-T}$ の最小固有値を表している．また $\boldsymbol{M}$ は $\boldsymbol{Z}$ をコレスキー分解することによって得られ $\boldsymbol{Z} = \boldsymbol{M}\boldsymbol{M}^T$ である．

SDPA は SDP (2.1) の (近似) 最適解に達する前に、現在の反復における解 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z}) \in \mathcal{S}_+ \times R^m \times \mathcal{S}_+$ が領域 $(\mathcal{S}_+ \times R^m \times \mathcal{S}_+)$ の境界に接近し過ぎないように監視をしている．ここで $\mathcal{S}_+$ は $n \times n$ の対称半正定値行列の集合とする．そこで、現在の解が境界からどれくらい離れているかの指標が必要になってくる．この目的のため、現在の解 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ から領域 $(\mathcal{S}_+ \times R^m \times \mathcal{S}_+)$ の境界までの距離を評価するために次の指標を導入する．

$$\frac{\boldsymbol{X} \bullet \boldsymbol{Z}/n - \boldsymbol{X}\boldsymbol{Z}\text{の最小固有値}}{\boldsymbol{X} \bullet \boldsymbol{Z}/n} \in [0,1)$$

$\boldsymbol{X} \bullet \boldsymbol{Z}/n$ は $\boldsymbol{X}\boldsymbol{Z}$ の全ての (正の) 固有値の平均値と一致する．現在の解 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ が中心パスに近い場所にあれば、$\boldsymbol{X}\boldsymbol{Z}$ は単位行列の定数倍 $(\mu\boldsymbol{I})$ に近い行列になるので全ての固有値の値が近くなり、結果として上式は 0 に近い値をとることになる．よって $\boldsymbol{X}\boldsymbol{Z} = \mu\boldsymbol{I}$ ならば、上式の値は 0 になる．反対に現在の反復における解 $(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{Z})$ が境界に接近すれば、最小固有値の値は 0 に近づき上式の値は 1 に接近していく．そこで、前もってパラメーター $\delta \in [0,1]$ を決めておき、このパラメーター値を越えたら SDPA は中心パス付近に次の解を戻す操作を行う．ここで $\boldsymbol{X}\boldsymbol{Z}$ の最小固有値は、対称行列 $\boldsymbol{L}^T\boldsymbol{Z}\boldsymbol{L}$ の最小固有値の値と一致する ($\boldsymbol{X} = \boldsymbol{L}\boldsymbol{L}^T$ であり、コレスキー分解を用いて求める).

この節の残りでは、$n \times n$ の対称行列 $\boldsymbol{U} \in \mathcal{S}$ の最小固有値を効率良く求める方法を紹介する．この方法は対称三重対角行列の固有値を求める 2 分割法として知られている．まず、ハウスホルダー変換 (Householder tridiagonalization) によって行列 $\boldsymbol{U}$ を三重対角行列 $\boldsymbol{T}$ に

変換する。このとき $\boldsymbol{T}$ と $\boldsymbol{U}$ は、同じ固有値を持つ ([14]). 次に、$\boldsymbol{T}$ の $r \times r$ の大きさを持つ部分行列 (principal submatrix) である $\boldsymbol{T}_r (1 \leq r \leq n)$ を導入する.

$$\boldsymbol{T}_r = \begin{pmatrix} a_1 & b_2 & & \dots & 0 \\ b_2 & a_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & a_{r-1} & b_r \\ 0 & \dots & & b_r & a_r \end{pmatrix}.$$

まず $\boldsymbol{T} = \boldsymbol{T}_n$ と定義し、ここである b_r $(2 \leq r \leq n)$ が 0 であると仮定する. この場合、もし T を次のように書き直すことが出来るならば、

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{T}_r & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix},$$

$\lambda_{\min}(\boldsymbol{T}) = \min\{\lambda_{\min}(\boldsymbol{T}_r),\ \lambda_{\min}(\boldsymbol{S})\}$ となる. つまり、$\boldsymbol{T}_r$ と $\boldsymbol{S}$ 別々に最小固有値を求めることになる. このとき $\boldsymbol{T}_r$ と $\boldsymbol{S}$ のどちらとも大きさ $n \times n$ 以下の三重対角行列である. よってこれら $\boldsymbol{T}_r$ と $\boldsymbol{S}$ に対して同じような議論を展開することができる. それゆえ、b_r $(2 \leq r \leq n)$ は全て 0 でないと仮定してもよい.

新たに、以下の多項式 $p_0(\lambda) = 1$ と $p_r(\lambda) = \det(\boldsymbol{T}_r - \lambda \boldsymbol{I})$ $(1 \leq r \leq n)$ を定義する. ここで、計算したい $\boldsymbol{T}$ の最小固有値 $\lambda_{\min}(\boldsymbol{T})$ は、多項式 $p_r(\lambda)$ の最小根に対応する. $p_r(\lambda) = \det(\boldsymbol{T}_r - \lambda \boldsymbol{I})$ は次のような漸化式を用いて計算することが可能である.

$$\begin{aligned} p_0(\lambda) &= 1 \\ p_1(\lambda) &= a_1 - \lambda \\ p_r(\lambda) &= (a_r - \lambda) p_{r-1}(\lambda) - b_r^2 p_{r-2}(\lambda) \quad (2 \leq r \leq n). \end{aligned}$$

定理 3.3.1 Strum Sequence Property [14, Theorem 8.5-1]
$b_i \neq 0$ $(1 \leq i \leq n-1)$ であると仮定する. このとき、

(i) ある $k \in \{1, 2, \dots, n-1\}$ に対して $p_k(\lambda) = 0$ ならば $p_{k+1}(\lambda) p_{k-1}(\lambda) < 0$ である.

(ii) $\lambda \in R$ として、$\sigma(\lambda)$ は次の数列の符号の変化する回数を示すものとする.

$$\{p_0(\lambda), p_1(\lambda), \dots, p_n(\lambda)\}.$$

このとき $\sigma(\lambda)$ は、λ より小さい $\boldsymbol{T}$ の固有値の数に等しい. ここで $p_{r-1}(\lambda) \neq 0$ でかつ $p_r(\lambda) \times p_{r-1} \leq 0$ ならば $p_r(\lambda)$ と $p_{r-1}(\lambda)$ は異なる符号を持つものとする. また $p_{r-1}(\lambda) = 0$ または $p_r(\lambda) \times p_{r-1} > 0$ ならば $p_r(\lambda)$ と $p_{r-1}(\lambda)$ は同じ符号を持つものとする.

Gerschgorin circle 定理 ([14, Theorem 7.2-1]) から, $\lambda_{min}(\boldsymbol{T}) \in [\underline{\lambda}, \overline{\lambda}]$ が言える. このとき、

$$\underline{\lambda} = \min_{1 \leq i \leq n} a_i - |b_i| - |b_{i+1}|\ ,\ \overline{\lambda} = \max_{1 \leq i \leq n} a_i + |b_i| + |b_{i+1}|.$$

である．ここで $b_1 = b_{n+1} = 0$ を仮定する．

$\underline{\lambda} < \lambda < \overline{\lambda}$ として、$p_0(\lambda) = 1 > 0$ なので上の定理から次のことが言える．もし、ある $r \in \{1, 2, \ldots, n\}$ の対して

$$p_k(\lambda) > 0 \ (0 \leq k < r) \ \text{かつ} \ p_r(\lambda) \leq 0 \tag{3.29}$$

ならば、$\lambda_{\min}(\boldsymbol{T}) \in [\underline{\lambda}, \lambda]$ であり、そうでなければ $\lambda_{\min}(\boldsymbol{T}) \in [\lambda, \overline{\lambda}]$ である．

これまでの議論を要約すると、次にような $\lambda_{\min}(\boldsymbol{T})$ の近似値を求める 2 分割法を作成することができる．

$\lambda_{min}(\boldsymbol{T})$ の近似値を求める 2 分割法:

Step 0:

$$\underline{\lambda} = \min_{1 \leq i \leq n} a_i - |b_i| - |b_{i-1}|, \ \overline{\lambda} = \max_{1 \leq i \leq n} a_i + |b_i| + |b_{i-1}| \ \text{とする．}$$

Step 1: $|\overline{\lambda} - \underline{\lambda}| < \epsilon(|\underline{\lambda}| + |\overline{\lambda}|)$ ならば、結果 $(\underline{\lambda})$ を出力してアルゴリズムを終了する．
(ϵ は十分小さい値を取る).

Step 2: $\lambda = (\underline{\lambda} + \overline{\lambda})/2$, $k = 1$ また $q_1(\lambda) = a_1 - \lambda$ とする．

Step 3: $p_k(\lambda) \leq 0$ ならば, $\overline{\lambda} = \lambda$ として Step 1 に戻る．

Step 4: $k = k + 1$.

Step 5: $k > n$ ならば, $\underline{\lambda} = \lambda$ として Step 1 に戻る．

Step 6: $p_k(\lambda) = (a_k - \lambda)p_{k-1}(\lambda) - b_k^2 p_{k-2}(\lambda)$ として Step 3 に戻る．

注意 3.3.2 実用的には, $\{p_k(\lambda) : k = 1, 2, \ldots, n\}$ の数列を生成していくことは、しばしば数値的不安定性を引き起こすことがある．つまり、大きい k に対して $p_k(\lambda)$ が桁あふれ (overflow) が発生するとこがある．そこで、SDPA では新たに数列 $\{q_k(\lambda) : k = 1, 2, \ldots, n\}$ を定義して $\{p_k(\lambda) : k = 0, 1, 2, \ldots, n\}$ の代わりに用いる．全ての $r = 1, 2, \ldots, n$ と $\lambda \in R$ に対して、

$$q_r(\lambda) \ = \ \begin{cases} p_r(\lambda)/p_{r-1}(\lambda) & p_{r-1}(\lambda) \neq 0, \\ +\infty & \text{その他.} \end{cases}$$

とする．また全ての $\lambda \in R$ に対して, $q_{r-1}(\lambda) \neq 0$ である限り、$q_r(\lambda)$ $(1 \leq r \leq n)$ を次の式を用いて再帰的に計算することができる．

$$q_r(\lambda) = a_r - \lambda - b_r^2/q_{r-1}(\lambda)$$

ここで $b_1 = 0$ とする．元の定義により, (3.29) が成り立つことと、

$$q_k(\lambda) > 0 \ (1 \leq k < r) \ \text{かつ} \ q_r(\lambda) \leq 0 \tag{3.30}$$

が成り立つことは、必要十分である．それゆえ、矛盾することなく $\{p_k(\lambda) : k = 0, 1, 2, \ldots, n\}$ のかわりに、$\{q_k(\lambda) : k = 1, 2, \ldots, n\}$ を利用することができる (数値的安定性も増している)．このとき、Step 6 を次の Step 6' で置き換える．

Step 6': Let $q_k(\lambda) = (a_k - \lambda) - b_k^2/q_{k-1}(\lambda)$ として Step 3 に戻る．

第 4章

SDPA の実験的解析

この章では SDPA の実験的解析について述べる．主な項目は以下の通りである．

1. SDPA の数値実験で用いる問題の説明:
   主に 4.4 節で使用する問題の説明を行う．ここで説明する以外の問題を使用するときはその都度説明を行う．

2. 3.2 節で提案した探索方向の計算方法の有効性の検証:
   前に提案した探索方向の計算方法は、理論的にも優れた性質を持っている (定理 3.2.1)．よって n, m や $\boldsymbol{A}_i$ の密度など様々な性質を持った問題を用いて実用上での有効性を検証する．

3. 3.3 節で説明した固有値計算方法の有効性の検証:
   SDP に対するソフトウェアでは初めて実装されたこの計算方法の有効性を検証する．後の実験結果が示すように、入力問題によっては固有値計算の時間が全体の実行時間の中で大きな割合を占めてくるので、全体の高速化のためにも非常に重要である．

4. SDPA の数値実験による評価と解析:
   4.1 節で解説した問題を用いて、SDPA の性能 (実行時間、反復回数及び終了時の解の精度など) を調べると共に、SDPT3 [50] との比較実験も併せて行う．入力問題と SDPA の主要な部分の実行時間の比率の関係についても調べる．

5. 入力行列の大きさや密度を変化させたときの SDPA の主要な部分及び関数の実行時間 (回数) の解析:
   前節では、行列 $\boldsymbol{B}$ の計算といった機能別、目的別に実行時間を測定したが、この節では、行列同士の乗算といったような主要な関数にも着目する．人工的に n, m や $\boldsymbol{A}_i$ の密度などを変化させた問題を作成して、これらの要因によって実行時間の比率がどのように変化するかを総合的に検証する．

4 での全ての数値実験は、DEC AlphaServer 8400 (CPU Alpha 21164A-437MHz : 8GB 主記憶装置) で行った．その他の数値実験は、DEC Alpha (CPU Alpha 21164A-300MHz : 256MB 主記憶装置) で行い、OS は共に DIGITAL UNIX V3.2G てある．なおこの論文には

記載していないが、実験結果に DEC Alpha 特有の現象が起きていないことを確認するために他の幾つかのマシンや OS、例えば Sun Ultra-1 (Ultra-SPARC 143MHz : Solaris 2.5) や , PC (Pentium 166MHz : Linux および FreeBSD) 等でも数値実験を行って、同等で同様の結果が得られることを確認した．また、全ての数値実験において HRVW/KSH/M 探索方向を用いる．

4.1 SDPA の数値実験に用いる問題の説明

この節ではランダムに作成した問題から SDP の代表的な応用問題まで、７種類の問題を集めてそれぞれ説明を行う．なお次節以降ではこれらの問題を使用して数値実験を行い、SDPA の評価と解析を達成する．

4.1.1 乱数を用いて作成した全ての要素が非零の SDP

最初の問題例は $\boldsymbol{A}_i \in \mathcal{S}$ $(1 \leq i \leq m)$ の全ての要素が非零の SDP の等式標準形 (2.1) である．標準正規分布 $\mathcal{N}(0,1)$ を使用して、$\boldsymbol{A}_i$ $(1 \leq i \leq m)$ の各要素を作成する．このとき $\boldsymbol{A}_0 \in \mathcal{S}$ と $\boldsymbol{b} \in R^m$ は SDP (2.1) が実行可能内点を持つように作成する．

4.1.2 ノルム最小化問題 (Norm Minimization Problem)

$\boldsymbol{F}_i \in R^{q \times r}$ $(0 \leq i \leq p)$ とする．このときノルム最小化問題は次のように定義することができる．

$$\begin{array}{ll} \text{minimize} & \left\| \boldsymbol{F}_0 + \sum_{i=1}^{p} \boldsymbol{F}_i y_i \right\| \\ \text{subject to} & y_i \in R \ (1 \leq i \leq p). \end{array}$$

ここで $\|\boldsymbol{C}\|$ は行列 $\boldsymbol{C}$ の 2-ノルム を示す, つまり $\|\boldsymbol{C}\| = \max_{\|\boldsymbol{u}\|=1} \|\boldsymbol{C}\boldsymbol{u}\| =$ 行列 $\boldsymbol{C}^T\boldsymbol{C}$ の最大固有値の平方根である．そして、このノルム最小化問題は次のような SDP として定式化することができる．

$$\begin{array}{ll} \text{maximize} & -y_{p+1} \\ \text{subject to} & \sum_{i=1}^{p} \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_i^T \\ \boldsymbol{F}_i & \boldsymbol{O} \end{pmatrix} y_i + \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} y_{p+1} + \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0^T \\ \boldsymbol{F}_0 & \boldsymbol{O} \end{pmatrix} \succeq \boldsymbol{O}. \end{array}$$

ここで次のように設定することによって、SDP の等式標準形 (2.1) の双対問題として定式化することができる．

$$\begin{aligned} m &= p+1,\ n = r+q,\ \boldsymbol{A}_0 = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0^T \\ \boldsymbol{F}_0 & \boldsymbol{O} \end{pmatrix}, \\ \boldsymbol{A}_i &= \begin{pmatrix} \boldsymbol{O} & -\boldsymbol{F}_i^T \\ -\boldsymbol{F}_i & \boldsymbol{O} \end{pmatrix},\ b_i = 0 \ (1 \leq i \leq p), \\ \boldsymbol{A}_{p+1} &= \begin{pmatrix} -\boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & -\boldsymbol{I} \end{pmatrix},\ b_{p+1} = -1, \end{aligned}$$

4.1.3 行列のチェビシェフ近似問題 (Chebyshev Approximation Problem for a Matrix)

この問題はノルム最小化問題の特殊形であり、実数正方行列 $\boldsymbol{F} \in R^{p\times p}$ が与えられたときに, この問題は次のように定式化することができる.

$$\begin{array}{ll} \text{minimize} & \left\|\boldsymbol{F}^r + \sum_{j=1}^r x_j \boldsymbol{F}^{j-1}\right\| \\ \text{subject to} & \boldsymbol{x} \in R^r, \end{array}$$

ここで $\boldsymbol{F}^0 = \boldsymbol{I}$ を仮定する．しかし行列集合 $\{\boldsymbol{I}, \boldsymbol{F}^1, \ldots, \boldsymbol{F}^r\}$ は、数値的安定性や収束性といった意味で優れた行列基底 (matrix basis) を構成しないことが知られている．そこで、優れた収束性や数値的安定性を得るために集合 $\{\boldsymbol{I}, \boldsymbol{F}^1, \ldots, \boldsymbol{F}^r\}$ の変わりに 直行正規化基底 $\{\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_{r+1}\}$ を使用する．この基底は、集合 $\{\boldsymbol{I}, \boldsymbol{F}^1, \ldots, \boldsymbol{F}^r\}$ から修正グラハム-シュミット直行正規化法 (a modified Gram-Schmidt orthonormalization procedure) を用いて得られる．これは [50, Section 5](詳しくは [51]) によって提案された．さらにソフトウェア SDPT3[50] に所属する MATLAB の関数 "chebymat.m" にもこのようなチェビシェフ近似問題を生成する機能が備わっている．今回に数値実験においては使用する問題をこの関数から生成した.

4.1.4 制御とシステム理論分野に対する SDP

$\boldsymbol{P} \in R^{\ell\times\ell}$, $\boldsymbol{Q} \in R^{\ell\times k}$, $\boldsymbol{R} \in R^{k\times\ell}$ とする．ここで次のような式で表される SDP を考える.

$$\begin{array}{ll} \text{minimize} & -\lambda \\ \text{subject to} & \begin{pmatrix} -\boldsymbol{P}^T\boldsymbol{S} - \boldsymbol{S}\boldsymbol{P} - \boldsymbol{R}^T\boldsymbol{D}\boldsymbol{R} & -\boldsymbol{S}\boldsymbol{Q} \\ -\boldsymbol{Q}^T\boldsymbol{S} & \boldsymbol{D} \end{pmatrix} \succeq \lambda \boldsymbol{I},\ \boldsymbol{S} \succeq \lambda \boldsymbol{I}, \end{array}$$

ここで、$\boldsymbol{D}$ は大きさ $k\times k$ の対角行列 $\boldsymbol{D} = \mathrm{diag}(d_1, d_2, \ldots, d_k)$、$\boldsymbol{S}$ は大きさ $\ell\times\ell$ の対称行列 $\boldsymbol{S} \in \mathcal{S}^\ell$ である．また λ は実数である．この問題は、次の線形系に不変楕円対 (invariant ellipsoid) が存在するかを判定する問題である.

$$\frac{d\boldsymbol{x}(t)}{dt} = \boldsymbol{P}\boldsymbol{x}(t) + \boldsymbol{Q}\boldsymbol{u}(t), \boldsymbol{y}(t) = \boldsymbol{R}\boldsymbol{x}(t),\ |u_i(t)| \le y_i(t)\ (1 \le i \le k).$$

詳しくは [53] を参照のこと.

上記の SDP は、SDP の等式標準形 (2.1) の双対問題として定式化することができる．ただし、$m = \ell(\ell+1)/2 + k + 1$, $n = 2k + \ell$ である．数値実験では、データ (定数) 行列である $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$ は乱数を用いて生成する.

4.1.5 最大カット問題 (Maximum Cut Problem) に対する SDP 緩和

$G = (V, E)$ を完全無向グラフ、$V = \{1, 2, \ldots, n\}$ を頂点集合、そして $E = \{(i,j) : i, j \in V,\ i < j\}$ を枝集合とする．また全ての枝 ($(i,j) \in E$) には非負の重みが存在して、$C_{ij} = C_{ji}$

である．このとき、最大カット問題とはカット $(c(L,R) = \sum_{i\in L, j\in R} C_{ij})$ を最大にするような V の分割 (L,R) を見つける最大化問題である．次に変数ベクトル $\boldsymbol{u} \in R^n$ を導入して、この問題を非凸二次計画問題として定式化する．

$$\text{maximize} \quad \frac{1}{2}\sum_{i<j} C_{ij}(1-u_iu_j) \ \text{ subject to} \quad u_i^2 = 1 \ (1 \le i \le n).$$

この問題の実行可能解 $(\boldsymbol{u} \in R^n)$ とは、次のようなカット $(L,R) : (L = \{i \in V : u_i = -1\}$ かつ $R = \{i \in V : u_i = 1\})$ に相当する．ここで幾つか定義を行う． まず行列 $\boldsymbol{C}$ は大きさ $n \times n$ の対称行列であり、$C_{ji} = C_{ij}$ $((i,j) \in E)$, $C_{ii} = 0$ $(1 \le i \le n)$ であるとする．また、行列 $\boldsymbol{A}_0 \in \mathcal{S}$ は次の式で定義される大きさ $n \times n$ の対称行列である．

$$\boldsymbol{A}_0 = \mathrm{diag}(\boldsymbol{Ce}) - \boldsymbol{C},$$

このとき $\boldsymbol{e} \in R^n$ は全ての要素が 1 のベクトルであり、$\mathrm{diag}(\boldsymbol{Ce})$ は、ベクトル $\boldsymbol{Ce} \in R^n$ を対角部分に持つ対角行列である．これらの記号を用いて最大カット問題は次のような二次計画問題として定式化することができる．

$$\text{minimize} \quad -\boldsymbol{x}^T\boldsymbol{A}_0\boldsymbol{x} \ \text{subject to} \quad x_i^2 = 1/4 \ (1 \le i \le n).$$

もし、$\boldsymbol{x} \in R^n$ が後者の二次計画問題の実行可能解ならば、(i,j) 番目の要素が $X_{ij} = x_ix_j$ で与えられる半正定値行列 $\boldsymbol{X}$ は、$\boldsymbol{A}_0 \bullet \boldsymbol{X} = \boldsymbol{x}^T\boldsymbol{A}_0\boldsymbol{x}$ と $X_{ii} = 1/4$ $(1 \le i \le n)$ の条件を満たす．これらのことから次の最大カット問題への SDP 緩和を導くことができる．

$$\begin{array}{ll} \text{minimize} & -\boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1/4 \ (1 \le i \le n), \ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}$$

$\boldsymbol{E}_{ii}$ は大きさ $n \times n$ の対称行列を示し、(i,i) 要素は 1、他の要素は全て 0 である．この場合 $\mathrm{rank}(\boldsymbol{X}) = 1$ の条件を緩和していることになる．詳しくは、[19, etc.] を参照のこと．

4.1.6 グラフ分割問題 (Graph Partitioning Problem) に対する SDP 緩和

$G = (V,E)$ を完全無向グラフ、$V = \{1,2,\ldots,n\}$ を頂点集合、そして $E = \{(i,j) : i,j \in V, \ i < j\}$ を枝集合とする．また全ての枝 $((i,j) \in E)$ には、非負の重みが存在し、$C_{ij} = C_{ji}$ である．この場合 n が偶数であると仮定する．このときグラフ分割問題とはカット $(c(L,R) = \sum_{i\in L, j\in R} C_{ij})$ を最小にするような V の分割 (L,R) を見つける最小化問題である (ただし $|L| = |R| = n/2$)．次に変数ベクトル $\boldsymbol{u} \in R^n$ を導入して、この問題を非凸二次計画問題として定式化する．

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\sum_{i<j} C_{ij}(1-u_iu_j) \\ \text{subject to} & \left(\sum_{i=1}^n u_i\right)^2 = 0, \ u_i^2 = 1 \ (1 \le i \le n). \end{array}$$

最大カット問題と同様に、次のようにグラフ分割問題に対する SDP 緩和問題を次のように導くことができる.

$$\left.\begin{array}{ll} \text{minimize} & \boldsymbol{A}_0 \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ii} \bullet \boldsymbol{X} = 1/4 \ (1 \leq i \leq n),\ \boldsymbol{E} \bullet \boldsymbol{X} = 0,\ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}\right\} \tag{4.1}$$

$\boldsymbol{A}_0$ と $\boldsymbol{E}_{ii}$ $(1 \leq i \leq n)$ は、前節の最大カット問題と同じ行列である．そして $\boldsymbol{E}$ は、全ての要素が 1 の大きさ $n \times n$ の行列である．詳細は [19, etc.] を参照のこと.

4.1.7 最大クリーク問題 (Maximum Clique Problem) に対する SDP 緩和

$G = (V, E)$ を無向グラフ、$V = \{1, 2, \ldots, n\}$ を頂点集合、$E = \{(i, j) : i, j \in V,\ i < j\}$ を枝集合とする．ここで V の部分集合 C は、$i < j$ となるような全ての $i,\ j \in C$ に対して $(i, j) \in E$ ならばクリークという．最大クリーク問題とは、G の中で位数最大のクリークを見つける問題である．この問題に対しては、次の SDP を解くことによって G の最大クリーク数の上界が得られることが良く知られている．この数のことをロバッシュ数 (Lovász number) ともいう.

$$\left.\begin{array}{ll} \text{minimize} & -\boldsymbol{E} \bullet \boldsymbol{X} \\ \text{subject to} & \boldsymbol{E}_{ij} \bullet \boldsymbol{X} = 0 \ ((i, j) \notin E), \\ & \boldsymbol{I} \bullet \boldsymbol{X} = 1,\ \boldsymbol{X} \succeq \boldsymbol{O}. \end{array}\right\} \tag{4.2}$$

ここで、$\boldsymbol{E}$ は全ての要素が 1 の大きさ $n \times n$ の行列を示す．また $\boldsymbol{E}_{ij}$ は (i, j) と (j, i) 要素が 1/2 で、残りの要素が全て 0 の大きさ $n \times n$ の行列を示している．詳細は [19, etc.] を参照のこと.

この SDP は $m = n(n-1)/2 - |E| + 1$ 個の等式制約を持つ．このとき $|E|$ は枝集合 E の位数を示す．よって m は $\mathcal{O}(n^2)$ であって、変数行列 $\boldsymbol{X}$ の大きさ n よりもはるかに大きい．一方、全ての制約行列 $\boldsymbol{E}_{ij}$ $((i, j) \notin E)$ と $\boldsymbol{I}$ は疎行列である．$\boldsymbol{E}_{ij}$ $((i, j) \notin E)$ には、それぞれ非零要素が 2 個しかなく、$\boldsymbol{I}$ も n 個しか非零要素を持っていない.

4.2 探索方向の計算方法の有効性の検証

4.2.1 密行列と疎行列を混在させた SDP

ここで、探索方向計算の有効性の検証のために 4.1.1 節とは生成方法がやや異なる乱数を用いた SDP を導入する．まず p を正の整数とする．このとき次の関数 $f(\cdot; m, n, p) : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, n^2\}$ を定義する:

$$f(i; m, n, p) = \left[\frac{(n^2 - 1)(m - i)^p}{(m - 1)^p} + 1 \right].$$

このとき $[a]$ は a より小さい最大の整数をあらわす．そして、大きさ $n \times n$ の行列 $\boldsymbol{A}_i \in \mathcal{S}(1 \leq i \leq m)$ の非零要素の期待値が $f(i; m, n, p)$ となるように SDP を生成する．もし、$\boldsymbol{A}_i = \boldsymbol{O}$ ならば、少なくとも 非零要素を 1 つ持つように修正を施す．また p は、$f(i; m, n, p)$

の値を調節するパラメーターである．しかし p がどのような値を取っても $\boldsymbol{A}_1$ は n^2 個の非零要素を持ち、$\boldsymbol{A}_m$ は少なくとも 1 個の非零要素を持つようにしたい．今回の数値実験では $p = \log_2 n$ とする．この場合 $\boldsymbol{A}_{[m/2]}$ の非零要素の期待値は n になる．表 4.1 は数値実験結果を示している．各列 $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3, $\mathcal{F}$-$*(1.5)$ は、それぞれ計算式 $\mathcal{F}$-1, $\mathcal{F}$-2, $\mathcal{F}$-3 そして $\mathcal{F}$-$*(1.5)$ を使用したときの、1 反復における行列 $\boldsymbol{B}$ の計算に要する平均実行時間を秒で表したものである．また最後の列は $\mathcal{F}$-$*(1.5)$ を使用したときに、1 反復の平均実行時間から行列 $\boldsymbol{B}$ の平均実行時間を引いた時間を同じく秒で表している．1 反復の実行時間には、探索方向の計算、ステップ長の計算や固有値計算などが含まれている．また全ての実行時間は 10 反復の平均を取っている．また、q_1 と q_2 は (3.14) よって計算された $\boldsymbol{B}$ の行番号である．つまり、計算式 $\mathcal{F}$-$*(1.5)$ では、$0 < i \leq q_1$ ならばそれに対応する $\boldsymbol{B}$ の行を $\mathcal{F}$-1 で計算を行う、また $q_1 < i \leq q_2$ ならば $\mathcal{F}$-2 で、最後に $q_2 < i \leq m$ ならば、$\mathcal{F}$-3 で計算を行う．

表 4.1 から次のようなことが言える．

- $\mathcal{F}$-$*(1.5)$ の使用は有効である．
- m の値が大きくなるに連れて、$\boldsymbol{B}$ の計算が 1 反復に占める割合は非常に高くなる．

図 4.1 と図 4.2 は、1 反復における重み付き乗算の回数 $d_{*i}(\boldsymbol{\sigma}^*) = \min\{d_{1i}(\boldsymbol{\sigma}^*), d_{2i}(\boldsymbol{\sigma}^*), d_{3i}(\boldsymbol{\sigma}^*)\}$ と $B_{\sigma^*(i)\sigma^*(j)}$ $(i \leq j \leq m)$ の計算の平均実行時間を比較したものである．このとき、$p = 7$, $n = 128$, $m = 256$ である．この二つの図がほぼ一致していることより $d_{*i}(\boldsymbol{\sigma}^*)$ による計算時間の推定が非常に有効であるといえる．

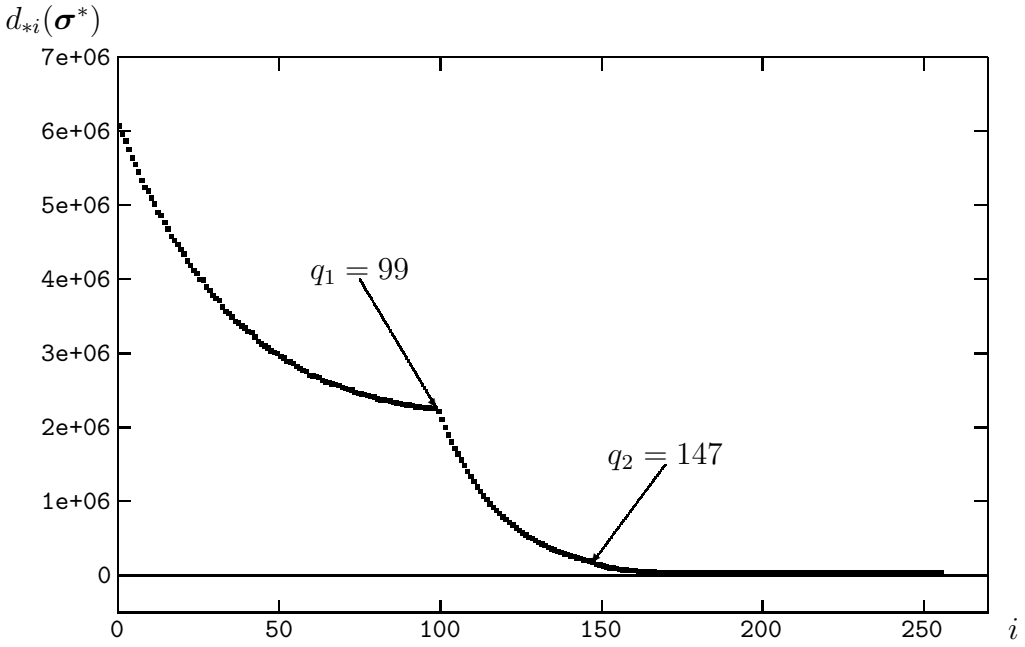

図 4.1: 重み付き乗算の回数 $d_{*i}(\sigma^*)$: $p = 7$, $n = 128$, $m = 256$

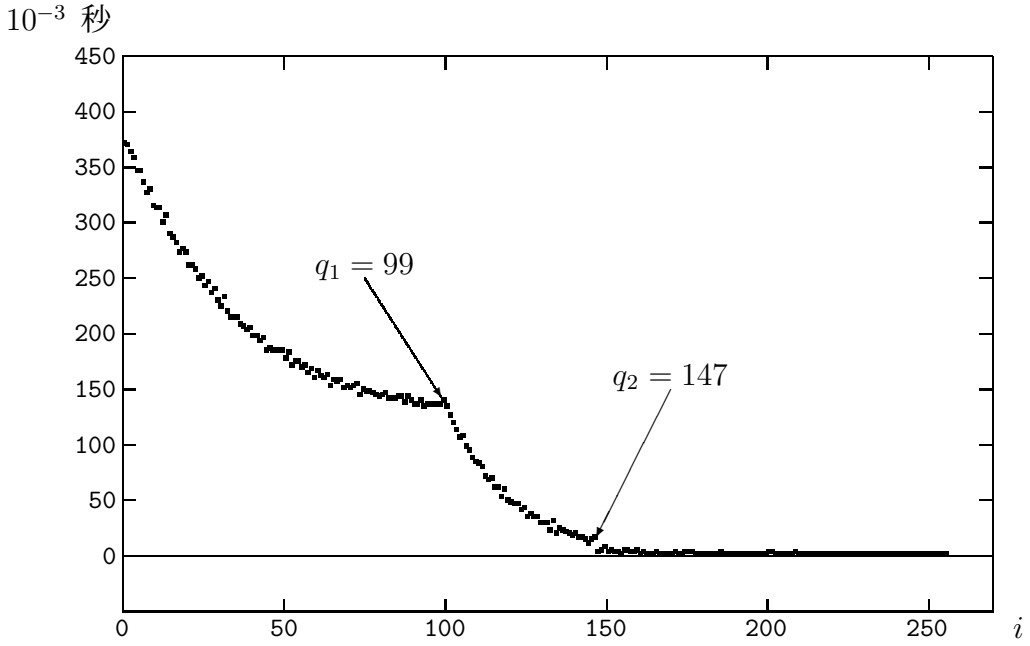

図 4.2: $B_{\sigma^*(i)\sigma^*(j)}$ $(i \le j \le m)$ の計算の平均実行時間: $p = 7$, $n = 128$, $m = 256$

4.2.2 二次割当問題

さらにもう一つ問題を導入する．二次割当問題の緩和にも SDP が用いられる場合がある ([43, 58]).

この場合も SDP の等式標準形 (2.1) に従って問題を作成する．$n = s^2+1$, $m = 2s+s^3+1$ であり、$2s$ 個の $\boldsymbol{A}_i$ は s^4 個の非零要素を持つ、また s^2 個の $\boldsymbol{A}_i$ は 3 個の非零要素を、(s^3-s^2) 個の $\boldsymbol{A}_i$ が 2 個の非零要素を、1 個の $\boldsymbol{A}_i$ が 1 個の非零要素をそれぞれ持つ．ここで s は二次割当問題の大きさを表す．この問題の場合には、

$$
\begin{aligned}
&f_{\sigma^*(i)} = s^4 \ (1 \le i \le 2s),\\
&f_{\sigma^*(i)} \in \{1,2,3\} \ (2s+1 \le i \le 2s+s^3+1),\\
&d_{1i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \ \text{ かつ } d_{1i}(\boldsymbol{\sigma}^*) < d_{3i}(\boldsymbol{\sigma}^*) \ (1 \le i \le 2s),\\
&d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*) \ \text{ かつ } d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \ (2s+1 \le i \le 2s+s^3+1)
\end{aligned}
$$

である．よって $q_1 = q_2 = 2s$ である．表 4.2 は数値実験結果を示している．この表から、$\mathcal{F}$-$*(1.5)$ の有効性を確認することができる．

4.2.3 グラフ分割問題

グラフ分割問題に関しては既に 4.1.6 節で説明を行った．この問題では $n = s$, $m = s+1$ である．また 1 個の $\boldsymbol{A}_i$ は s^2 個の非零要素を持つが、残りの s 個の $\boldsymbol{A}_i$ はわずか 1 個の非零要素しか持たない．ここで s は、グラフの点数を示す．この問題では、

$$f_{\sigma^*(1)} = s^2,$$

表 4.1: 行列の密度を変化させた SDP

p	n	m	$\mathcal{F}$-1	$\mathcal{F}$-2	$\mathcal{F}$-3	$\mathcal{F}$-$*(1.5)$	q_1	q_2	1 反復 $-\mathcal{F}$-$*(1.5)$
5	32	32	0.07	0.08	1.65	0.04	11	20	0.08
5	32	64	0.14	0.30	5.87	0.09	26	40	0.10
5	32	128	0.30	1.02	23.19	0.22	62	80	0.14
6	64	64	1.10	1.79	71.56	0.59	22	39	0.71
6	64	128	2.24	7.02	279.45	1.33	52	76	0.77
6	64	256	4.75	23.81	-	3.14	118	152	1.09
7	128	128	19.89	-	-	11.98	49	79	6.43
7	128	256	42.82	-	-	24.07	99	147	7.06

表 4.2: 二次割当問題.

s	n	m	$\mathcal{F}$-1	$\mathcal{F}$-2	$\mathcal{F}$-3	$\mathcal{F}$-$*(1.5)$	$q_1 = q_2$	1 反復 $-\mathcal{F}$-$*(1.5)$
5	26	136	0.14	0.12	2.15	0.04	10	0.10
6	37	229	0.67	0.48	11.73	0.13	12	0.34
7	50	358	3.33	1.91	56.52	0.42	14	1.32
8	65	529	8.29	4.65	-	0.90	16	3.16
9	82	748	22.89	11.79	-	1.99	18	11.08
10	101	1021	61.28	29.47	-	4.50	20	36.75

表 4.3: グラフ分割問題

s	t	n	m	$\mathcal{F}$-1	$\mathcal{F}$-2	$\mathcal{F}$-3	$\mathcal{F}$-$*$(1.5)	1 反復 $-\mathcal{F}$-$*$(1.5)
124	1271	124	125	21.15	0.56	22.13	0.36	6.36
250	2421	250	251	360.09	5.07	356.96	2.99	59.50
500	5120	500	501	7247.16	52.04	6341.55	29.29	607.23

$$
\begin{aligned}
&f_{\sigma^*(i)} = 1 \ (2 \leq i \leq s+1),\\
&d_{11}(\boldsymbol{\sigma}^*) < d_{21}(\boldsymbol{\sigma}^*) \text{ かつ } d_{11}(\boldsymbol{\sigma}^*) < d_{31}(\boldsymbol{\sigma}^*),\\
&d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*) \text{ かつ } d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \ (2 \leq i \leq s+1).
\end{aligned}
$$

となる．よって $q_1 = 1, q_2 = 1$ である．表 4.3 は数値実験結果を示す．ここで t はグラフの枝数を示す．この表からも同じく $\mathcal{F}$-$*$(1.5) の有効性を確認することができる．

4.2.4 最大クリーク問題

最大クリーク問題については 4.1.7 節で説明した通りである．この問題では $n = s$, $m = u+1$ である．1 個の $\boldsymbol{A}_i$ が s 個の非零要素を持ち、u 個の $\boldsymbol{A}_i$' が 2 個の非零要素を持つ．ここで s はグラフの点数を示している．また u は、2 点間に枝を持たない点の対の数を示している ("$n(n-1)/2 -$ グラフの枝数"). この問題では、

$$
\begin{aligned}
&f_{\sigma^*(1)} = s,\\
&f_{\sigma^*(i)} = 2 \ (2 \leq i \leq u+1),\\
&d_{21}(\boldsymbol{\sigma}^*) < d_{11}(\boldsymbol{\sigma}^*) \text{ かつ } d_{21}(\boldsymbol{\sigma}^*) < d_{31}(\boldsymbol{\sigma}^*),\\
&d_{3i}(\boldsymbol{\sigma}^*) < d_{1i}(\boldsymbol{\sigma}^*) \text{ かつ } d_{3i}(\boldsymbol{\sigma}^*) < d_{2i}(\boldsymbol{\sigma}^*) \ (2 \leq i \leq u+1).
\end{aligned}
$$

である．よって $q_1 = 0, q_2 = 1$ である．表 4.4 は数値実験結果を示していて、この場合も $\mathcal{F}$-$*$(1.5) が有効に動作している．

4.3 固有値計算方法の有効性の検証

通常、ある対称行列 $\boldsymbol{X} \in \mathcal{S}$ の固有値を求めるには $\mathcal{O}(n^3)$ の実行時間を要する．既に 3.3 節で述べたように主双対内点法の実行には最小固有値だけを求めればよい (実際には A.5 節で示すように SDPA は終了時に $\boldsymbol{X}, \boldsymbol{Z}$ そして $\boldsymbol{XZ}$ の全ての固有値を出力するので、このときには全ての固有値を計算する必要がある). しかし、最小固有値の近似値を求める 2 分割法も最初に行列のハウスホルダー変換を行う必要があるので最悪計算量は $\mathcal{O}(n^3)$ である．そこで数値実験によってこの計算方法の有効性の検証を行う．最小固有値だけを求めるのと全ての固有値を求めるのとでは、前者の計算の方が高速なのは自明であるかもしれない．しかし、この前者は手法としては知られているが、SDP のソフトウェアなどではま

表 4.4: 最大クリーク問題

s	u	n	m	$\mathcal{F}$-1	$\mathcal{F}$-2	$\mathcal{F}$-3	$\mathcal{F}$-$*$(1.5)	1 反復 $-\mathcal{F}$-$*$(1.5)
150	561	150	562	123.15	5.35	0.36	0.36	13.20
150	1101	150	1102	238.90	19.70	1.46	1.46	44.75
200	621	200	622	336.58	9.30	0.51	0.48	28.82
200	992	200	993	540.16	28.73	1.27	1.32	50.28
250	651	250	652	980.63	19.01	0.59	0.61	63.43
250	972	250	973	1396.54	40.86	1.54	1.33	80.04
300	943	300	944	2472.22	43.01	1.40	1.29	120.21

だ実装された例はなく、入力問題によっては、固有値計算の時間が大きな比率を占めるようになる．よって、SDPA でこの手法を実装し、数値実験により有効性を検証する．

この数値実験は次に示すような前提と条件で行う．

1. 3.3 節で示した方法 (BISECTION) と比較する方法は、同じくハウスホルダー変換によって得られる対称三重対角行列から全ての固有値を求める方法 (SYMEIG) である．
2. たとえ $\boldsymbol{A}_i$ が疎行列であっても固有値を求める行列は密になる．また m の大きさも関係しないので数値実験においては n だけ変化させればよい．
3. 以上の前提から数値実験には 4.1.1 節で説明した問題を用いる．

表 4.5: 固有値計算方法の比較実験結果.

n	BISECTION(秒)	SYMEIG(秒)	ハウスホルダー変換 (秒)
50	0.60	2.58	0.59
100	4.12	18.30	3.98
200	32.92	148.02	32.60
300	116.22	541.82	115.62

表 4.5 は、固有値計算の比較実験結果を示している．全ての数値実験は反復回数を 10 回に固定して行った．両計算方法共に $\mathcal{O}(n^3)$ であることは既に述べたが、実際の計算時間には大きな差が生じることが表から見て取ることができる．また BISECTION の方法では、ハウスホルダー変換がほとんどの実行時間を占めている．つまり、ハウスホルダー変換を行ったあとの処理 (3.3 節で示した 2 分割法) は極めて高速である．よって固有値の計算時間が全体の実行時間に占める割合が大きい場合には、BISECTION の方法が非常に有効になってくる．n, m そして $\boldsymbol{A}_i$ の密度による SDPA の各部分の実行時間の占める割合の変化については、4.4 節や 4.5 節などで実験、考察を行う．

4.4 SDPA の数値実験による評価と解析

この節では、4.1 節で説明した 7 種類のテスト問題に対して SDPA を用いた数値実験を行う．これらの例題を非零要素の割合によって 3 つに分類する．

dense problems	— 乱数を用いて作成した全ての要素が非零の SDP (4.1.1 節),
(密な問題)	ノルム最小化問題 (4.1.2 節),
	行列のチェビシェフ近似問題 (4.1.3 節),
mildly dense problems	— 制御とシステム理論分野に対する SDP
(やや密な問題)	(4.1.4 節),
sparse problems	— 最大カット問題に対する SDP 緩和 (4.1.5 節),
(疎な問題)	グラフ分割問題に対する SDP 緩和 (4.1.6 節),
	最大クリーク問題に対する SDP 緩和 (4.1.7 節).

この節の全ての数値実験は、DEC AlphaServer 8400 (CPU Alpha 21164A-437MHz : 8GB 主記憶装置)、OS は DIGITAL UNIX V3.2G である. 数値実験に関して次の 3 つの論点に注意を向けることにする．

1 つ目の論点は、SDPA の全体的な性能評価である．SDPA が妥当な実行時間でどのぐらい大きな問題を解くことができるかを調査する．そこで双対ギャップがあらかじめ設定した値 (10^{-5} または 10^{-8}) より小さくなったら近似最適解が求められたと判断して、そのときまでに要した反復回数と実行時間を報告する．

2 つ目の論点は、SDPA と SDPT3 [50] の比較実験である．両方のソフトウェアは、共に Mehrotra 型の主・双対 プリディクターコレクター内点法にも基づいて開発されている．正確には、Mehrotra-type primal-dual predictor-corrector interior-point method である．ただし、探索方向やステップ長などを制御するパラメータ等は異なっている．よって、できるだけ公平に数値実験を行うために、両方のソフトウェアで同じ初期点を用いる．両方のソフトウェア共に NT と AHO 方向も備えているが、今回の数値実験では HRVW/KSH/M 方向を用いる．さらに、両者の終了条件も同じにする．次に、SDPA と SDPT3 の大きく異なる点を幾つか示す.

(a) 開発言語が異なる: SDPA が C++ 言語で書かれている一方、SDPT3 は MATLAB で書かれている.

(b) SDPA は 3.2 節で解説した探索方向を求める際に疎行列を利用した高速な計算方法を組み入れている．また 3.3 節で解説した対称行列の最小固有値を高速に求める方法を備えている．

SDPA は、密な問題に適用した場合には SDPT3 のおおよそ 2 から 3 倍高速である．この場合主に (a) の理由によるものであると推察される．密な問題を解く場合には、図 3.3 のデータ構造は、二次元配列を用いるより、やや効率が落ちる．しかし、SDPA をやや密な問題や疎な問題に適用する場合には SDPA は SDPT3 よりはるかに高速である．これは、主に (b) の理由による．

最後の論点は、SDPA の実行時間の解析である．実行時間を多く消費する部分を幾つか選択して、問題の大きさ (n,m) 及び $\boldsymbol{A}_i$ の非零要素の密度を変化させたときに、実行時間を多く消費する部分がどう変化するかを調べる．次に、SDPA のアルゴリズムの中で実行時間を多く消費する部分 (major time-consuming parts) を列挙する．

(I) 大きさ $m \times m$ で全ての要素が非零の行列 $\boldsymbol{B}$ の計算 (3.2 節).

(II) 線形方程式 ($\boldsymbol{B}d\boldsymbol{y} = \boldsymbol{b}$) を解くための行列 $\boldsymbol{B}$ の LDL 分解 (3.2 節).

(III) 探索方向 $d\boldsymbol{X}$ と $d\boldsymbol{Z}$ の計算 (3.2 節).

(IV) 行列 $\boldsymbol{X}^{-1}d\boldsymbol{X}$, $\boldsymbol{Z}^{-1}d\boldsymbol{Z}$ 及び $\boldsymbol{X}\boldsymbol{Z}$ の最小固有値の計算 (3.3 節).

$\boldsymbol{X}$ と $\boldsymbol{Z}$ のコレスキー分解や逆行列 $\boldsymbol{Z}^{-1}$ の計算、それにステップ長の計算等の SDPA の他の部分は (I), (II), (III) および (IV) には含まれていない．

もし、疎行列を扱うことを考慮しない (つまり疎行列用のデータ構造や計算方法を使用しない) として、各部分の計算複雑性を記すと (I), (II), (III) および (IV) は、それぞれ $O(mn^3 + m^2n^2)$, $O(m^3)$, $O(n^3 + mn^2)$ および $O(n^3)$ 回の浮動小数点の演算が必要になる．多くの場合には (I) が最も多くの消費時間を必要とする．(I) は密あるいはやや密な問題では全体の実行時間の 80 から 95 % を占める．一方、疎な問題では疎行列を扱うためにこれまで解説してきたような種々の工夫を用いると (I) の実行時間に占める割合は 10 % 以下に減り、他の部分の割合の方が大きくなってくる．

今回の数値実験を通して、次の双対ギャップを定義し終了条件として使用する．

$$\frac{|P - D|}{\max\left\{1.0, \dfrac{|P| + |D|}{2}\right\}}$$

この双対ギャップがあらかじめ設定されている値 ϵ^* より小さくなったら SDPA はアルゴリズムを終了する．ここで、P と D はそれぞれ主問題と双対問題の目的関数値を示す.

4.4.1 乱数を用いて作成した全ての要素が非零の SDP

ここで扱う全ての SDP では、初期点は $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0) = (100\boldsymbol{I}, \boldsymbol{0}, 100\boldsymbol{I})$ に設定する．表 4.6 と 4.7 は数値実験結果を示す．アルゴリズムが終了するまでの反復回数は n と m が大きくなっても顕著な影響は受けない．また、全ての行列 $\boldsymbol{A}_i$ の要素は非零である．SDPA は $m/n = 1$ のときは、おおよそ 2 から 4 倍程度 SDPT3 より速いが、m/n の比が大きくなるにつれて、その差はさらに大きくなっていく．また反復回数という尺度から見ても、SDPA は SDPT3 と同程度の安定性と有効性を持っているといえる．

表 4.7 から, (I) つまり大きさ $m \times m$ の行列 $\boldsymbol{B}$ の計算が全体の計算時間の 90 % を占めていることがわかる．

表 4.6: 乱数を用いて作成した SDP に対する数値実験結果

		実行時間 (秒)		反復回数		双対ギャップ	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
10	10	0.1	0.4	9	9	4.10e-09	7.05e-09
20	20	0.4	1.7	10	9	7.57e-10	6.14e-09
25	25	0.8	2.4	9	9	9.02e-09	7.15e-10
30	30	1.5	4.2	10	9	3.98e-09	2.00e-09
50	50	10.9	35.6	10	9	6.61e-09	5.83e-09
50	200	65.9	285.1	11	10	2.81e-09	7.16e-09
100	100	129.3	367.8	10	10	8.90e-09	3.39e-09
100	200	433.3	2069.6	13	12	1.89e-09	6.19e-09
200	50	552.9	896.2	11	10	7.60e-10	9.10e-09
200	100	1130.1	2220.9	11	11	2.52e-09	4.96e-10

表 4.7: 乱数を用いて作成した SDP に対する SDPA の主要部分の実行時間

	$n = 100$, $m = 100$		$n = 100$, $m = 200$	
部分	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
(I)	116.80	90.32	413.07	95.34
(II)	0.05	0.04	0.47	0.11
(III)	2.60	2.01	5.88	1.36
(IV)	2.70	2.09	3.43	0.79
残りの部分	7.17	5.54	10.43	2.41

4.4.2 ノルム最小化問題と行列のチェビシェフ近似問題.

各ノルム最小化問題に対しては、次のように初期点 $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ を定める.

$$
\begin{aligned}
y_i^0 &= 0 \ (1 \leq i \leq p = m-1), \\
y_m^0 &= 1.1\|\boldsymbol{A}_0\|, \\
\boldsymbol{Z}^0 &= \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix} y_m^0 + \begin{pmatrix} \boldsymbol{O} & \boldsymbol{F}_0^T \\ \boldsymbol{F}_0 & \boldsymbol{O} \end{pmatrix} \succ \boldsymbol{O}, \\
\boldsymbol{X}^0 &= \frac{1}{n} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{pmatrix}.
\end{aligned}
$$

繰り返すがチェビシェフ近似問題はノルム最小化問題の特別形なのでノルム最小化問題に変換して、上記のようなノルム最小化問題と同じように初期点を生成して使用している. 表 4.8, 4.9 および 4.10 において数値実験結果を報告する. これらの問題において行列 $\boldsymbol{A}_i$ $(1 \leq i \leq p)$ のおおよそ 50 % の要素が非零である. しかし、(I) の全体実行時間に占める割合は、前節の $\boldsymbol{A}_i$ $(1 \leq i \leq p)$ の要素が全て非零の場合と比較してわずかに減っただけであり、以前として 85 % も占めている.

表 4.8: ノルム最小化問題に対する数値実験結果

		実行時間 (秒)		反復回数		双対ギャップ	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
20	10	0.1	0.7	9	9	1.11e-09	1.45e-09
40	20	1.1	5.4	9	9	3.90e-09	4.99e-09
50	25	3.5	8.3	9	9	8.24e-09	2.65e-09
60	30	5.4	15.9	9	9	3.26e-09	2.38e-09
100	50	38.3	165.7	10	10	6.30e-09	3.99e-10
200	100	721.2	1868.4	11	10	9.09e-10	9.65e-10

表 4.9: チェビシェフ近似問題に対する数値実験結果.

		実行時間 (秒)		反復回数		双対ギャップ	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
20	10	0.1	0.8	9	9	9.46e-10	6.67e-09
40	20	1.1	6.0	9	9	3.49e-09	4.45e-09
50	25	3.4	11.2	9	9	1.82e-09	8.20e-09
60	30	5.9	23.0	10	10	8.76e-10	1.14e-10
100	50	41.3	163.6	11	10	2.18e-09	3.72e-09
200	100	713.3	2613.3	11	10	6.86e-10	2.77e-09

表 4.10: ノルム最小化問題 (NMP) およびチェビシェフ近似問題 (CAP) に対する SDPA の主要部分の実行時間

	NMP, $n = 200$, $m = 100$		CAP, $n = 200$, $m = 100$	
部分	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
(I)	627.80	87.05	620.15	86.94
(II)	0.02	0.00	0.08	0.00
(III)	15.12	2.10	15.08	2.11
(IV)	24.82	3.44	24.88	3.49
残りの部分	53.44	7.41	53.1	7.44

4.4.3 制御とシステム理論分野に対する SDP

表 4.11 のように、実験データを $5 \leq k = l \leq 55$ に制限する．また $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0) = (10^i \boldsymbol{I}, \boldsymbol{0}, 10^i \boldsymbol{I})$ を初期点として使用する．ただし、$i = 4$ か 5 かは問題に依存する．今回用いた他の種類の問題と比較すると、反復回数が多めなのに気が付くが、この現象は主問題の実行可能領域が狭いために発生している可能性が高い．そのため、両者共に実行可能領域に達するのに多くの時間を要している．特に、 SDPT3 は双対ギャップが $1.0e-5$ より小さくなる前に、次のようなメッセージを出して終了する．“lack of progress in corrector”($k = \ell = 20$ の場合) と “lack of progress in predictor” ($k = \ell = 25$ の場合) である．(表 4.11 中の $\star$ を参照のこと)．しかし、多くの場合においては両方のソフトウェアは一度実行可能領域に達したら、その後数回の反復回数で近似最適解に達している．

表 4.11: 制御とシステム理論分野における SDP に対する数値実験結果

			実行時間 (秒)		反復回数		双対ギャップ	
$k = l$	n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
5	15	21	0.1	2.4	21	18	1.36e-07	4.21e-07
10	30	66	1.3	25.7	22	20	2.35e-07	8.10e-07
15	45	136	8.0	106.5	26	23	6.43e-07	3.97e-06
20	60	231	25.4	355.3	28	21	8.38e-07	1.18e-05*
25	75	351	114.9	1059.2	31	22	3.74e-08	2.02e-05*
30	90	496	221.0	-	32	-	6.67e-06	-
35	105	666	457.0	-	27	-	2.35e-06	-
40	120	861	630.1	-	26	-	8.42e-06	-
45	135	1081	1594.8	-	28	-	7.33e-06	-
50	150	1326	3217.5	-	37	-	6.21e-06	-
55	165	1596	6903.0	-	35	-	9.91e-06	-

表 4.12 では、表 4.11 の中から特に大きな問題を二つ取り上げて、SDPA の主要部分の実行時間を報告する．表 4.7 と 4.10 と比較すると、(II) つまり大きさ $m \times m$ の行列の LDL

分解が実行時間の中でより大きな割合を占めるようになっている．これは、m が表 4.7 と 4.10 の場合と比較して、おおよそ 9 から 10 倍になっているからである．

表 4.12: 制御とシステム理論分野における SDP に対する SDPA の主要部分の実行時間

	$n = 150, m = 1326$		$n = 165, m = 1596$	
部分	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
(I)	2553.15	79.35	5582.65	80.87
(II)	572.33	16.86	1194.02	17.30
(III)	9.60	0.30	16.28	0.24
(IV)	11.10	0.34	14.32	0.21
残りの部分	71.35	1.55	95.76	1.39

4.4.4 最大カット問題とグラフ分割問題に対する SDP 緩和

最大カット問題においては、実行可能初期点を簡単に生成することができるので、$(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ を次のように設定する．

$$\begin{aligned} \boldsymbol{X}^0 &= \mathrm{Diag}(\boldsymbol{b}), \\ \boldsymbol{y}^0 &= -1.1 \cdot \mathrm{abs}(\boldsymbol{A}_0) \cdot \boldsymbol{e}, \\ \boldsymbol{Z}^0 &= \boldsymbol{A}_0 - \mathrm{Diag}(\boldsymbol{y}^0). \end{aligned}$$

一方、グラフ分割問題では、初期点 $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0)$ を $(100\boldsymbol{I}, \boldsymbol{0}, 100\boldsymbol{I})$ に設定する．表 4.13, 4.14 および 4.15 は数値実験結果を示している．

最大カット問題とグラフ分割問題は疎な問題である．特に最大カット問題では全ての $\boldsymbol{A}_i$ $(1 \leq i \leq m)$ がたった一つしか非零要素を持っていない．またグラフ分割問題では、一つの $\boldsymbol{A}_i$ たけが n^2 個の非零要素を持つ密な行列であるが、残りの全ての行列は全て非零要素を１つしか持っていない．このような疎な問題に対しては、3.2 節で解説した計算方法が非常に効率良く働くことが先の数値実験からも予想される．事実、両方の問題に対して SDPA は SDPT3 よりはるかに高速である．表 4.15 から、最大カット問題では (I) の全体の実行時間への占める割合はわずか 1 % であり、グラフ分割問題に対しても 7 % 以下であることが読み取れる．

4.4.5 最大クリーク問題に対する SDP 緩和

この問題においては、SDPA は初期点 $(\boldsymbol{X}^0, \boldsymbol{y}^0, \boldsymbol{Z}^0) = (100\boldsymbol{I}, \boldsymbol{0}, 100\boldsymbol{I})$ から出発する．表 4.16 は 4.17 数値実験結果を示す．ここで m の値は、

“2 点間に枝を持たない点の対の総数 $n(n-1)/2 - |E|$ ” $+ 1$.

このことは、つまり m が $O(n^2)$ であることを示している．そのため表 4.16 からは、実行時間が m の値に強く依存していることが読み取れる．また表 4.17 より、m/n の比が大き

表 4.13: 最大カット問題に対する数値実験結果

	実行時間 (秒)		反復回数		双対ギャップ	
n	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
10	0.1	0.4	8	8	2.08e-09	5.51e-10
20	0.1	0.9	9	8	1.67e-09	9.16e-09
25	0.2	1.5	9	9	3.61e-09	5.68e-09
30	0.3	2.5	10	10	3.50e-09	2.95e-10
50	1.5	7.9	9	9	3.21e-09	3.66e-09
100	9.0	73.4	10	11	1.57e-09	2.54e-10
150	40.3	277.6	10	10	1.77e-09	1.89e-09
200	92.8	865.2	11	10	7.12e-10	7.13e-09
250	311.6	-	13	-	2.95e-09	-
500	2998.6	-	16	-	1.22e-09	-
1000	69490.1	-	16	-	3.59e-08	-
1250	111615.9	-	18	-	5.90e-09	-

表 4.14: グラフ分割問題に対する数値実験結果

	実行時間 (秒)		反復回数		双対ギャップ	
n	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
10	0.1	0.5	11	12	2.47e-06	9.83e-06
20	0.2	1.7	12	14	2.24e-06	4.38e-06
25	0.3	2.5	12	13	7.22e-06	7.23e-06
30	0.5	4.1	12	13	3.53e-06	4.36e-06
50	2.3	13.7	12	13	8.02e-07	4.31e-06
100	12.5	104.6	12	13	6.18e-06	8.67e-06
150	52.1	433.0	11	13	5.58e-06	4.25e-06
200	115.1	1174.1	12	12	2.39e-06	9.05e-06
250	418.6	-	15	-	2.61e-06	-
500	3197.9	-	15	-	1.22e-09	-
1000	63130.7	-	21	-	4.80e-06	-
1250	112375.7	-	15	-	4.16e-05	-

表 4.15: 最大カット問題 (MCP) とグラフ分割問題 (GPP) に対する SDPA の主要部分の実行時間

	(MCP) $n = 1250$		(GPP) $n = 1250$	
部分	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
(I)	26.07	0.02	7570.17	6.74
(II)	197.68	0.18	171.43	0.15
(III)	20469.43	18.34	17803.15	15.84
(IV)	16644.95	14.91	14686.77	13.07
その他の部分	74277.80	66.55	72144.17	64.20

いと (II) の比率が非常に高まることが読み取れる．しかし、疎な問題であるため (I) の占める割合は小さく押さえられている．

表 4.16: 最大クリーク問題に対する数値実験結果

		実行時間 (秒)		反復回数		双対ギャップ	
n	m	SDPA	SDPT3	SDPA	SDPT3	SDPA	SDPT3
50	132	2.8	49.8	15	15	7.43e-09	6.37e-10
50	253	4.8	147.9	15	14	3.85e-09	4.91e-09
50	597	20.8	716.7	14	13	5.72e-10	4.17e-09
100	539	30.2	1102.0	16	14	9.73e-10	6.33e-09
100	1024	143.0	4298.9	16	15	3.49e-09	6.26e-09
150	562	84.9	-	16	-	7.80e-09	-
150	1102	215.2	-	16	-	1.19e-09	-
200	622	191.7	-	19	-	7.89e-09	-
200	993	237.4	-	16	-	2.63e-09	-
250	652	492.7	-	18	-	8.09e-09	-
250	973	528.4	-	17	-	1.55e-09	-
300	944	749.3	-	18	-	1.81e-09	-
300	1401	1041.9	-	18	-	8.42e-10	-
300	4568	19028.7	-	19	-	3.55e-09	-

4.5 入力問題のパラメーターを変化させたときの SDPA の各部分の実行時間の解析

これまでの複数の節において、SDPA の数値実験による評価と解析を行ってきた．これまでの数値実験の内容と結果をまとめると次のようになる．

表 4.17: 最大クリーク問題に対する SDPA の主要部分の実行時間.

	$n = 300, m = 944$		$n = 300, m = 4568$	
部分	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
(I)	16.63	2.22	513.05	2.70
(II)	70.63	9.43	17773.65	93.40
(III)	104.37	13.93	107.38	0.56
(IV)	137.85	18.34	145.95	0.77
残りの部分	419.85	54.99	488.67	2.57

1. 4.2 節 : 行列 $\boldsymbol{B} \in \mathcal{S}^m$ の計算量は $\mathcal{O}(mn^3 + m^2n^2)$ であり、SDPA の中では最も実行時間を消費する部分である．そこでいかにして可能ならば $\boldsymbol{A}_i$ の疎行列性を利用して $\boldsymbol{B}$ の計算時間を減らすかが、SDPA 全体の高速化の鍵である．そのために 3.2 節において $\boldsymbol{B}$ の計算式 $\mathcal{F}$-1、$\mathcal{F}$-2 そして $\mathcal{F}$-3 の 3 つを組み合わせた効率的な計算方法を提案した.

   (a) 密行列と疎行列を混在させた SDP : 3 つの計算式を組み合わせた場合が最も高速で、推定計算量と実際の計算時間がほぼ一致していることを示した.

   (b) 二次割当問題及びグラフ分割問題 : 単独の式で計算する場合には $\mathcal{F}$-2 を用いるのが最も速いが、$\mathcal{F}$-1 と $\mathcal{F}$-3 を組み合わせることによって高速化を達成.

   (c) 最大クリーク問題 : 非常に疎な問題であるが、$\mathcal{F}$-3 だけでなく一部 $\mathcal{F}$-2 式を使用した方が高速化することを SDPA が自動判別して高速化を達成した.

2. 4.3 節 : SDPA においては最小固有値だけを求めれば良い．通常は対称行列をハウスホルダー変換して三重対角化した行列の固有値を全て求める (SYMEIG)．しかし、SDPA では 3.3 節で示した 2 分割法 (BISECTION) で最小固有値のみを求めている．両手法ともハウスホルダー変換を用いるが、BISECTION 法の実行時間のほとんど全部がハウスホルダー変換であり、SYMEIG 法と比較して極めて高速であることを示した.

3. 4.4 節 : SDP の応用問題などを題材にして数値実験を行った．ここでの目的は、SDPA の問題を解く能力 (問題の大きさ、実行時間、反復回数等) を調べること、SDPT3 との比較実験そして各問題が持つ固有の特徴 ($\boldsymbol{A}_i$ が疎、$n \ll m$ 等) によって SDPA の実行時間を消費する部分がどう変化するかを調べた.

   (a) 密な問題 : $\boldsymbol{A}_i$ が密で $0.5 \leq n/m \leq 2.0$ では、$\boldsymbol{B}$ の計算時間が全体のほとんどを占める.

   (b) やや密な問題 : $\boldsymbol{A}_i$ の平均密度が 2% 程度でも、$\boldsymbol{B}$ の計算時間が以前として多くを占める.

   (c) 疎な問題 : $\boldsymbol{A}_i$ が疎で n と m がほぼ同じ場合では、$\boldsymbol{B}$ の計算時間の占める割合も減って大きなボトルネックは存在しなくなる．また $n \ll m$ では、$\boldsymbol{B}$ の LDL 分解が占める割合が非常に大きくなる.

4.5.1 プロファイラーを用いた主要な関数の解析

収束性や反復回数等は、解く問題の難しさ (実行可能領域の広さ、狭さ) に影響される場合が多い．例えば、制御とシステム理論分野に対する SDP を解く場合には、小さい問題でも多くの反復回数を要する．しかし、1 反復における実行時間は問題の大きさ n, m および各 $\boldsymbol{A}_i$ の非零要素の数によってほぼ決定されると言ってよい．つまり、通常は $\boldsymbol{B}$ の計算が $\mathcal{O}(mn^3 + m^2n^2)$ なのでこの部分が最大のボトルネックになるが、$\boldsymbol{A}_i$ が疎になるにつれて $\boldsymbol{B}$ の実際の計算量は減っていき、すべての f_i ($\boldsymbol{A}_i$ の上三角部分の非零要素の数) が $\mathcal{O}(1)$ ならば $\mathcal{F}$-3 を用いて、$\mathcal{O}(m^2)$ まで下げることができる (最大カット問題等)．よって、この場合は $\mathcal{O}(n^3)$ か $\mathcal{O}(m^3)$ の部分が新しいボトルネックになる．どちらになるかは n と m の大きさの比に大きく依存する．

よってこの節では n, m そして f_i を変化させときに、SDPA の各部分の実行時間がどのように変化するかを実験的に解析する．また、この節では $\boldsymbol{B}$ の計算といったような目的別に実行時間を計測するだけでなく、密行列同士の乗算といったような関数別にも実行時間を測定することにする．これには以下のような理由が存在する．

1. 最大カット問題などでは、目的別に測定するとほぼ平準化されていて、実際に実行時間を消費している部分がわかりにくい．
2. $\boldsymbol{B}$ や $\boldsymbol{b}$ の計算などは、実際は行列の乗算や逆行列の計算 ($\mathcal{O}(n^3)$) などや内積や加算の計算 ($\mathcal{O}(n^2)$) で構成されているから．

まず最初に計測関数を決定するために UNIX 環境において広く使用されているプロファイラーである **prof** や **gprof** を利用する．4.4 節で扱った問題の中から幾つか選択して、プロファイラーで実行時間の大きい関数を列挙する．

まず $\boldsymbol{A}_i$ が密な問題で調べる (表 4.18, 4.23)．なお計算量は、$\boldsymbol{A}_i$ が全て密な場合を仮定していて、疎な $\boldsymbol{A}_i$ が存在する問題の場合には * がついている関数は実際の計算量よりは少なくなる．ここで各関数の説明を行うと、乗算 (密 $\times$ 密) とは、密行列構造 (二次元配列) $\boldsymbol{X}, \boldsymbol{Y} \in \mathcal{S}$ の積 $\boldsymbol{XY}$ に要する時間を表している．同様に乗算 (密 ^{T} $\times$ 密) は $\boldsymbol{X}^T\boldsymbol{Y}$、乗算 (疎 $\times$ 密) は疎行列構造の行列 $\boldsymbol{A}_i$ と 密行列構造の行列との積を表している．

ここで注意されたいのは、疎 $\times$ 密とは、疎行列構造の行列と密行列構造の行列の積を表しているので、実際には疎行列構造の行列に密行列を保持したり、その逆の場合もありうる．例えばこれらの問題では $\boldsymbol{A}_i$ に密行列を保持している．残りの行列の内積と和も同様に定義している．

また全ての実験は、10 回の反復回数で終了している．$\mathcal{F}$-k ($k = 1, 2, 3$) は、行列 $\boldsymbol{B}$ の i 番目の行をどの $\mathcal{F}$-k ($k = 1, 2, 3$) で計算したのかを示している．

既に述べたように、乱数を用いて作成した SDP の $\boldsymbol{A}_i$ の密度は 100 % であり、ノルム最小化問題はおよそ 50 % である．表 4.18 から言えることは、共に $n = m = 100$ なので * のついていない関数の実行時間は両問ともにほぼ同じである．逆に $\boldsymbol{A}_i$ の密度が 50% に減っただけでも、* のついている関数では実行時間がかなり減少している．また $\boldsymbol{B}$ の計算量も減少しているが、 $\mathcal{F}$-1 で計算する際にも疎と密の乗算や内積を使用しているので、$\boldsymbol{A}_i$ が少しでも疎になれば、高速化の効果が期待できるからである．

表 4.18: 密な問題に対する SDPA の主要部分および関数の実行時間 ($n = m = 100$)

	乱数を用いて作成した SDP		ノルム最小化問題	
部分または関数	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
$^*\boldsymbol{B}$: $\mathcal{O}(mn^3 + m^2n^2)$	121.27	91.70	62.78	87.38
$\boldsymbol{B}$ の LDL 分解 : $\mathcal{O}(m^3)$	0.07	0.05	0.02	0.02
$d\boldsymbol{X}$, $d\boldsymbol{Z}$: $\mathcal{O}(n^3)$	2.42	1.83	1.38	1.93
$^*\boldsymbol{b}$: $\mathcal{O}(n^3 + mn^2)$	1.98	1.50	1.07	1.48
固有値 : $\mathcal{O}(n^3)$	3.20	2.42	3.25	4.52
行列の乗算 (密 $\times$ 密) : $\mathcal{O}(n^3)$	13.72	10.37	13.26	18.46
行列の乗算 (密^T $\times$ 密) : $\mathcal{O}(n^3)$	1.20	0.91	1.25	1.74
* 行列の乗算 (疎 $\times$ 密) : $\mathcal{O}(n^3)$	67.20	50.81	31.98	44.51
ハウスホルダー変換 : $\mathcal{O}(n^3)$	3.20	2.39	3.22	4.48
主な $\mathcal{O}(n^3)$ 関数の合計	85.28	64.49	49.72	69.20
* 行列の内積 (疎 $\bullet$ 密) : $\mathcal{O}(n^2)$	43.30	32.74	19.50	27.14
* 行列の和 (疎 + 密) : $\mathcal{O}(n^2)$	1.87	1.41	0.87	1.21
$\mathcal{F}$-1	$1 \leq i \leq 100$		$1 \leq i \leq 99$	
$\mathcal{F}$-2	$-$		$i = 100$	

表 4.19: やや密な問題 (制御に対する SDP) に対する SDPA の主要部分および関数の実行時間

	$n = 75, m = 351$		$n = 120, m = 861$	
部分または関数	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
$^*\boldsymbol{B}$: $\mathcal{O}(mn^3 + m^2n^2)$	28.22	87.95	219.60	87.10
$\boldsymbol{B}$ の LDL 分解 : $\mathcal{O}(m^3)$	1.72	5.35	25.12	9.96
$d\boldsymbol{X}$, $d\boldsymbol{Z}$: $\mathcal{O}(n^3)$	0.57	1.77	1.48	0.59
$^*\boldsymbol{b}$: $\mathcal{O}(n^3 + mn^2)$	0.40	1.25	1.25	0.50
固有値 : $\mathcal{O}(n^3)$	0.37	1.14	1.73	0.69
行列の乗算 (密 $\times$ 密) : $\mathcal{O}(n^3)$	5.90	18.39	50.22	19.92
行列の乗算 (密^T $\times$ 密) : $\mathcal{O}(n^3)$	0.17	0.52	0.58	0.23
* 行列の乗算 (疎 $\times$ 密) : $\mathcal{O}(n^3)$	4.07	12.68	18.85	7.48
ハウスホルダー変換 : $\mathcal{O}(n^3)$	0.37	1.14	1.733	0.69
主な $\mathcal{O}(n^3)$ 関数の合計	10.50	32.73	71.38	28.31
* 行列の内積 (疎 $\bullet$ 密) : $\mathcal{O}(n^2)$	12.57	39.17	109.05	43.25
* 行列の和 (疎 + 密) : $\mathcal{O}(n^2)$	0.25	0.78	0.82	0.32
$\mathcal{F}$-1	$1 \leq i \leq 345$		$1 \leq i \leq 852$	
$\mathcal{F}$-2	$346 \leq i \leq 350$		$853 \leq i \leq 860$	
$\mathcal{F}$-3	$i = 351$		$i = 861$	

表 4.20: 疎な問題に対する SDPA の主要部分および関数の実行時間 ($n = 100$)

	最大カット問題		グラフ分割問題	
部分または関数	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
${}^*\boldsymbol{B}$: $\mathcal{O}(mn^3 + m^2n^2)$	0.03	0.47	0.83	10.08
$\boldsymbol{B}$ の LDL 分解 : $\mathcal{O}(m^3)$	0.07	0.93	0.02	0.20
$d\boldsymbol{X}$, $d\boldsymbol{Z}$: $\mathcal{O}(n^3)$	0.53	7.46	0.53	6.45
${}^*\boldsymbol{b}$: $\mathcal{O}(n^3 + mn^2)$	0.27	3.73	0.30	3.63
固有値 : $\mathcal{O}(n^3)$	3.08	43.12	3.15	38.10
行列の乗算 (密 $\times$ 密) : $\mathcal{O}(n^3)$	1.62	22.61	1.93	23.39
行列の乗算 (密^T $\times$ 密) : $\mathcal{O}(n^3)$	1.13	15.85	1.17	14.11
* 行列の乗算 (疎 $\times$ 密) : $\mathcal{O}(n^3)$	0.00	0.00	0.67	8.06
ハウスホルダー変換 : $\mathcal{O}(n^3)$	3.00	41.96	3.10	37.50
主な $\mathcal{O}(n^3)$ 関数の合計	5.75	80.42	6.87	83.06
* 行列の内積 (疎 $\bullet$ 密) : $\mathcal{O}(n^2)$	0.00	0.00	0.12	1.41
* 行列の和 (疎 + 密) : $\mathcal{O}(n^2)$	0.03	0.47	0.02	0.20
$\mathcal{F}$-1	$-$		$i = 1$	
$\mathcal{F}$-3	$1 \leq i \leq 100$		$2 \leq i \leq 101$	

表 4.21: 疎な問題に対する SDPA の主要部分および関数の実行時間 (最大クリーク問題)

	$n = 100, m = 539$		$n = 100, m = 1530$	
部分または関数	実行時間 (秒)	比率 (%)	実行時間 (秒)	比率 (%)
${}^*\boldsymbol{B}$: $\mathcal{O}(mn^3 + m^2n^2)$	2.87	17.06	24.05	7.41
$\boldsymbol{B}$ の LDL 分解 : $\mathcal{O}(m^3)$	6.17	36.71	289.55	89.23
$d\boldsymbol{X}$, $d\boldsymbol{Z}$: $\mathcal{O}(n^3)$	0.75	4.46	1.07	0.33
${}^*\boldsymbol{b}$: $\mathcal{O}(n^3 + mn^2)$	0.45	2.68	0.70	0.22
固有値 : $\mathcal{O}(n^3)$	3.25	19.35	3.20	0.99
行列の乗算 (密 $\times$ 密) : $\mathcal{O}(n^3)$	1.67	9.92	1.80	0.55
行列の乗算 (密^T $\times$ 密) : $\mathcal{O}(n^3)$	1.10	6.55	1.12	0.34
* 行列の乗算 (疎 $\times$ 密) : $\mathcal{O}(n^3)$	0.03	0.20	0.00	0.00
ハウスホルダー変換 : $\mathcal{O}(n^3)$	3.23	19.25	3.17	0.98
主な $\mathcal{O}(n^3)$ 関数の合計	6.03	35.91	6.08	1.87
* 行列の内積 (疎 $\bullet$ 密) : $\mathcal{O}(n^2)$	0.13	0.79	0.48	0.15
* 行列の和 (疎 + 密) : $\mathcal{O}(n^2)$	0.08	0.50	0.28	0.09
$\mathcal{F}$-1	$-$		$i = 1$	
$\mathcal{F}$-2	$i = 1$		$-$	
$\mathcal{F}$-3	$2 \leq i \leq 539$		$2 \leq i \leq 1530$	

また表 4.18 と表 4.19 を見ると共に $\mathcal{O}(n^2)$ の内積の占める割合が大きくなっている．しかしこの理由は異なっていて、表 4.23 と表 4.24 を比較すると、前者は $\boldsymbol{A}_i$ が密なので内積を計算する回数が少なくても 1 回の計算に要する時間が長くなっており、後者は $\boldsymbol{A}_i$ の平均密度は 2% ぐらいなので 1 回の内積の計算にかかる時間は小さいが、m が大きいので内積を計算しなければならない回数が非常に多いからである．表 4.20 の最大カット問題は、全ての $\boldsymbol{A}_i$ の非零要素が 1 個という非常に疎な問題であるので、大きなボトルネックも無く非常に理想的である．また全て $\mathcal{F}$-3 を使用していて、疎と密の内積を 1 回も使用していないことが表 4.25 から読み取ることができる．

表 4.23, 4.25, 4.26 では、n の値が全て 100 なので、密行列同士の乗算 1 回あたりの実行時間はほぼ同じである．しかし * のついた疎行列性を利用できる関数の実行時間にははっきりと差が生じている．以下に実験結果についてまとめる．場合分けをして整理すると表 4.22 のようになる．

表 4.22: n, m およびデータの密度とボトルネックとなる部分の関係

	$\boldsymbol{A}_i$ の密度	
n, m	$\boldsymbol{A}_i$ が密	$\boldsymbol{A}_i$ が疎
$n \approx m$	$\boldsymbol{B}$ の計算：密構造の $\mathcal{O}(n^3)$、密と疎の内積	大きなボトルネックなし
$n \ll m$	$\boldsymbol{B}$ の計算：密と疎の内積	LDL 分解：$\mathcal{O}(m^3)$

1. 密行列構造同士の演算は $\boldsymbol{A}_i$ の密度に依存することなく、n が同じならば 1 回あたりの実行時間はほぼ一定である．しかし $\boldsymbol{A}_i$ の密度によって $\mathcal{F}$-k の組合せが変化するので、演算の回数は依存している．つまり、$\boldsymbol{A}_i$ が密だと、$\mathcal{F}$-1 式を多く使用することになりこれらの演算の回数が多くなり、結果として全体の計算時間に占める割合が大きくなる．

2. 問題によっては、$\mathcal{O}(n^2)$ の内積の計算時間が大きな割合を占めてくる．$\boldsymbol{B}$ の計算には、$\mathcal{F}$-1 式を使用すると仮定すると $\mathcal{O}(m^2)$ 回の内積の計算が必要である．よって一般的には、m の値が大きいと内積の計算の占める割合が高くなるが、表 4.21 からも読み取れるように、m が大きくても $\boldsymbol{A}_i$ が疎で $\mathcal{F}$-1 式を使用しなければ割合は小さくなる．また、この場合には LDL 分解 $\mathcal{O}(m^3)$ の占める割合が非常に大きくなる．

表 4.23: 密な問題に対する SDPA の関数の実行回数と一回あたりの実行時間 ($n = m = 100$ また ms はミリセカンド (10^{-3}秒 を表す))

	乱数を用いて作成した SDP		ノルム最小化問題	
関数	実行回数	実行時間 (ms/回)	実行回数	実行時間 (ms/回)
行列の乗算 (密 × 密)	1139	11.75	1135	11.78
行列の乗算 (密^T × 密)	21	55.76	21	55.90
* 行列の乗算 (疎 × 密)	1000	67.22	990	32.42
* 行列の内積 (疎 • 密)	52725	0.82	52721	0.36
* 行列の和 (疎 + 密)	2000	0.93	2000	0.42

表 4.24: やや密な問題 (制御に対する SDP) に対する SDPA の関数の実行回数と一回あたりの実行時間 (ms はミリセカンド (10^{-3}秒 を表す))

	$n = 75, m = 351$		$n = 120, m = 861$	
関数	実行回数	実行時間 (ms/回)	実行回数	実行時間 (ms/回)
行列の乗算 (密 × 密)	3752	1.46	8822	5.62
行列の乗算 (密^T × 密)	40	4.27	40	15.36
* 行列の乗算 (疎 × 密)	3500	1.13	8600	2.19
* 行列の内積 (疎 • 密)	633016	0.01	3748366	0.02
* 行列の和 (疎 + 密)	14040	0.01	34440	0.02

表 4.25: 疎な問題に対する SDPA の関数の実行回数と一回あたりの実行時間 ($n = 100$ また ms はミリセカンド (10^{-3}秒 を表す))

	最大カット問題		グラフ分割問題	
関数	実行回数	実行時間 (ms/回)	実行回数	実行時間 (ms/回)
行列の乗算 (密 × 密)	137	11.88	161	12.10
行列の乗算 (密^T × 密)	20	56.40	20	55.91
* 行列の乗算 (疎 × 密)	0	0.00	10	66.80
* 行列の内積 (疎 • 密)	2225	0.00	3243	0.01
* 行列の和 (疎 + 密)	2000	0.00	2020	0.01

表 4.26: 疎な問題に対する SDPA の関数の実行回数と一回あたりの実行時間 (最大クリーク問題、また ms はミリセカンド (10^{-3}秒 を表す))

	$n = 100, m = 539$		$n = 100, m = 1530$	
関数	実行回数	実行時間 (ms/回)	実行回数	実行時間 (ms/回)
行列の乗算 (密 $\times$ 密)	141	11.85	151	12.44
行列の乗算 (密^T $\times$ 密)	20	56.05	20	56.25
* 行列の乗算 (疎 $\times$ 密)	10	1.86	10	1.95
* 行列の内積 (疎 $\bullet$ 密)	11879	0.00	48981	0.00
* 行列の和 (疎 $+$ 密)	10780	0.00	30600	0.00

4.5.2 入力行列の非零要素の密度を変化させたときの計算効率の検証

行列 $\boldsymbol{B}$ の計算が多くの場合において (特に $\boldsymbol{A}_i$ が密の場合)、大きなボトルネックとなることは既に述べた．$\boldsymbol{A}_i$ が密で、$n \ll m$ の場合にはミクロ的に見れば内積計算の占める割合が大きくなるものの、マクロ的にみれば $\boldsymbol{B}$ の計算が最も時間がかかることに変化はない．$\boldsymbol{A}_i$ が疎ならば、$\boldsymbol{B}$ の計算時間も少なくなるのだが、例えば $\boldsymbol{A}_i$ の平均密度が何 % 以下ならば、疎と判定してよいかということは漠然としかわかっていない．SDPA は 4.2 節で検証したように、その判定部分を自動化することに成功している．この節では 3.2 節で導入した 3 つの計算式を用いて、行列 $\boldsymbol{A}_i$ の密度と $\mathcal{F}$-1, $\mathcal{F}$-2 および $\mathcal{F}$-3 の計算式の切り替えの時期ついて数値実験を通して調査を行う。この数値実験において $n = m = 100$ に固定する。

図 4.3 は、行列 $\boldsymbol{A}_i$ の密度と行列 $\boldsymbol{B}$ の計算時間が全体の SDP の計算時間に対して占める割合との関係、図 4.4 は行列 $\boldsymbol{B}$ の計算時間との関係をそれぞれ示している．図 4.3 と図 4.4 から次のことがいえる．

1. 既に述べたように $\mathcal{F}$-1 式の計算にも疎行列を考慮した部分があるので、$\boldsymbol{A}_i$ が疎になるにつれて高速になっていく、しかし平均密度が 1% ぐらいになると密行列同士の乗算などの $\mathcal{O}(n^3)$ の部分が大きく効いてきてその後はほとんど減らない．

2. 図 4.4 は行列 $\boldsymbol{B}$ の計算時間で表しているので、この図から $\boldsymbol{A}_i$ が密になると $\mathcal{F}$-3 式の計算時間は $\mathcal{F}$-2 よりも、$\mathcal{F}$-2 は $\mathcal{F}$-1 よりもはるかに長いことが言える．

3. 行列 $\boldsymbol{A}_i$ が密な場合には、$\mathcal{F}$-1 式で行列 $\boldsymbol{B}$ を計算するのが効率的であるが、$1\% \sim 100\%$ の区間は $\mathcal{F}_1$ の優位性が保たれる。逆に言えば、現在のデータ構造、計算方法を使用する限りにおいては、1% でもまだ密な問題であると言うことができる。

4. $\mathcal{F}$-3 式は問題が疎な場合には非常に高速である。

5. $\mathcal{F}$-$*$ 式は、$\mathcal{F}$-1, 2, 3 式を組み合せることによって $\mathcal{F}$-$* \leq \min\{\mathcal{F}\text{-}1, \mathcal{F}\text{-}2, \mathcal{F}\text{-}3\}$ を達成している。

また、図 4.5, 4.6, 4.7, 4.8, 4.9, 4.10 は、入力問題のパラメータ $(n, m, \boldsymbol{A}_i)$ と SDPA の主要な部分や関数の実行時間の比率を表している．これらの実験結果からも、表 4.22 で示した部分がそれぞれの場合においてボトルネックとなっていることがわかる．

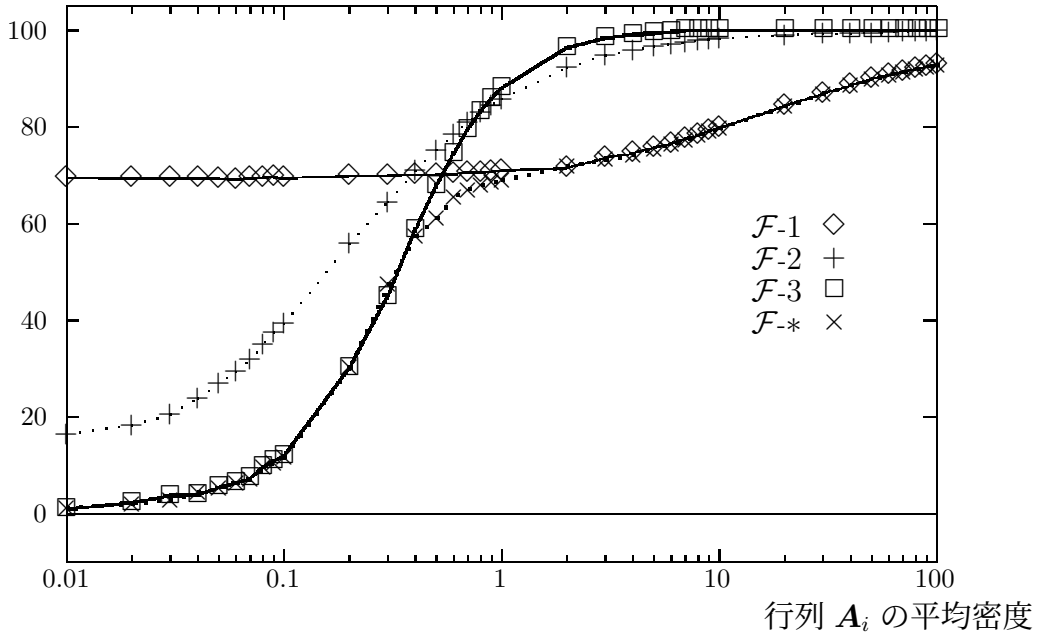

図 4.3: 入力行列の密度と各計算式が全体の計算時間に占める比率との関係

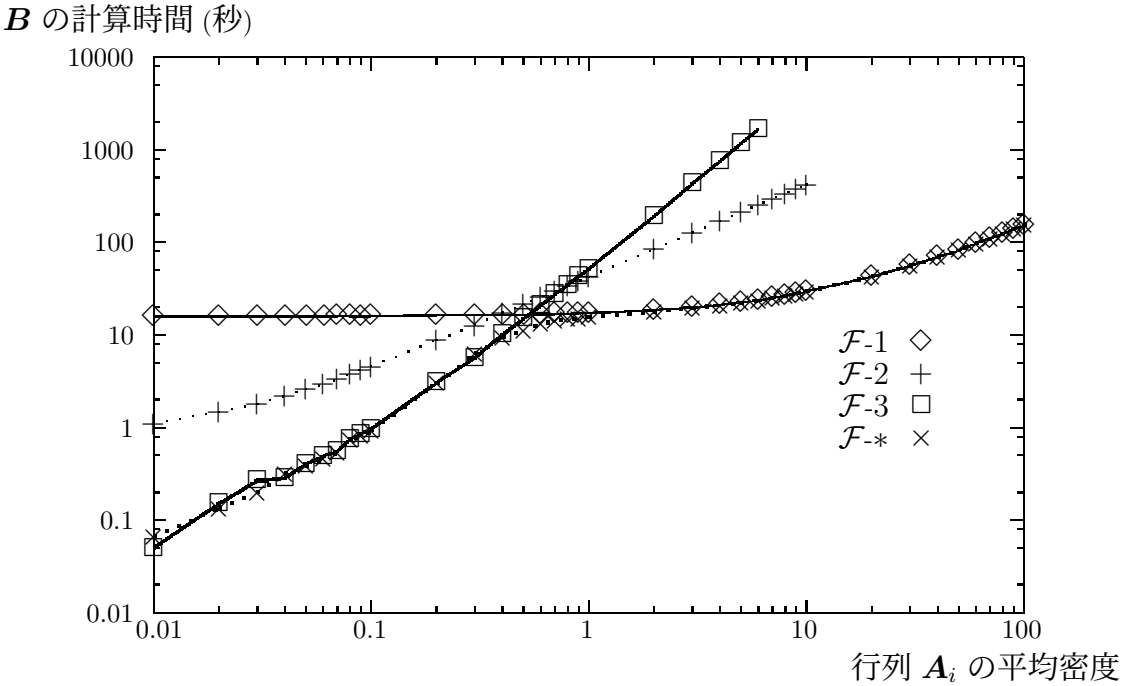

図 4.4: 入力行列の密度と各計算式の計算時間との関係

SDPA の主要部分の実行時間の比率 (%)

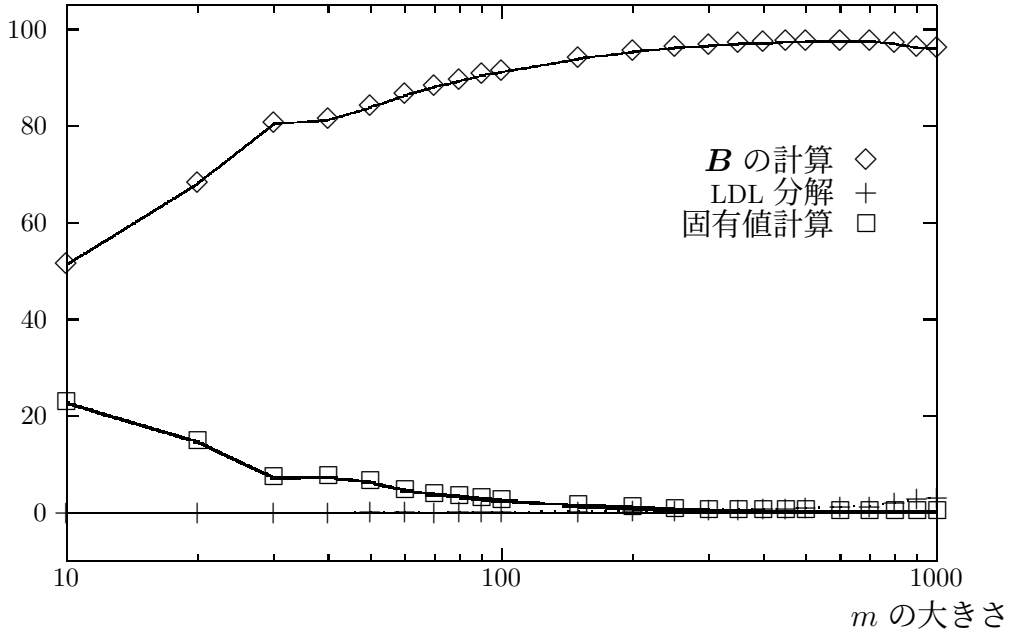

図 4.5: 入力問題と SDPA の主要部分の実行時間の比率との関係 ($n = 64$, 行列の平均密度 $= 100\%$)

SDPA の主要関数の実行時間の比率 (%)

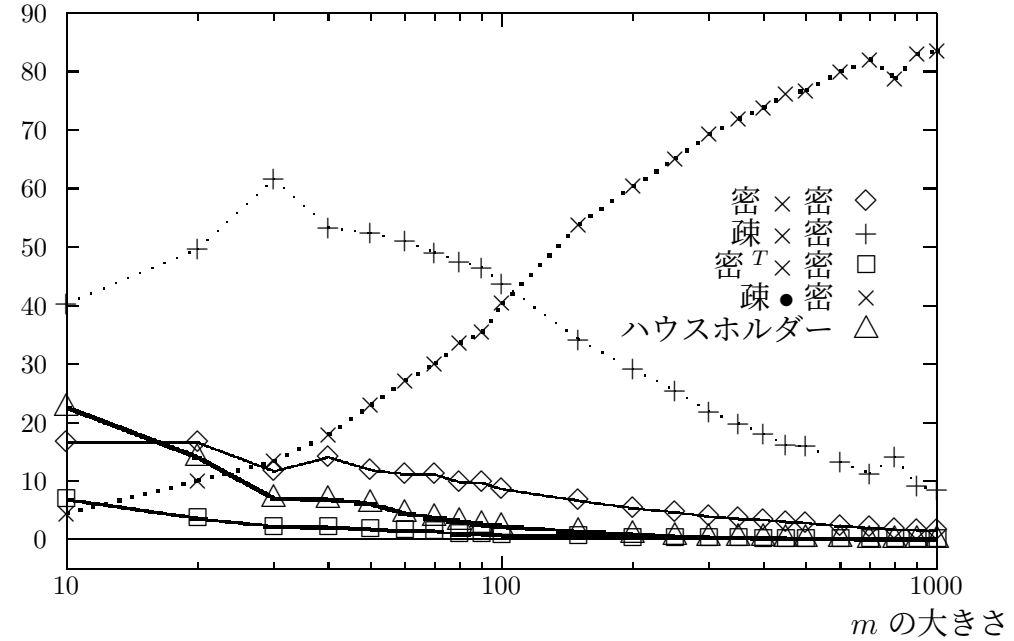

図 4.6: 入力問題と SDPA の主要関数の実行時間の比率との関係 ($n = 64$, 行列の平均密度 $= 100\%$)

SDPA の主要部分の実行時間の比率 (%)

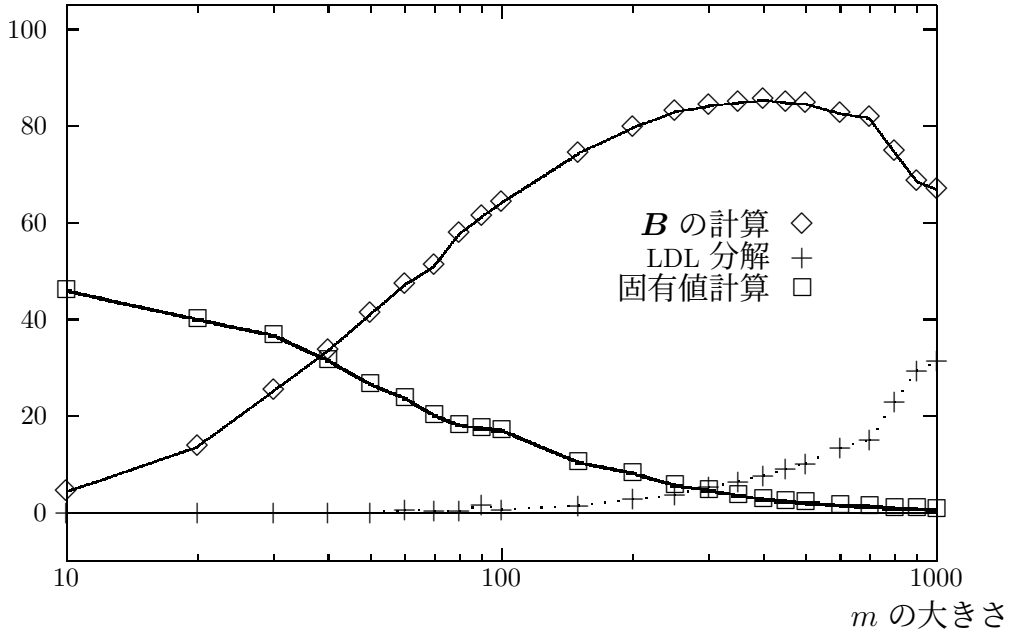

図 4.7: 入力問題と SDPA の主要部分の実行時間の比率との関係 ($n = 64$, 行列の平均密度 $= 1\%$)

SDPA の主要関数の実行時間の比率 (%)

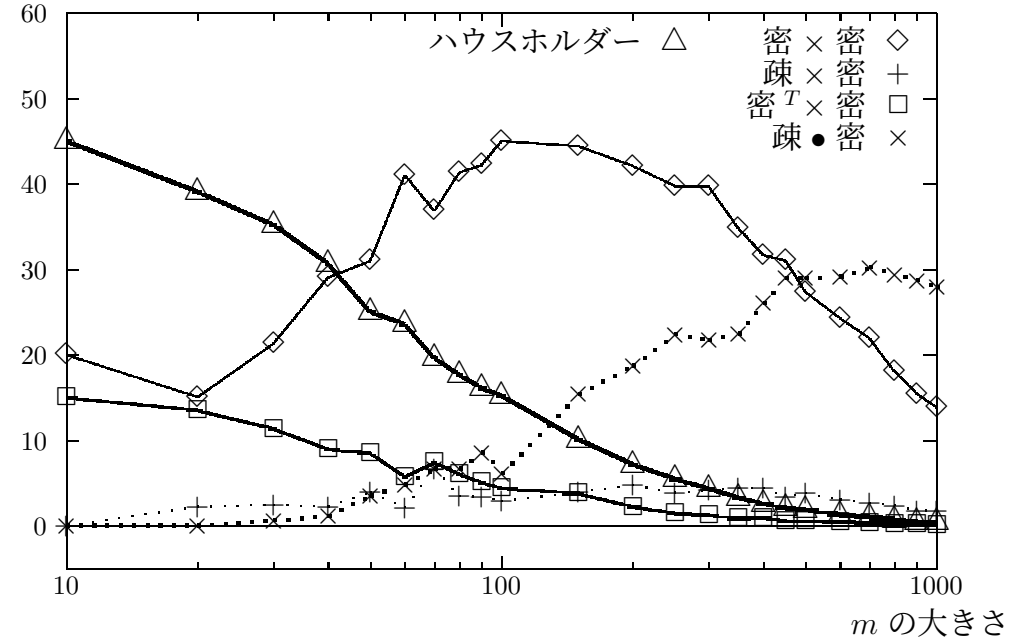

図 4.8: 入力問題と SDPA の主要関数の実行時間の比率との関係 ($n = 64$, 行列の平均密度 $= 1\%$)

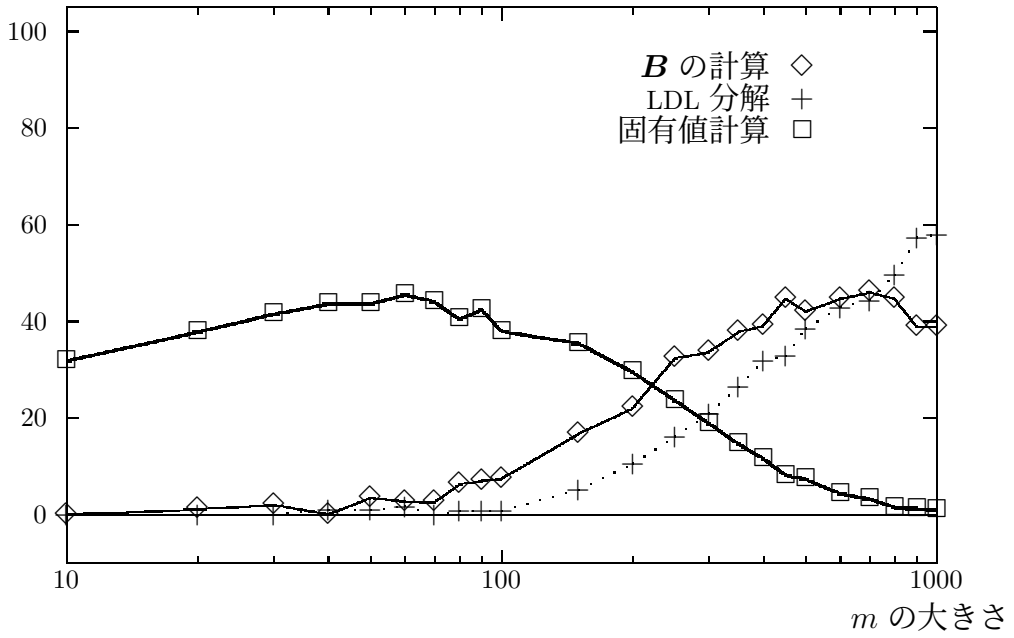

図 4.9: 入力問題と SDPA の主要部分の実行時間の比率との関係 ($n = 64$, 行列の平均密度 $= 0.1\%$)

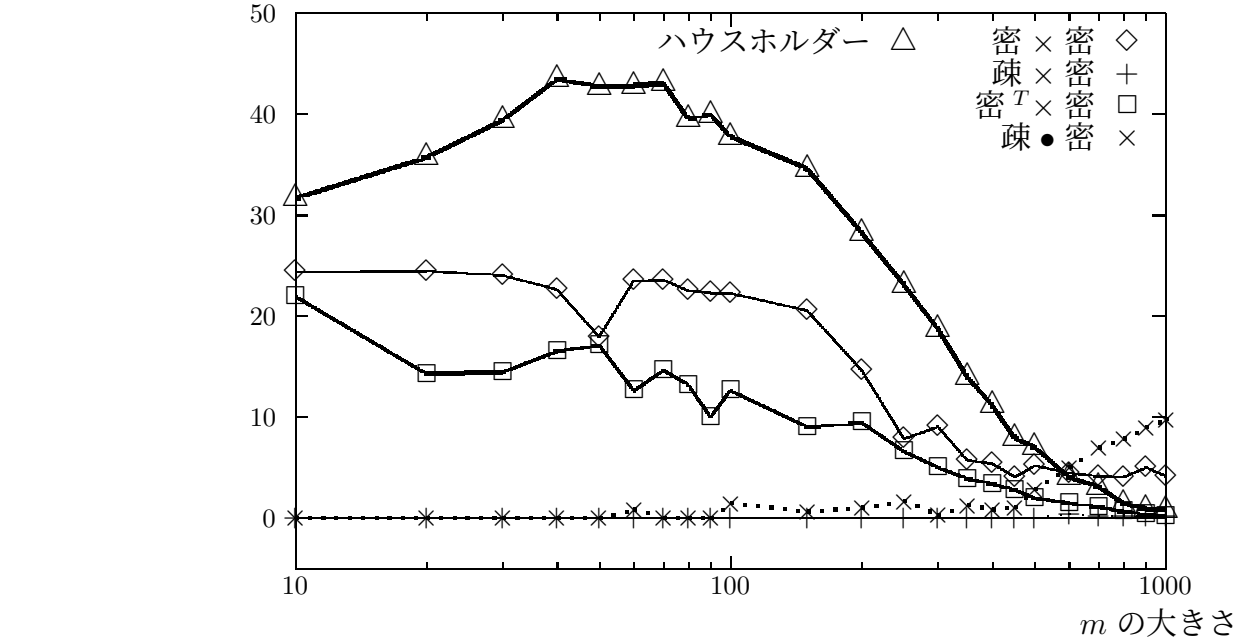

図 4.10: 入力問題と SDPA の主要関数の実行時間の比率との関係 ($n = 64$, 行列の平均密度 $= 0.1\%$)

第 5章

SDPA のライブラリ

この章においては、SDPA のライブラリ (Callable Library) ついて解説する．SDPA は、付録 A で説明したようにデータファイルを指定して問題を解く方式を取っているので、例えば別の C のプログラムから繰り返して呼ぶことができない (UNIX のシステムコールを用いれば繰り返して呼ぶことはできる)．SDPA のライブラリは、次のような方針で開発を行った．

1. 最新の SDPA (1997 年 10 月現在 Ver 4.0) を元にして作成し、SDPA とまったく同じ機能、アルゴリズムを持つようにする．
2. 自動的に確保したメモリは１回の実行を終えたら、自動的に解放するか、使用者が個別に解放できるようにする．
3. 次回に入力する問題の構造が同じならば、つまり同じ mDIM, nBLOCK, bLOCKsTRUCT (付録 A を参照) を持つならば、前回確保したオブジェクトがそのまま使用できるようにする．

そこで、SDPA のライブラリの簡単な使用方法について説明を行う．SDPA のライブラリを用いるには次の SDPA クラスを用いる必要がある (図 5.1)．この SDPA クラスは sdpa_lib.hpp のファイルに含まれる．次に SDPA のライブラリの実行に関係するファイルを示す．

SDPA のライブラリの実行に関係するファイル

1. ヘッダファイ６つ： sdpa−algebra.hpp, sdpa−subproc.hpp, sdpa−matutl.hpp, sdpa−io_tool.hpp, sdpa−problem.hpp, sdpa_lib.hpp
2. SDPA のライブラリ：sdpa.a
3. Meschach [48] のライブラリ：meschach.a
4. サンプルプログラム：test.cpp

```
class SDPA
{
public:
        char*           InputFileName;   // 入力ファイル名へのポインタ
        char*           OutputFileName;  // 出力ファイル名へのポインタ
        char*           OutputFileName2; // デバッグ用のファイルへのポインタ
        char*           ParameterFileName;  // パラメータファイルへのポインタ
        char*           InitialPtFileName;  // 初期点 (解) ファイルへのポインタ
        which_method    Direction;  // 探索方向の指定
        boolean         CheckSymmetric;  // 入力ファイルの対称性の判定
        boolean         AllDelete;  // 実行時に生成したオブジェクトの自動消去

        int mDIM; // 主問題の等式制約数
        int nBLOCK;  // ブロック数
        int* bLOCKsTRUCT; // ブロック対角行列の構造ベクトル

        int** NonzeroElement; // 入力行列の非零要素の数を保持

        SparseBlockMatrix* sfMAT; // SparseBlockMatrix クラスへのポインタ

// ここから下は、SDPA の実行時に自動的に生成されるオブジェクトへのポインタ
        matrix* CountSparse;
        parameterClass* pARAM;
        newtonClass* newton;
        p_d_Triplet* iNITPT;
        p_d_Triplet* currentPt;
        stepLength* alpha;
        residuals* iNITRES;
        residuals* currentRes;
public:
        SDPA(); // SDPA クラスのコンストラクタ
};
```

図 5.1: sdpa_lib.hpp に記述されている SDPA クラス

次に、サンプルプログラム test.cpp を使用として SDPA のライブラリの使用法の一つを解説する．図 5.2 は、使用法の一例だが順を追って説明していく．まず

```
g++ -O3 -o test test.cpp sdpa.a meschach.a -lm
```

とコンパイルする．この例においては example1.dat の問題を HRVW/KSH/M 方向で、また example2.dat の問題を AHO 方向を用いて解くとする．出力結果ファイルをそれぞれ example1.out, example2.out とする．

始めに、main 関数から見ていくことにする．

```
SDPA* SDPAproblem;
SDPAproblem = new SDPA[2];
```

複数の問題を解くのであっても SDPA クラスへのオブジェクトは兼用しれば１つでも構わない．しかし、上のように問題ごとに複数持つことも可能である．一つでよければ単に

```
SDPA*   SDPAproblem = new SDPA;
```

とすればよい．

```
SDPAproblem[0].InputFileName = ''example1.dat''  //  入力ファイル名
SDPAproblem[0].OutputFileName = ''example1.out'' //  出力ファイル名
SDPAproblem[0].OutputFileName2 = ''out1''        //  デバッグ用ファイル
名

SDPAproblem[0].Direction = KSH;    //  探索方向の指定
SDPAproblem[0].AllDelete = FALSE; //  オブジェクトは手動消去
```

OutputFileName2 はデバッグ用の出力ファイルなので通常は使用しない．Direction は KSH (HRVW/KSH/M 方向), NT (NT 方向), AHO (AHO 方向) を使用することができる．AllDelete は TRUE にすると SDPA が自動的に動的に確保したオブジェクトを解放する．FALSE にすると 本例のように関数 DeleteProblem(SDPA& SDPAproblem) を作成して手動で解放する必要がある．SDPAproblem[1] の場合も同様に作業を行っている．

```
SDPAmain(SDPAproblem[0]);
DeleteProblem(SDPAproblem[0]);
SDPAmain(SDPAproblem[1]);
DeleteProblem(SDPAproblem[1]);
```

SDPAmain 関数は、SDPA クラスへのオブジェクトの参照型を引数とする．DeleteProblem 関数も同じ引数を持つ．DeleteProblem 関数では、全てのオブジェクトの解放を行っている．再利用したいオブジェクトがあれば、ここで解放しないようにすることによって再利用が可能となる．

```
#include "sdpa_lib.hpp"

boolean DeleteProblem(SDPA& SDPAproblem)
{
        delete [] SDPAproblem.bLOCKsTRUCT;

        for (int i = 0; i < SDPAproblem.mDIM + 1; i++)
                delete [] SDPAproblem.NonzeroElement[i];
        delete [] SDPAproblem.NonzeroElement;

        (*SDPAproblem.CountSparse).matrixDelete();
        (*SDPAproblem.newton).newtonClassDelete();
        (*SDPAproblem.iNITPT).p_d_TripletDelete();
        (*SDPAproblem.currentPt).p_d_TripletDelete();
        (*SDPAproblem.alpha).stepLengthDelete();
        (*SDPAproblem.iNITRES).residualsDelete();
        (*SDPAproblem.currentRes).residualsDelete();

        delete [] (SDPAproblem.sfMAT);

        return TRUE;
};

int main(int argc, char *argv[])
{
        SDPA* SDPAproblem;
        SDPAproblem = new SDPA[2];

        SDPAproblem[0].InputFileName = ''example1.dat''
        SDPAproblem[0].OutputFileName = ''example1.out''
        SDPAproblem[0].OutputFileName2 = ''out1''

        SDPAproblem[0].Direction = KSH;
        SDPAproblem[0].AllDelete = FALSE;

        SDPAproblem[1].InputFileName = ''example2.dat''
        SDPAproblem[1].OutputFileName = ''example2.out''
        SDPAproblem[1].OutputFileName2 = ''out2''

        SDPAproblem[1].Direction = AHO;
        SDPAproblem[1].AllDelete = FALSE;

        SDPAmain(SDPAproblem[0]);
        DeleteProblem(SDPAproblem[0]);
        SDPAmain(SDPAproblem[1]);
        DeleteProblem(SDPAproblem[1]);

        exit(0);
};
```

図 5.2: SDPA のライブラリの使用法の一例

第 6章

結論と今後の課題

SDPA は、C++ 言語を用いて初めて作成された SDP に対する主双対内点法のソフトウェアである．他の SDP ソフトウェアとは基本となるアルゴリズムはほとんど同じなので、大きな差異は、使用言語そして実際のアルゴリズムの実装方法ということになる．本論文の結論をまとめると以下のようになる．

1. SDPA は、動的なメモリ確保及び解放を行っているので、メモリなどのハードウェア資源さえ整っていれば、大規模な問題を解くことができる．
2. SDPA は、疎行列 $\boldsymbol{A}_i$ を効率良く扱うためのデータ構造、計算方法を備えている．通常、行列 $\boldsymbol{B}$ の計算が最大のボトルネックとなるが、入力に疎行列を含む場合にはそのボトルネックを解消している (現時点ではこのような工夫は他の SDP のソフトウェアに含まれていない)．
3. SDPA は、特定の SDP を対象としているのではなく汎用のソフトウェアである．よってどのような $\boldsymbol{A}_i$ が入力されても、可能ならば高速化することができる．
4. さらなる高速化を目指す際の指針になるように、$n, m, \boldsymbol{A}_i$ の密度などによってボトルネックになる可能性のある部分を解析した．

また、今後の課題には次のようなものが挙げられる．

1. プログラムの簡素化、数値的安定性、データ構造の改良等を行う．
2. 組合せ最適化問題等では、繰り返し SDP を解く必要があるので SDPA のライブラリの改良等を行う．
3. 単一 CPU のコンピュータでは、さらなる大幅な高速化は困難と思われるので、並列化も指向する．

SDPA をインターネット上からフリーソフトとして配布していることは既に述べたが、実際にはユーザーから寄せられた多くの質問、意見、要望等が新しいアルゴリズムの開発やプログラムの改良などに大変役立っていることは事実である．SDPA に関する質問や意見

等はどんなことでもよいので、こちらの方まで寄せていただきたいと考えている．現時点では、SDPA に匹敵するようなソフトウェアはないが、これからもさらに多くの改良を加えていく予定である．

Appendix A

SDPA の使用方法

この章においては、SDPA の最新版 (Version 4.02 :1997 年 12 月現在) の使用方法について説明を行う．なお細部については SDPA のマニュアル [10] を参照されたい。さらに SDPA のライブラリ (Callable Library) については、5 章で触れる．

A.1 SDPA における SDP の等式標準形

SDPA は以下のような等式標準形 (A.1) を解く、2 章で解説した等式標準形 (2.1) とはやや異なっているが、本質的に等価であり容易に (2.1) から (A.1) に変換可能である．

$$\left.\begin{array}{lll} \mathcal{P}: & \text{min.} & \displaystyle\sum_{i=1}^{m} c_i x_i \\ & \text{subject to} & \boldsymbol{X} = \displaystyle\sum_{i=1}^{m} \boldsymbol{F}_i x_i - \boldsymbol{F}_0 \\ & & \boldsymbol{X} \succeq \boldsymbol{O}.\ \ \boldsymbol{X} \in \mathcal{S}. \\ \mathcal{D}: & \text{max.} & \boldsymbol{F}_0 \bullet \boldsymbol{Y} \\ & \text{subject to} & \boldsymbol{F}_i \bullet \boldsymbol{Y} = c_i \ (i = 1, 2, \ldots, m) \\ & & \boldsymbol{Y} \succeq \boldsymbol{O}.\ \ \boldsymbol{Y} \in \mathcal{S}. \end{array}\right\}. \tag{A.1}$$

2.1 から A.1 への変換方法

$$\begin{aligned} -\boldsymbol{A}_i (i = 0, \ldots, m) &\longrightarrow \boldsymbol{F}_i (i = 0, \ldots, m) \\ -a_i (i = 1, \ldots, m) &\longrightarrow c_i (i = 1, \ldots, m) \\ \boldsymbol{X} &\longrightarrow \boldsymbol{Y} \\ \boldsymbol{y} &\longrightarrow \boldsymbol{x} \\ \boldsymbol{Z} &\longrightarrow \boldsymbol{X} \end{aligned}$$

また 2 章で述べたように以下の仮定を置く．

条件 1.1. $\{\boldsymbol{F}_i \ : \ i = 1, 2, \ldots, m\} \subset \mathcal{S}$ は線形独立である．

もし、SDPA に入力する SDP が上記の仮定を満たしていない場合には探索方向を求める部分などで数値的に不安定になる場合がある.

A.1.1 例題１ (example1.dat)

まず始めに、等式標準形 A.1 に即した簡単な例題を紹介する.

$$\left.\begin{array}{lll} \mathcal{P} & \text{Minimize} & 48x_1 - 8x_2 + 20x_3 \\ & \text{subject to} & \boldsymbol{X} = \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix} x_1 + \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix} x_2 + \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix} x_3 - \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix} \\ & & \boldsymbol{X} \succeq \boldsymbol{O}. \\ \mathcal{D} & \text{Maximize} & \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix} \bullet \boldsymbol{Y} \\ & \text{subject to} & \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix} \bullet \boldsymbol{Y} = 48, \ \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix} \bullet \boldsymbol{Y} = -8 \\ & & \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix} \bullet \boldsymbol{Y} = 20, \ \boldsymbol{Y} \succeq \boldsymbol{O}. \end{array}\right\}$$

この問題では、

$$\begin{aligned} m &= 3, \ n = 2, \ \boldsymbol{c} = \begin{pmatrix} 48 \\ -8 \\ 20 \end{pmatrix}, \ \boldsymbol{F}_0 = \begin{pmatrix} -11 & 0 \\ 0 & 23 \end{pmatrix}, \\ \boldsymbol{F}_1 &= \begin{pmatrix} 10 & 4 \\ 4 & 0 \end{pmatrix}, \ \boldsymbol{F}_2 = \begin{pmatrix} 0 & 0 \\ 0 & -8 \end{pmatrix}, \ \boldsymbol{F}_3 = \begin{pmatrix} 0 & -8 \\ -8 & -2 \end{pmatrix}. \end{aligned}$$

A.1.2 例題２ (example2.dat)

SDPA にでは、ブロック対角な入力データ行列を扱えるのが大きな特徴の一つである. この例題では、一つの定数行列 $\boldsymbol{F}_i (i = 0, \ldots, m)$ が３つのブロックを持っている.

$$\begin{aligned} m &= 5, \ n = 7, \ \boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} = \begin{pmatrix} 1.1 \\ -10 \\ 6.6 \\ 19 \\ 4.1 \end{pmatrix}, \\ \boldsymbol{F}_0 &= \begin{pmatrix} -1.4 & -3.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -3.2 & -28 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 15 & -12 & 2.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & -12 & 16 & -3.8 & 0.0 & 0.0 \\ 0.0 & 0.0 & 2.1 & -3.8 & 15 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -4.0 \end{pmatrix}, \end{aligned}$$

$$
\boldsymbol{F}_1 = \begin{pmatrix}
0.5 & 5.2 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
5.2 & -5.3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 7.8 & -2.4 & 6.0 & 0.0 & 0.0 \\
0.0 & 0.0 & -2.4 & 4.2 & 6.5 & 0.0 & 0.0 \\
0.0 & 0.0 & 6.0 & 6.5 & 2.1 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -4.5 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -3.5
\end{pmatrix}
$$

$$
\vdots
$$

$$
\boldsymbol{F}_5 = \begin{pmatrix}
-6.5 & -5.4 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
-5.4 & -6.6 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 6.7 & -7.2 & -3.6 & 0.0 & 0.0 \\
0.0 & 0.0 & -7.2 & 7.3 & -3.0 & 0.0 & 0.0 \\
0.0 & 0.0 & -3.6 & -3.0 & -1.4 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 6.1 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.5
\end{pmatrix}.
$$

このブロック対角構造を利用して、ソフトウェア全体を高速化する方法については 3 章において解説する.

A.2 SDPA の実行に必要なファイルについて

SDPA の実行には以下のようなファイルを必要とする.

- “sdpa” — SDP を解く実行形式のファイル.
- “入力データファイル” — A.3 節で示すデータ形式に即していればどんなファイルの名前でも良い (ただし 255 文字以内). また、後述する疎行列のデータ形式を使用する場合は、ファイルの名前の末尾に “-s” を付けなくてはならない.
- “param.sdpa” — SDPA の制御や修了条件等のパラメータを集めたファイル. A.4 節で詳しく説明する.
- “出力結果ファイル’ —’ SDPA の出力結果ファイル. ファイルの中身については、A.5 節で説明する.

前出の例題 1 が example1.dat、例題 2 が example2.dat にそれぞれ対応しているとすると、これらの例題を解くには、

% **sdpa example1.dat example1.out**
% **sdpa example2.dat example2.out**
と入力する.

A.3　入力データファイル

A.3.1　“example1.dat” — 例題１の入力ファイル

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
   3  =  mDIM
   1  =  nBLOCK
   2  = bLOCKsTRUCT
{48, -8, 20}
{ {-11,  0}, { 0, 23} }
{ { 10,  4}, { 4,  0} }
{ {  0,  0}, { 0, -8} }
{ {  0, -8}, {-8, -2} }
```

A.3.2　“example2.dat” — 例題２の入力ファイル

```
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}"
   5  =  mDIM
   3  =  nBLOCK
   (2, 3, -2)   = bLOCKsTRUCT
{1.1, -10, 6.6, 19, 4.1}
{
{ { -1.4, -3.2 },
  { -3.2,-28   }   }
{ { 15,  -12,    2.1 },
  {-12,   16,   -3.8 },
  {  2.1, -3.8, 15   }   }
  {  1.8, -4.0 }
}
{
{ {  0.5,  5.2 },
  {  5.2, -5.3 }   }
{ {  7.8, -2.4,  6.0 },
  { -2.4,  4.2,  6.5 },
  {  6.0,  6.5,  2.1 }   }
  { -4.5, -3.5 }
}
```

•
•

•

```
{
{ { -6.5, -5.4 },
  { -5.4, -6.6 }   }
{ {  6.7, -7.2, -3.6 },
  { -7.2,  7.3, -3.0 },
  { -3.6, -3.0, -1.4 }   }
  {  6.1, -1.5 }
}
```

A.3.3 入力データファイルの書式

入力データファイルは次のような構造をしている.
問題名及び注釈
m — 主問題の変数の数 (双対問題の制約式の数).
nBLOCK — ブロック対角行列のブロック数
bLOCKsTRUCT — ブロック対角行列のブロック構造を示すベクトル
$\boldsymbol{c}$
$\boldsymbol{F}_0$
$\boldsymbol{F}_1$
.
.
$\boldsymbol{F}_m$

これらの構成要素に対して、以下説明を行う.

A.3.4 問題名及び注釈

入力ファイルの先頭から、単数または複数行の注釈等を付け加えることができる. ただし、次のような制約がある.

1. 各行は、$\star$ か " (スペース、タブ等は入れないこと) で始めなければならない.
2. 複数行に渡っても良いが、一行には 75 文字以内とする.
3. 一度上記の記号以外で開始される行が存在したら、以下は注釈行を入れることは出来ない.

"example1.dat" の注釈行は次の行である.

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
```

同様に "example2.dat" の注釈行も以下の通りである.

```
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}
```

これらの問題名及び注釈行は省略することも出来る.

A.3.5 主問題の変数の数

次に、1行使用して主問題の変数の数 (または双対問題の制約式の数) m を記述する．なお、m のあとの文字はこの行の最後まで無視されるので、

```
3  =  mDIM
```

のように、"= mDIM" 等の説明を入れても良い.

A.3.6 ブロック対角行列のブロック数とその構造ベクトル

SDPA は、例題 2 のようなブロック対角行列構造を持つような問題を効率良く処理することが出来て、SDPA の大きな特徴の一つになっている．このブロック対角行列は、$\boldsymbol{F}_0,\ \boldsymbol{F}_1,\ \ldots,\ \boldsymbol{F}_m$ に共通である.

次のようなブロック対角行列 $\boldsymbol{F}$ を SDPA で扱う場合には以下のように表現する.

$$\left.\begin{array}{rcl} \boldsymbol{F} & = & \begin{pmatrix} \boldsymbol{B}_1 & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{B}_2 & \boldsymbol{O} & \cdots & \boldsymbol{O} \\ \cdot & \cdot & \cdot & \cdots & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} & \cdots & \boldsymbol{B}_\ell \end{pmatrix}, \\ \boldsymbol{B}_i & : & p_i \times p_i \text{ 対称行列 } (i = 1, 2, \ldots, \ell), \end{array}\right\} \tag{A.2}$$

次に、ブロック数 nBLOCK とブロック対角行列の構造ベクトル bLOCKsTRUCT を以下のように定義する.

$$\begin{aligned} \text{nBLOCK} &= \ell, \\ \text{bLOCKsTRUCT} &= (\beta_1,\ \beta_2,\ \ldots,\ \beta_\ell), \\ \beta_i &= \begin{cases} p_i & \boldsymbol{B}_i \text{ が対称行列} \\ -p_i & \boldsymbol{B}_i \text{ が対角行列} \end{cases} \end{aligned}$$

例えば、$\boldsymbol{F}$ が次のような構造を持つ場合には、

$$\begin{pmatrix} 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 2 & 4 & 5 & 0 & 0 & 0 & 0 \\ 3 & 5 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}, \tag{A.3}$$

$$\mathrm{nBLOCK} = 3, \quad \mathrm{bLOCKsTRUCT} = \ (3,\ 2,\ -2)$$

となる．もし、次のようなブロック対角構造を持たないような通常の対称行列を扱う場合には、

$$\boldsymbol{F} = \begin{pmatrix} \star & \star & \star \\ \star & \star & \star \\ \star & \star & \star \end{pmatrix}, \ \star \text{は実数を表す.}$$

nBLOCK と bLOCKsTRUCT は以下のように表現する．

$$\mathrm{nBLOCK} = 1,\ \mathrm{bLOCKsTRUCT} = 3$$

nBLOCK と bLOCKsTRUCT はそれぞれ一行に記述する．注釈行と同様に、nBLOCK と bLOCKsTRUCT 記述後の文字は無視される．空白 (スペース) やタブに加えて次のような記号も含まれる．

, ( ) { }

例題１では次のように記述する．

```
1  =  nBLOCK
2  = bLOCKsTRUCT
```

同様に、例題２でも次のように記述する．nBLOCK の数だけ bLOCKsTRUCT の要素の数が必要であることに注意されたい．

```
3  =  nBLOCK
2    3   -2   = bLOCKsTRUCT
```

ここでも、前述のように “= nBLOCK” や “= bLOCKsTRUCT” 等は無視される．

A.3.7 定数ベクトル

次に、定数ベクトル $c_1, c_2, \ldots, c_m$ を記述する．数字と数字の区切りに、空白 (スペース) やタブに加えて次のような記号も使用出来る．

, ( ) { }

例題１では次のように記述する．

```
{48, -8, 20}
```

同様に、例題２では次のように記述する．

```
{1.1, -10, 6.6, 19, 4.1}
```

A.3.8 定数行列

最後に nBLOCK と bLOCKsTRUCT の記述に即して、定数行列 $\boldsymbol{F}_0,\ \boldsymbol{F}_1,\ \ldots,\ \boldsymbol{F}_m$ を記述する．数字と数字の区切りに、空白 (スペース) やタブに加えて次のような記号も使用出来る．

, ( ) { }

ここで、注意することは定数行列 $\boldsymbol{F}$ (A.2) を記述するときには、各ブロック $\boldsymbol{B}_1,\ \boldsymbol{B}_2,\ \ldots\ ,\boldsymbol{B}_\ell$ **のみ** を順々に記述していく．もし前述 $\boldsymbol{F}$ (A.3) (nBLOCK $= 3$, bLOCKsTRUCT $= (3, 2, -2)$) を記述するには以下のように行う．

```
{ {{1  2  3} {2  4  5} {3  5  6}}, {{1  2} {2  3}}, 4, 5 }
```

また次のように複数行で記述することも可能である．

```
{{1  2  3}
 {2  4  5}
 {3  5  6}
          {1  2}
          {2  3}
                4
                   5}
```

例題 1 の定数行列部分は以下のように記述する．

```
{ {-11,  0}, { 0, 23} }
{ { 10,  4}, { 4,  0} }
{ {  0,  0}, { 0, -8} }
{ {  0, -8}, {-8, -2} }
```

同様に、nBLOCK $= 3$, bLOCKsTRUCT $= (2, 3, -2)$ の構造を持つ例題 2 の定数行列部分は以下のように記述する．

```
{
{ { -1.4, -3.2 },
  { -3.2,-28   }   }
{ { 15,  -12,    2.1 },
  {-12,   16,   -3.8 },
  {  2.1, -3.8, 15   }   }
  {  1.8, -4.0 }
}
{
{ {  0.5,  5.2 },
  {  5.2, -5.3 }   }
```

```
{ {  7.8, -2.4,  6.0 },
  { -2.4,  4.2,  6.5 },
  {  6.0,  6.5,  2.1 }   }
  { -4.5, -3.5 }
}
```

•
•
•

```
{
{ { -6.5, -5.4 },
  { -5.4, -6.6 }   }
{ {  6.7, -7.2, -3.6 },
  { -7.2,  7.3, -3.0 },
  { -3.6, -3.0, -1.4 }   }
  {  6.1, -1.5 }
}
```

所見 データ部分を記述する際に以下ような記号を用いないで、下記のような表現も可能である．大規模なデータを記述する際にはファイル容量を節約することができ便利である．

, ( ) { }

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
3
1
2
 48  -8  20
-11   0   0  23
 10   4   4   0
  0   0   0  -8
  0  -8  -8  -2
```

A.4 パラメータファイル

パラメータファイル “param.sdpa” 以下のような要素で構成されている．

```
100 unsigned int maxIteration;
1.0E-6 double 0.0 < epsilonStar;
1.0E3 double 0.0 < lambdaStar;
2.0 double 1.0 < omegaStar;
```

```
-1.0E5 double lowerBound;
1.0E5 double upperBound;
0.05 double 0.0 <= betaStar <  1.0;
0.10 double 0.0 <= betaBar  <  1.0, betaStar <= betaBar;
0.95 double 0.0 < gammaStar  <  1.0;
```

“param.sdpa” は９個のパラメータをそれぞれ９行で記述する必要があり、一つも省略することが出来ない。また “param.sdpa” という名前にファイルしか使用することが出来ない．しかし 5 章で説明する SDPA のライブラリでは、複数のファイルを使用可能である．また、各行の始めの数字 (パラメータ値) の後の注釈文は無視される．

- maxIteration — SDPA の反復回数 (iteration) の上限値．反復回数がこの値を過ぎると、最適解に達していなくともアルゴリズムを修了する．

- epsilonStar — 解くべき SDP の近似最適解の精度．現在の反復において $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ が以下の不等式を満たしたときにアルゴリズムを修了する．

$$
\begin{aligned}
\min\{\text{epsilonStar},\ \text{1.0E-7}\} &\geq \max\left\{\left|[X^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0]_{pq}\right| \ :\ p,q = 1,2,\ldots,n\right\},\\
\min\{\text{epsilonStar},\ \text{1.0E-7}\} &\geq \max\left\{\left|\boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i\right| \ : i = 1,2,\ldots,m\right\},\\
\text{epsilonStar} &\geq \frac{|\sum_{i=1}^{m} c_i x_i^k - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|}{\max\{1.0, \frac{|\sum_{i=1}^{m} c_i x_i^k| + |\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k|}{2}\}}\\
&= \frac{\text{主問題の目的関数値} - \text{双対問題の目的関数値}}{\max\{1.0, \frac{|\text{主問題の目的関数値}| + |\text{双対問題の目的関数値}|}{2}\}},
\end{aligned}
$$

  あまり epsilonStar が小さいと数値的に不安定になる場合があるので、その場合は少し大きめの epsilonStar $\geq 1.0E-7$ を選択する．

- lambdaStar — 初期点 $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ を決定するパラメータ．

$$\boldsymbol{x}^0 = \boldsymbol{0},\ \boldsymbol{X}^0 = \text{lambdaStar}\ \times\ \boldsymbol{I},\ \boldsymbol{Y}^0 = \text{lambdaStar}\ \times\ \boldsymbol{I}.$$

  ここで $\boldsymbol{I}$ は単位行列を表す． 通常望ましい $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ は最適解 $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ の大きさ、スケーリングで決定されるので事前に知ることは難しい．もし事前に $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ の大きさ等の情報が無い場合には十分に大きいを持つ **以下のような lambdaStar を取ることを推奨する．**

$$\boldsymbol{X}^* \preceq \text{lambdaStar}\ \times\ \boldsymbol{I} \quad \text{かつ} \quad \boldsymbol{Y}^* \preceq \text{lambdaStar}\ \times\ \boldsymbol{I}.$$

- omegaStar — SDPA の最適解を探索する領域を決定するパラメータ．主問題 $\mathcal{P}$ においては、SDPA は以下の領域で最小解 $(\boldsymbol{x}^*, \boldsymbol{X}^*)$ を探索する．

$$\boldsymbol{O} \preceq \boldsymbol{X} \preceq \text{omegaStar}\ \times\ \boldsymbol{X}^0 = \text{omegaStar} \times \text{lambdaStar}\ \times\ \boldsymbol{I},$$

もし、主問題 $\mathcal{P}$ において、この領域内で最小解が発見出来ない場合にはアルゴリズムを終了する．また双対問題 $\mathcal{D}$ において, SDPA は以下の領域で最大解 $\boldsymbol{Y}$ を探索する．

$$\boldsymbol{O} \preceq \boldsymbol{Y} \preceq \text{omegaStar} \times \boldsymbol{Y}^0 = \text{omegaStar} \times \text{lambdaStar} \times \boldsymbol{I},$$

双対問題 $\mathcal{D}$ においても同様にこの領域内で最大解を発見出来ない場合にはアルゴリズムを終了する．この場合には、より大きい lambdaStar とより小さい $\underline{\text{omegaStar}} > 1$ を設定して再度問題を解くことを推奨する．

- lowerBound — 主問題 $\mathcal{P}$ の目的関数値の下限．SDPA は $\sum_{i=1}^{m} c_i x_i^k <$ lowerBound となるような主問題の実行可能解 $(\boldsymbol{x}^k, \boldsymbol{X}^k)$ を得たときにアルゴリズムを終了する．もし、主問題が $\mathcal{P}$ 非有界で双対問題 $\mathcal{D}$ が実行可能解を持たない可能性がある場合は、lowerBound は十分小さく取ると良い．

- upperBound — 双対問題 $\mathcal{D}$ の目的関数値の上限．SDPA は $\boldsymbol{F}_0 \bullet \boldsymbol{Y}^k >$ upperBound となるような双対問題の実行可能解 $\boldsymbol{Y}^k$ を得たときにアルゴリズムを終了する．もし、双対問題が $\mathcal{D}$ 非有界で主問題 $\mathcal{P}$ が実行可能解を持たない可能性がある場合は、UpperBound は十分大きく取ると良い．

- betaStar — 現在の解 $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ が実行可能解のときに探索方向を制御するパラメータの一つ．小さめの betaStar > 0.0 を取るに連れて, 探索方向はセンタリング (中心パス付近に近づく操作) 無しのアフィンスケーリング方向に近づく．

- betaBar — 現在の解 $(\boldsymbol{x}^k, \boldsymbol{X}^k, \boldsymbol{Y}^k)$ が実行不可能解のときに探索方向を制御するパラメータの一つ．小さめの betaBar > 0.0 を取るに連れて, 探索方向はセンタリング (中心パス付近に近づく操作) 無しのアフィンスケーリング方向に近づく．また betaStar と betaBar の間には次のような関係がある必要がある．$0 \leq$ betaStar $\leq$ betaBar.

- gammaStar — ステップ長を制御するパラメータ．$0.0 <$ gammaStar < 1.0.

A.5 出力結果

この節では出力結果について説明する．

A.5.1 SDPA 実行時の画面への出力

例えば% **sdpa example1.dat example1.out**
と実行すると、SDPA は次のような情報を出力する．

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
Search Direction = HRVW/KSH/M
   mu      thetaP  thetaD   objValP   objValD  alphaP  alphaD  beta delta
 0 1.0e+06 1.0e+00 1.0e+00 +0.00e+00 +1.20e+04 1.0e+00 9.5e-01 0.10 0.0e+00
```

```
 1 6.3e+04 0.0e+00 4.9e-02 +8.16e+03 +5.53e+02 2.9e+01 9.6e-01 0.10 4.6e-02
 2 1.0e+04 0.0e+00 1.9e-03 +6.06e+03 -1.95e+01 2.7e+00 1.0e+00 0.10 8.2e-01
 3 3.1e+02 0.0e+00 0.0e+00 +5.74e+02 -4.19e+01 9.9e-01 9.5e+01 0.05 2.4e-01
 4 1.8e+01 0.0e+00 0.0e+00 -5.22e+00 -4.19e+01 1.0e+00 1.0e+00 0.05 4.0e-02
 5 9.5e-01 0.0e+00 0.0e+00 -4.00e+01 -4.19e+01 1.0e+00 1.0e+00 0.05 1.6e-03
 6 4.8e-02 0.0e+00 0.0e+00 -4.18e+01 -4.19e+01 1.0e+00 1.0e+00 0.05 2.6e-06
 7 2.4e-03 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.05 7.1e-12
 8 1.2e-04 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.05 2.7e-15
 9 6.0e-06 0.0e+00 0.0e+00 -4.19e+01 -4.19e+01 1.0e+00 1.0e+00 0.05 2.8e-15
phase.value = pdOPT
objValPrimal   = -4.189999e+01
objValDual     = -4.190000e+01
p. feas. error = 1.751488e-12
d. feas. error = 2.029488e-12
total time in seconds = 0.000000
```

- 最初の行 — もし存在すれば入力ファイル (この場合は example1.dat) の “Title and Comment” が出力される．

- Search direction = HRVW/KSH/M — この場合には HRVW/KSH/M 方向 [20, 26, 35] を選択している． さらに NT 方向 [38, 49], 及び AHO 方向 [1, 2] も選択することが出来る．詳しくは、A.6 節を参照のこと．

- mu — 相補性の平均値 $\boldsymbol{X}^k \bullet \boldsymbol{Y}^k/n$ (解の最適性の指標の一つ). $\mathcal{P}$ と $\mathcal{D}$ 共に実行可能ならば, 次の関係が保持される．

$$\begin{aligned} \text{mu} &= \left(\sum_{i=1}^{m} c_i x_i^k - \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k\right)/n \\ &= \frac{\text{主問題の目的関数値 - 双対問題の目的関数値}}{n} \end{aligned}$$

- thetaP — SDPA は、主問題 $\mathcal{P}$ の初期解 $(\boldsymbol{x}^0, \boldsymbol{X}^0)$ が実行可能ならば thetaP = 0.0 に設定して探索を開始する．また実行不可能ならば thetaP = 1.0 に設定する．そのため多くの場合 thetaP = 1.0 に設定して探索を開始する．また、k 反復目には thetaP は次の式で与えられる．

$$\text{thataP } = \frac{\|\boldsymbol{X}^k - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^k + \boldsymbol{F}_0\|}{\|\boldsymbol{X}^0 - \sum_{i=1}^{m} \boldsymbol{F}_i x_i^0 + \boldsymbol{F}_0\|};$$

  thetaP の値は単調非増加であり、0.0 に達したときに $(\boldsymbol{x}^k, \boldsymbol{X}^k)$ は実行可能解になっている．上の例では第 1 反復から主問題 $\mathcal{P}$ は実行可能になっている．

- thetaD — SDPA は、双対問題 $\mathcal{D}$ の初期解 $\boldsymbol{Y}^0$ が実行可能ならば thetaD = 0.0 に設定して探索を開始する．また実行不可能ならば thetaD = 1.0 に設定する．そのため多

くの場合 thetaD = 1.0 に設定して探索を開始する．また、k 反復目には thetaD は次の式で与えられる．

$$\text{thataD } = \frac{\left(\sum_{i=1}^{m}(\boldsymbol{F}_i \bullet \boldsymbol{Y}^k - c_i)^2\right)^{1/2}}{\left(\sum_{i=1}^{m}(\boldsymbol{F}_i \bullet \boldsymbol{Y}^0 - c_i)^2\right)^{1/2}};$$

thetaD の値は単調非増加であり、0.0 に達したときに $\boldsymbol{Y}^k$ は実行可能解になっている．上の例では第 3 反復から双対問題 $\mathcal{D}$ は実行可能になっている．

- objValP — 主問題の目的関数値.
- objValD — 双対問題の目的関数値.
- alphaP — 主問題におけるステップ長.
- alphaD — 双対問題におけるステップ長.
- beta — 探索方向を制御するパラメーター.
- delta — 現在の解が中心パスからどれだけ離れているかを示す指標．次の式で与えられる．

$$\text{delta } = \frac{\lambda_{\text{ave}} - \lambda_{min}}{\lambda_{\text{ave}}},$$

delta は [0, 1) の値の範囲を取る．ここで λ_{ave} と λ_{min} はそれぞれ $\boldsymbol{X}^k\boldsymbol{Y}^k$ の固有値の平均と最小値を示す．$\boldsymbol{X}^k$ と $\boldsymbol{Y}^k$ のどちらかが SDP の凸錐の境界に接近したときに delta の値は大きくなる．

- phase.value — SDPA が終了したときの状態を示す．具体的には、次の値の中から一つ選択される．pdOPT, noINFO, pFEAS, dFEAS, pdFEAS, pdINF, pFEAS_dINF, pINF_dFEAS, pUNBD, dUNBD.

  pdOPT : 主問題及び双対問題の両方が実行可能で、近似最適解 $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$ が得られたとき (正常終了).

  noINFO : 反復回数が上限 (maxIteration) に達し、主問題及び双対問題の実行可能性に関して何の情報も得られなかったとき.

  pFEAS : 主問題 $\mathcal{P}$ は実行可能になったが、反復回数が maxIteration に達して SDPA が終了するとき.

  dFEAS : 双対問題 $\mathcal{D}$ は実行可能になったが、反復回数が maxIteration に達して SDPA が終了するとき.

  pdFEAS : 主問題 $\mathcal{P}$ 及び双対問題 $\mathcal{D}$ は実行可能になったが、反復回数が maxIteration に達して SDPA が終了するとき.

pdINF： 少なくとも主問題 $\mathcal{P}$ か双対問題 $\mathcal{D}$ が実行不可能であると推定されるとき．より正確には、次のような条件の下で最適解が存在しない場合である．

$$\begin{aligned}
&\boldsymbol{O} \preceq \boldsymbol{X} \preceq \text{omegaStar} \times \boldsymbol{X}^0 = \text{lambdaStar} \times \text{omegaStar} \times \boldsymbol{I}, \\
&\boldsymbol{O} \preceq \boldsymbol{Y} \preceq \text{omegaStar} \times \boldsymbol{Y}^0 = \text{lambdaStar} \times \text{omegaStar} \times \boldsymbol{I}, \\
&\sum_{i=1}^{m} c_i x_i = \boldsymbol{F}_0 \bullet \boldsymbol{Y}.
\end{aligned}$$

pFEAS_dINF： 主問題 $\mathcal{P}$ は実行可能だが、双対問題 $\mathcal{D}$ が実行不可能であると推定されるとき．より正確には、次のような条件の下で双対問題に実行可能解が存在しない場合である．

$$\boldsymbol{O} \preceq \boldsymbol{Y} \preceq \text{omegaStar} \times \boldsymbol{Y}^0 = \text{lambdaStar} \times \text{omegaStar} \times \boldsymbol{I}.$$

pINF_dFEAS： 双対問題 $\mathcal{D}$ は実行可能だが、主問題 $\mathcal{P}$ が実行不可能であると推定されるとき．より正確には、次のような条件の下で主問題に実行可能解が存在しない場合である．

$$\boldsymbol{O} \preceq \boldsymbol{X} \preceq \text{omegaStar} \times \boldsymbol{X}^0 = \text{lambdaStar} \times \text{omegaStar} \times \boldsymbol{I}.$$

pUNBD： 主問題が非有界であると推定されるとき．より正確には、主問題の目的関数値が lowerBound より小さくなったときである．

$$\text{objValP} = \sum_{i=1}^{m} c_i x_i^k < \text{lowerBound}.$$

dUNBD： 双対問題が非有界であると推定されるとき．より正確には、双対問題の目的関数値が upperBound より大きくなったときである．

$$\text{objValD} = \boldsymbol{F}_0 \bullet \boldsymbol{Y}^k > \text{upperBound}.$$

A.5.2 出力ファイルの構成

次に example2.dat を SDPA で実行したときの出力ファイル example2.out を示す．

```
Data   file name = example2.dat
Output file name = example2.out
*Example 2:
*mDim = 5, nBLOCK = 3, {2,3,-2}
maxIteration = 100
epsilonStar  = 1.00e-06
lambdaStar   = 1.00e+03
omegaStar    = 2.00e+00
lowerBound   = -1.00e+05
```

```
upperBound   = 1.00e+05
betaStar     = 5.00e-02
betaBar      = 1.00e-01
gammaStar    = 9.50e-01
Search Direction = HRVW/KSH/M
   mu      thetaP  thetaD   objValP   objValD  alphaP  alphaD  beta delta
 0 1.0e+06 1.0e+00 1.0e+00 +0.00e+00 +1.44e+04 9.2e-01 7.1e-01 0.10 0.0e+00
 1 2.3e+05 7.9e-02 2.9e-01 +4.35e+03 +1.81e+03 1.0e+00 8.9e-01 0.10 7.5e-01
 2 3.1e+04 0.0e+00 3.2e-02 +6.60e+03 -9.62e+01 9.6e-01 9.6e-01 0.10 5.3e-01
 3 2.4e+03 0.0e+00 1.2e-03 +7.47e+03 -5.23e-01 1.6e+00 1.0e+00 0.10 4.7e-01
 4 1.3e+02 0.0e+00 0.0e+00 +8.93e+02 -2.19e-01 9.4e-01 5.0e+00 0.05 4.2e-01
 5 1.1e+01 0.0e+00 0.0e+00 +9.84e+01 +2.43e+01 9.0e-01 1.0e+00 0.05 8.9e-01
 6 1.5e+00 0.0e+00 0.0e+00 +3.81e+01 +2.77e+01 1.0e+00 8.0e-01 0.14 9.2e-01
 7 3.2e-01 0.0e+00 0.0e+00 +3.29e+01 +3.07e+01 9.8e-01 9.9e-01 0.10 7.4e-01
 8 3.7e-02 0.0e+00 0.0e+00 +3.22e+01 +3.19e+01 9.7e-01 9.9e-01 0.05 4.9e-01
 9 2.6e-03 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.8e-01 1.0e+00 0.05 4.0e-01
10 1.5e-04 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.05 2.4e-01
11 7.9e-06 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.05 1.8e-01
12 4.0e-07 0.0e+00 0.0e+00 +3.21e+01 +3.21e+01 9.9e-01 1.0e+00 0.05 1.7e-01
Computational Result :
No of Iterations =  12
phase.value = pdOPT
mu0 = 1.000e+06
mu  = 4.018e-07
objValPrimal   = 3.206269e+01
objValDual     = 3.206269e+01
p. feas. error = 3.979039e-13
d. feas. error = 2.207390e-11
.xVect =
{+1.552E+00,+6.710E-01,+9.815E-01,+1.407E+00,+9.422E-01}
.xMat =
{
{ {+1.87445293E-07,-2.82970428E-08 },
  {-2.82970428E-08,+1.32845207E-07 }   }
{ {+7.11917496E+00,+5.02466973E+00,+1.91631412E+00 },
  {+5.02466973E+00,+4.41475734E+00,+2.50604256E+00 },
  {+1.91631412E+00,+2.50604256E+00,+2.04812099E+00 }   }
{+3.43233951E-01,+4.39115507E+00 }
}
.yMat =
```

```
{
{ {+2.64030415E+00,+5.60567528E-01 },
  {+5.60567528E-01,+3.71764541E+00 }   }
{ {+7.61547114E-01,-1.51352929E+00,+1.13936202E+00 },
  {-1.51352929E+00,+3.00804909E+00,-2.26441411E+00 },
  {+1.13936202E+00,-2.26441411E+00,+1.70461710E+00 }   }
{+9.97628420E-07,+7.80238872E-08 }
}
.xEigenValues =
{
{+1.995E-07,+1.208E-07}
{+1.192E+01,+1.666E+00,+8.936E-08}
{+3.432E-01,+4.391E+00}
}
.yEigenValues =
{
{+2.402E+00,+3.956E+00}
{+5.474E+00,+2.861E-08,+2.032E-07}
{+9.976E-07,+7.802E-08}
}
.xyEigenValues =
{
{+4.791E-07,+4.780E-07}
{+4.958E-07,+3.399E-07,+3.350E-07}
{+3.424E-07,+3.426E-07}
}
total time in seconds = 0.016667
```

- xVect — 主問題の変数ベクトル $\boldsymbol{x}$.
- xMat — 主問題の変数行列 $\boldsymbol{X}$.
- yMat — 双対問題の変数行列 $\boldsymbol{Y}$.
- xEigenValues — 行列 $\boldsymbol{X}$ の固有値. $\boldsymbol{X}$ が半正定値行列なので、全ての固有値は非負でなければならない.
- yEigenValues — 行列 $\boldsymbol{Y}$ の固有値. $\boldsymbol{Y}$ が半正定値行列なので、全ての固有値は非負でなければならない.
- xyEigenValues — 行列 $\boldsymbol{XY}$ の固有値. 全ての値が十分 0 に近いかを見ることによって、解の最適性を確認することが出来る.

A.6 SDPA の高度な使用法

A.6.1 SDPA に組み込まれている３種類の探索方向について

SDPA は Version 2.0 から３種類の探索方向を組み込んでいる．既に述べたように、一つ目は HRVW/KSH/M[20, 26, 35] 方向であり、二つ目は NT[38] 方向である．そして三つ目は AHO[1, 2] 方向である．SDPA の実行時に “-1”, “-2”,“-3” のいずれかのオプションを選択することによってこれらの探索方向を切り替えて実行することが出来る．例えば、example2.dat を NT 方向を用いて実行する場合には、

% sdpa example2.dat example2.out −2 と入力する．

同様に、exmaple1.dat を AHO 方向を用いて実行するには

% sdpa example1.dat example1.out −3 と入力する．

なお “-1” を入力するか、いずれのオプションも入力しないときには HRVW/KSH/M 方向が自動的に選択される．

A.6.2 初期点

SDPA では、あらかじめ初期点 (解) を用意している場合には、その初期点からアルゴリズムを開始することが出来る．初期点 $(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0)$ はかならずしも実行可能解でなくとも良いが、$\boldsymbol{X}^0 \succ \boldsymbol{O}, \boldsymbol{Y}^0 \succ \boldsymbol{O}$ を満たしていなければならない．もし、例題１で次のような初期点を使用したいならば、

$$(\boldsymbol{x}^0, \boldsymbol{X}^0, \boldsymbol{Y}^0) = \left(\begin{pmatrix} 2.0 \\ 0.0 \\ 0.0 \end{pmatrix} \begin{pmatrix} 31.0 & 3.0 \\ 3.0 & 5.0 \end{pmatrix} \begin{pmatrix} 2.0 & 0.0 \\ 0.0 & 2.0 \end{pmatrix} \right),$$

次のように入力する．

% sdpa example1.dat example1.out example1.ini

ここで “example1.ini” には以下のような記述を行う．

```
{0.0, -4.0, 0.0}
{ {11.0, 0.0}, {0.0, 9.0} }
{ {5.9,  -1.375}, {-1.375, 1.0} }
```

初期点ファイルは、混乱を避けるためにファイルの名前の最後に “.ini” を付けることを推奨するが、特にファイル名に規定は無い．また、初期点ファイルの指定は必ず出力ファイルの後に行う必要がある．また他のオプション “−c”, “−1”, “−2”, “−3” とは順不同であり、次のような指定が可能である．

% sdpa example1.dat example1.out example1.ini −c −3

% sdpa example1.dat example1.out −3 −c example1.ini

また、当然のことながら “−c”, “−1”, “−2”, “−3” の名前を持つ初期点ファイルは作成出来ない.

初期点ファイルは、以下のような構成にする.

$\boldsymbol{x}^0$
$\boldsymbol{X}^0$
$\boldsymbol{Y}^0$

記述の方法は、A.3 節で示した入力ファイルの記述方法と基本的に同じである．$\boldsymbol{x}^0$ は 定数ベクトル $\boldsymbol{c}$ と同様に記述し、$\boldsymbol{X}^0$ と $\boldsymbol{Y}^0$ は定数行列 $\boldsymbol{F}_i$ と同様に記述する.

A.6.3 疎行列の入力データ形式

A.3 節では、ブロック対角行列 $\boldsymbol{F}$(A.2) でかつ各ブロック $\boldsymbol{B}_i$ が密行列の場合の入力データ形式について解説した．しかし各ブロック $\boldsymbol{B}_i$ が疎行列である場合は、前述の入力データ形式は効率が悪くなる．そこでこの節では、SDPA で使用可能な疎行列の入力データ形式について説明する．なお、後述するように初期点の入力にもこのデータ形式を使用することが出来る．しかし混乱を防ぐために一つの入力ファイルに密行列と疎行列のデータ形式の混在は出来ないようになっている.

まず、疎行列のデータ形式を使用した入力ファイルの名前には末尾に “-s” を付ける必要がある．よって例題 1 の場合には “example1.dat-s” といった名前にする必要がある．つぎに “example1.dat-s” の例を示す.

```
"Example 1: mDim = 3, nBLOCK = 1, {2}"
   3  =  mDIM
   1  =  nBLOCK
   2  = bLOCKsTRUCT
{48, -8, 20}
0 1 1 1 -11
0 1 2 2 23
1 1 1 1 10
1 1 1 2 4
2 1 2 2 -8
3 1 1 2 -8
3 1 2 2 -2
```

“example1.dat” と比較した場合、5 行目の定数行列 c の記述までは同じである．この例では、6 行目から定数行列 $\boldsymbol{F}_i$ の記述に入る．ここで注意したいのは、ある 1 行にはある定数行列 $\boldsymbol{F}_i$ の 1 要素の記述のみを書き、項目数はかならず 5 個である．6 行目の “0 1 1 1

-11” は、$\boldsymbol{F}_0$ の 1 番目のブロックの $(1,1)$ 要素が -11 であることを示している．また同様に１１行目の “3 1 1 2 -8” は $\boldsymbol{F}_3$ の 1 番目のブロックの $(1,2)$ 要素が -8 であることを示している．

一般的には、疎行列の入力データ形式は以下のような構造をしている．

ファイル名または注釈行 (なくても良い)
m — 主問題の変数 x の数 (または双対問題の制約式の数)
nBLOCK — ブロック対角行列のブロック数
bLOCKsTRUCT — ブロック対角行列の構造ベクトル
$\boldsymbol{c}$
$k_1\ b_1\ i_1\ j_1\ v_1$
$k_2\ b_2\ i_2\ j_2\ v_2$
$\cdots$
$k_p\ b_p\ i_p\ j_p\ v_p$
$\cdots$
$k_q\ b_q\ i_q\ j_q\ v_q$

ここで $k_p \in \{0,1,\ldots,m\}$, $b_p \in \{1,2,\ldots,\mathrm{nBLOCK}\}$, $1 \le i_p \le j_p$ そして $v_p \in R$ である．各行の “$k_p,\ b_p,\ i_p,\ j_p,\ v_p$” は $\boldsymbol{F}_{k_p}$ の b_p ブロックの (i_p, j_p) 番目の要素が v_p であることを示している．もし b_p 番目のブロックが $\ell \times \ell$ 対称で (対角でない) 行列ならば、(i_p, j_p) は $1 \le i_p \le j_p \le \ell$ を満たす必要がある．つまり b_p 番目のブロックの上三角部分のみ記述すればよいということである．また b_p 番目のブロックが $\ell \times \ell$ の対角行列ならば (i_p, j_p) は $1 \le i_p = j_p \le \ell$ を満たす必要がある．

A.6.4 疎行列データ構造を用いた初期点の入力

以下に “example1.ini” を疎行列データ入力形式に変換したファイル “example1.ini-s” を示す．この場合も疎行列データ入力形式を使用していることを示すためにファイル名の末尾に “-s” を付ける必要がある．

```
{0.0, -4.0, 0.0}
1 1 1 1 11
1 1 2 2 9
2 1 1 1 5.9
2 1 1 2 -1.375
2 1 2 2 1
```

“example1.ini” と比較した場合、1 行目の $\boldsymbol{x}^0$ の記述法は同じである．その後は、$\boldsymbol{X}^0$, $\boldsymbol{Y}^0$ の順に記述を行う．記述方法は、前述した “example1.dat-s” とほぼ同じであるが、若干補足する．まず 2 行目の “1 1 1 1 11” は、$\boldsymbol{X}^0$ の 1 番目のブロックの $(1,1)$ 要素が 1 であることを記述している．また同様に 5 行目の “2 1 1 2 -1.375” は、$\boldsymbol{Y}^0$ の 1 番目のブロックの

(1, 2) 要素が -1.375 であることを示している．一般的には、疎行列の入力データ形式は以下のような構造をしている．

$\boldsymbol{x}^0$
$s_1\ b_1\ i_1\ j_1\ v_1$
$s_2\ b_2\ i_2\ j_2\ v_2$
$\cdots$
$s_p\ b_p\ i_p\ j_p\ v_p$
$\cdots$
$s_q\ b_q\ i_q,\ j_q\ v_q$

ここで、$s_p = 1$ または 2, $b_p \in \{1, 2, \ldots, \mathrm{nBLOCK}\}$, $1 \leq i_p \leq j_p$ そして $v_p \in R$ である．$s_p = 1$ のとき各行 "$s_p\ b_p\ i_p\ j_p\ v_p$" は $\boldsymbol{X}^0$ の b_p 番目のブロックの (i_p, j_p) 要素が v_p であることを示している．また $s_p = 2$ のときは, 各行 $s_p\ b_p\ i_p\ j_p\ v_p$" は $\boldsymbol{Y}^0$ の b_p 番目のブロックの (i_p, j_p) 要素が v_p であることを示している．もし b_p 番目のブロックが $\ell \times \ell$ 対称で (対角でない) 行列ならば、 (i_p, j_p) は $1 \leq i_p \leq j_p \leq \ell$ を満たす必要がある．つまり b_p 番目のブロックの上三角部分のみ記述すればよいということである．また b_p 番目のブロックが $\ell \times \ell$ の対角行列ならば (i_p, j_p) は $1 \leq i_p = j_p \leq \ell$ を満たす必要がある．

A.6.5 疎行列データ入力ファイルの使用法についての補足

これまでに、例題 1 についてデータファイルは "example1.dat" と "example1.dat-s" の二種類．また初期点ファイルは、"example1.ini" と "example1.ini-s" の二種類について説明を行った．前述したように、ファイルの名前の末尾に "-s" がついていれば、自動的に疎行列データ入力形式と判別可能なので、以下のような使用法は全て可能である．

sdpa example1.dat example1.out example1.ini
sdpa example1.dat example1.out example1.ini-s
sdpa example1.dat-s example1.out example1.ini
sdpa example1.dat-s example1.out example1.ini-s

Appendix B

SDPA の変遷および入手方法

B.1 SDPA について

SDPA の前身は、Mathematica Ver 2.2 で記述された **pinpal** (小島 1994) **ftp://ftp.is.titech.ac.jp/pub/OpRes/software/pinpal.tar.Z** である．この pinpal は、SDP に対する主双対内点法のソフトウェアとして当時では非常に画期的であった．しかし、Mathematica がインタープリタであるのと、疎行列等を効率良く扱うことが非常に困難であるために小規模な問題を解くことしかできなかった．

そのため、大規模な問題を効率良く解くために次のような方針で SDP を解くソフトウェアを開発するプロジェクトを開始した．

1. 高度な記述ができる、移植性が高い、動作が安定して広く普及している等の理由で ANSI C++ 言語を用いて作成を行う．
2. 動的なメモリの確保、解放を行うことによって、解くことのできる問題の大きさにソフトウェア上の制限を設けないことにする．つまり、どのくらい大きな問題が解けるかは実行するコンピュータのハードウェア資源に依存する．
3. SDP では疎行列を入力に含む場合が多いので、この特性を活かしたデータ構造及び計算方法を開発する．
4. 作成したソフトウェアは、原則フリーソフトとしてインターネット上から配布する．

次に SDPA の変遷について簡潔に示す．

SDPA の変遷

1. **pinpal** (小島 1994 年 11 月) : Mathematica Ver 2.2 で作成されたソフトウェア.
2. **SDPA Ver 0.1** (小島 1995 年 4 月) : Borland C++ で開発された. そのあと UNIX 上に移植 (藤沢 1995 年 4 月).
3. **SDPA Ver 1.0** (藤沢, 小島 1995 年 12 月) : 疎行列を扱えるように変更を加え、乗算、加算、内積等を高速化. また数値演算ライブラリ meschach [48] を用いて線形方程式、固有値計算等を高速化.
   ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA 上で配布を開始.
4. **SDPA Ver 2.0** (藤沢, 小島, 中田 1996 年 8 月、以下作成者同じ) : 3 種類の探索方向 (HRVW/KSH/M, NT, AHO) を組み込む. 疎行列を活かして探索方向の計算の高速化に成功、特に HRVW/KSH/M と NT 方向では著しく改善. Mehrotra のプリディクター・コレクター法 ([32]) を採用.
5. **SDPA Ver 2.1** (1997 年 1 月) : Ver 2.0 のバグを修正.
6. **SDPA Ver 3.0**β (1997 年 1 月) : [11] で提案した手法を実現するためにデータ構造を大幅に変更. HRVW/KSH/M 方向のみだが大幅な高速化を実現 (詳しくは [11]).
7. **SDPA Ver 3.1**β (1997 年 7 月) : さらに NT, AHO 方向も新データ構造に対応.
8. **SDPA Ver 4.0** (1997 年 10 月) : [11] の高速化手法にブロック対角行列レベルにまで完全対応. 一部データ構造を変更. Ver 3.x でのバグを修正.
9. **SDPA Ver 4.02** (1997 年 12 月) : HRVW/KSH/M および NT 方向の計算時にコレスキー分解を採用. Ver 4.0 でのバグを修正.

次に、インターネット上で入手することのできるソフトウェアを紹介する.

SDP を解く主双対内点法のソフトウェア

1. **SP** : (L. Vandenberghe & S. Boyd) ⟶
   http://www-ISL.Stanford.EDU/people/boyd/SP.html
2. **SDPSOL** : (Boyd & Wu) ⟶
   http://www-isl.stanford.edu/ boyd/sdpsol.html
3. **SDPT3** : (K.C. Toh & M.J. Todd & R.H.Tutuncu) ⟶
   http://www.math.nus.sg/ mattohkc/index.html
4. **SDPHA** : (F.A. Potra & R. Sheng & N. Brixius) ⟶
   http://www.math.uiowa.edu/ rsheng/SDPHA/sdpha.html
5. **SDPpack** : (F. Alizadeh &J-P A. Haeberly & M. V. Nayakkankuppam & M. L. Overton) ⟶
   http://www.cs.nyu.edu/phd_students/madhu/sdppack
6. **CSDP** : (B. Borchers) ⟶
   http://www.nmt.edu/ borchers/csdp.html
7. **SDPA** (K. Fujisawa & M. Kojima & K. Nakata) ⟶
   ftp://ftp.is.titech.ac.jp/pub/OpRes/software/SDPA

以上のように、主要なソフトウェアを集めただけでも 7 つほど存在する．この中で C++ で記述され、疎行列等に本格的に対応しているのは現時点では SDPA だけであり、高速でかつ大規模な問題が解けるようになっている．また、SDPA は主要な検索エンジン AltaVista **http://www.altavista.digital.com/** や YAHOO **http://www.yahoo.com/** から

http://SAL.KachinaTech.COM/B/3/SDPA.html
http://orly1.snu.ac.kr/ kjw/orms.html
http://www.cs.uiowa.edu/ brixius/sdplinks.html

⋮

など 10 個所以上リンクが張られている。よってインターネットさえ使用できれば、だれでも瞬時に検索して利用することができるという大きな利点を持っている．

謝辞

東京工業大学の小島政和教授には、研究全般にわたり貴重な助言、意見等をいただきました．共同研究を行えたことによって、教授の持たれている 主双対内点法の理論、論文作成の技術などに触れ、その一部を吸収できたことは大きな喜びでもあり、今後の研究者生活の中で貴重な財産となることと思います．また東京商船大学の久保幹雄助教授には、学部４年生のころから目をかけていただきました．多くの共同研究を行うことによって、いろいろな面で技術を収得することができました．また常に最新の情報を提供してくれました．ここに謝意を表します．また小島研究室の学生の皆さんにも大変お世話になりました．特に修士２年の中田和秀君には、様々なアイデアを提供していただいただけではなく、プログラム作成も手伝っていただきました．また修士１年の福田光浩君にも、情報収集や数値実験などを手伝っていただきました．東京工業大学間瀬研究室４年生の池添禎孝君を始めとして、間瀬研究室のみなさんにもお世話になりました．最後に今日にいたるまでずっと励まして支えてくれた両親に謝意を表したいと思います．

参考文献

[1] F. Alizadeh, J. -P. A. Haeberly and M. L. Overton, "Primal-dual interior-point methods for semidefinite programming," Working Paper, 1994.

[2] F. Alizadeh, J. -P. A. Haeberly and M. L. Overton, "Primal-dual interior point methods for semidefinite programming: convergence rates, stability and numerical results," Report 721, Computer Science Department, New York University, New York, NY, 1966.

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

[4] R. Bellman and K. Fan, "On systems of linear inequalities in Hermittan matrix variables," in Convexity, V.L. Klee, ed., Vol 7 Proc. Symposia in Pure Mathematics, Amer. Math. Soc., Providence, RI, 1963, 1–11.

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

[6] S. Boyd, L.E. Ghaoui, E. Feron and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory,* (SIAM, Philadelphia, 1994).

[7] N. Brixius, F.A. Potra and R. Sheng, "Solving semidefinite programming in mathematica," Reports on Computational Mathematics, No 97/1996, Dept. of Mathematics, University of Iowa, October 1996. Available at http://www.cs.uiowa.edu/ brixius/sdp.html.

[8] B. Craven and B. Mond, "Linear programming with matrix variables," Linear Algebra Appl., 38, 1981, 73–80.

[9] I. Dikin, "Iterative solution of problems of linear and quadratic programming," Soviet Math. Dokl., 8 (1967), 674-675.

[10] K. Fujisawa, M. Kojima and K. Nakata, "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,

December 1995, Revised August 1996. Available via anonymous ftp at ftp.is.titech.ac.jp in pub/OpRes/software/SDPA.

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

[12] K. Fujisawa, M. Fukuda, M. Kojima and K. Nakata, "Numerical Evaluation of the SDPA (SemiDefinite Programming Algorithm)", submitted to Proceedings of the Second Workshop on High Performance Optimization Techniques, held in Rotterdam, The Netherlands, August 1997.

[13] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. 1994. unpublished manuscript.

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

[15] C. Goh and D. Teo, "On minimax eigenvalue problems via constrained optimization," J. Optim. Theory Appl., 57(1988), 59–68.

[16] M, Grötschel, Lovász and A. Schrijver, "Polynomial algorithms for perfect graphs," *Annals of Discrete Mathematics* 21 (1984) 325-356.

[17] M, Grötschel, Lovász and A. Schrijver, *Geometric algorithms and combinatorial optimization* (Springer, New York, 1988).

[18] 原辰次, "ロバスト制御理論の動向 – 実用的な制御計設計法を目指して –", Proceedings of the 38th Annual Conference of the ISCIE (1994) 25–27.

[19] C. Helmberg, F. Rendl, R.J. Vanderbei and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization* **6** (1996) 342–361.

[20] C. Helmberg, F. Rendl, R. J. Vanderbei and H. Wolkowicz, "An interior-point method for semidefinite programming," *SIAM Journal on Optimization* **6** (1996) 342–361.

[21] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part I, graph partitioning. *Operations Research*, 37:865–892, 1989.

[22] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica* **4** (1984) 373–395.

[23] M. Kojima, S. Mizuno and A. Yoshise, "A primal-dual interior point algorithm for linear programming," In N. Megiddo, ed., *Progress in Mathematical Programming, Interior-Point and Related Methods* (Springer-Verlag, New York, 1989) 29–47.

[24] M. kojima, N. Megiddo, T. Noma and A. Yoshise, "A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems", Lecture Note in Computer Science, Springer-Verlag, New York, Berlin, 1991.

[25] M. Kojima, S. Kojima and S. Hara, "Linear algebra for semidefinite programming," Research Reports B-290, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, October 1994.

[26] M. Kojima, S. Shindoh and S. Hara, "Interior-point methods for the monotone semidefinite linear complementarity problems," *SIAM Journal on Optimization* **7** (1997) 86–125.

[27] M. Kojima, M. Shida and S. Shindoh, "Local Convergence of Predictor-Corrector Infeasible-Interior-Point Algorithms for SDPs and SDLCPs" Research Report #306, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152, Japan, December 1995, Revised October 1996.

[28] C.-J. Lin and R. Saigal, "A predictor-corrector method for semi-definite linear programming," Working paper, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan 48109-2117, October 1995.

[29] L. Lovász and A. Schrijver, "Cones of matrices and set functions and 0-1 optimization," *SIAM J. on Optimization* 1 (1991) 166-190.

[30] Z.-Q. Luo, J.F. Sturm and S. Zhang, "Superlinear convergence of a symmetric primal-dual path following algorithm for semidefinite programming," to appear in *SIAM Journal on Optimization.*

[31] C.-J. Lin and R. Saigal, "Predictor corrector methods for semidefinite programming," Informs Atlanta Fall Meeting, November 3-6, 1996.

[32] S.Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on Optimization* **2** (1992) 575–601.

[33] O. L. Mangasarian, *Nonlinear Programming* (McGraw-Hill, New York, 1969).

[34] N. Megiddo, "Pathways to the optimal set in linear programming," In N. Megiddo, ed., *Progress in Mathematical Programming, Interior-Point and Related Methods* (Springer-Verlag, New York, 1989) 131–158.

[35] R.D.C. Monteiro, "Primal-Dual Path Following Algorithms for Semidefinite Programming," to appear in *SIAM Journal on Optimization.*

[36] Y. E. Nesterov and A. S.Nemirovskii, "A general approach to polynomial-time algorithms design for convex programming," Tech. Report, Centr. Econ. and Math. Inst., USSR Acad. Sci., Moscow, USSR, 1988.

[37] Y. E. Nesterov and A. S.Nemirovskii, *Interior Point Polynomial Algorithms in Convex Programming* (SIAM, Philadelphia, 1993).

[38] Ju. E. Nesterov and M. J. Todd, "Self-scaled cones and interior-point methods in nonlinear programming," Working Paper, CORE, Catholic University of Louvain, Louvain-la-Neuve, Belgium, April 1994.

[39] Yu.E. Nesterov and M.J. Todd, "Primal-dual interior-point methods for self-scaled cones," Technical Report 1125, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York 14853-3801, USA, 1995.

[40] M. Overton, "On minimizing the maximum eigenvalue of a symmetric matrix," SIAM J. Matrix Anal. Appl., 9 (1988), 256–268.

[41] E. Panier, "On the need for special purpose algorithms for minimax eigenvalue problems," J. Optim. Theory Appl., 62 (1989), 279–287.

[42] S. Poljak and F. Rendl. Nonpolyhedral relaxations of graph-bisection problems. 1993. unpublished manuscript.

[43] S.Poljak, F.Rendl and H.Wolkowicz, "A recipe for semidefinite relaxation for (0,1) quadratic programming," *Journal of Global Optimization* **7** (1995) 51–73.

[44] F.A. Potra and R. Sheng, "Superlinear convergence of infeasible-interior-point algorithms for semidefinite programming," Department of Mathematics, University of Iowa, Iowa City, IA 52242, April 1996.

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

[46] K. Tanabe, "Centered Newton method for mathematical programming," In M. Iri and K. Yajima, eds., *System Modeling and Optimization* (Springer-Verlag, New York, 1988) 197-206.

[47] A. Shapiro, "Weighted minimum trace factor analysis," Psychometrika, 47, 1982, 243–264.

[48] D. E. Stewart and Z. leyd, *Meschach:* Matrix Computation in C," Proceedings of the Center for Mathematics and Its Applications, The Australian National University, Volume 32, 1994.

[49] M.J. Todd, K.C. Toh and R.H. Tütüncü, "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, March 1996, Revised May 1996.

[50] K.C. Toh, M.J. Todd and R.H. Tütüncü, "SDPT3 – a MATLAB software package for semidefinite programming," Dept. of Mathematics, National University of Singapore, 10 Kent Ridge Crescent, Singapore, December 1996. Available at http://www.math.nus.sg/ mattohkc/index.html.

[51] K.C. Toh and L.N. Trefethen, "The Chebyshev polynomial of matrix", manuscript, Center for Applied Mathematics, Cornell University, Ithaca, NY, 1996.

[52] L. Vandenberghe and S. Boyd, "A primal-dual potential reduction method for problems involving matrix inequalities," *Mathematical Programming, Series B* **69** (1995) 205–236.

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

[54] H. Wolkowicz, "Some applications of optimization in matrix theory," Linear Algebra Appl., 40, 1981, 101-118.

[55] 吉瀬章子, "凸計画問題に対する最適化手法 – 内点法と解析中心", システム／制御／情報, Vol.3, No.3. 155–160, 1994.

[56] D. B. Yudin and A. S. Nemirovsky, "Informational complexity and efficient methods for solving complex extremal problems," Matekon, 13 (1977), 25–45.

[57] Y. Zhang, "On Extending Primal-Dual Interior-Point Algorithms from Linear Programming to Semidefinite Programming," to appear in *SIAM Journal on Optimization.*

[58] Q. Zhao, S.E. Karisch, F. Rendl and H. Wolkowicz, "Semidefinite programming relaxations for the quadratic assignment problem," CORR Report 95-27, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, September 1996.