適応的にブロックサイズを決定する Gram-Schmidt 法の研究

2015 年 3 月

慶應義塾大学大学院 理工学研究科

松尾 洋一

学位論文　博士　(工学)

要　旨

科学技術計算における Gram-Schmidt (GS) 法は，行列計算の中で重要な算法の 1 つであり，今までに多くの GS の計算手法が提案されている．その応用分野は広く，固有値問題を解く Arnoldi 法や Lanczos 法，特殊な構造を持つ固有値問題に対する Symplectic 法，線形方程式を解くための GMRES 法のような Krylov 部分空間法に GS 法は使われている．

本論文では GS 法のブロック化とブロックサイズについて考察し，GS 法の改良手法とその効率的な実装を提案する．Block Gram-Schmidt（BGS）法は，行列 X を m 列のブロック行列ごとに直交化することで，正規直交基底列の計算を高速に行う手法である．しかし，行列 X を分割するブロックサイズの大きさ m の取り方によって計算速度が変わるので，ユーザーが問題ごとにブロックサイズを検討し，適宜に決定する必要がある．そこで本論文では，計算時間の増加と計算量の増加に着目し，計算量を用いてブロックサイズごとの計算時間を予測することによって，適応的にブロックサイズを決定する手法を提案する．

近年，大規模な行列問題を高速に解くことが必要となり，そのためには並列化により計算を高速に処理することが重要である．本論文では列方向分散によって BGS 法を並列化し，並列化による速度向上について検討する．さらに，並列化されたブロックサイズの決定手法を用いた Parallel Block Gram-Schmidt(PBGS) 法を提案する．

また Symplectic Gram-Schmidt（SGS）法は，Gram-Schmidt 法の応用手法の 1 つであり，Hamiltonian 行列のような特殊構造を持つ行列の固有値を求めることに適している．現在，大規模疎行列に対しては Symplectic Lanzcos 法が提案されており，そこで Symplectic なベクトル列を生成する際に SGS 法が使われている．しかし，SGS 法はパラメーターの選択によっては J-直交性の崩れが起きやすいなどの問題点が指摘されている． 本論文では，SGS 法のブロック化を提案し，算法の高速化について考察する．そして適応的なブロックサイズの決定手法を SGS 法に適用させるとともに，再直交化の条件についても検討し，J-直交行列を安定的に高速に計算する手法を提案する．

最後に数値実験を用いて，これらの提案手法の有用性の検証を行う．

A Study on an Adaptive Gram-Schmidt Method for Determining the Block-size

Abstract

In scientific computation, the Gram-Schmidt procedure is one of the most important algorithms. Variants of Gram-Schmidt methods have been studied extensively. Examples of these algorithms are as follows : the Lanczos and Arnoldi methods for solving eigenvalue problems, and the Krylov subspace method like the GMRES for solving linear systems of equations. In the structure-preserving method for Hamiltonian eigenvalue problems, an orthogonalization process is also used.

In this thesis, we propose new procedures of the Block Gram-Schmidt (BGS) method and the Symplectic Gram-Schmidt (SGS) method. BGS computes orthonormal vectors rapidly by partitioning the columns of matrix X, which are then orthogonalized into blocks with a block-size of m. However, since the optimal block-size m is not consistent when using BGS, it is integral to determine block-size m. It should be noted that there is no unique block-size m for all computations, and this is why it is necessary to determine m accurately through trial and error. In this thesis we propose a new scheme for determining the optimal block-size m by focusing on the computation time and cost.

Recently, it has become necessary to parallelize various methods that solve the large scale problems for simulating complex phenomenon or improving simulations. In this thesis, we parallelized the BGS through Column-Wise Distribution, recorded the speed-up, and applied the parallelized new scheme for determining the optimal block size for parallel BGS.

The Gram-Schmidt procedure is also used in the Symplectic methods which are structure-preserving methods for solving eigenvalue problems arising from special matrices like the Hamiltonian matrix. It enables us to compute eigenvalues faster by using the structure of a matrix unlike the QR or Lanczos method. For large scale problems, the symplectic Lanczos method has been proposed, which uses the Symplectic Gram-Schmidt method (SGS) to compute symplectic vectors. However, previous studies have shown that the selection process of the parameter in the SGS method is flawed, as it results in a par-

tially destroyed J-orthogonality of the J-orthogonal matrix. In this thesis, we propose the block type of the SGS and a new condition for the reorthogonalization to maintain J-orthogonality. Applying the block-size scheme to this method, we propose a new procedure for computing symplectic vectors with much more rapidity and stability.

Lastly, some numerical experiments will be shown, to illustrate and evaluate the effectiveness of our proposed algorithms.

目 次

1 序論 **1**

1.1 背景 1

1.2 本論文の目的と構成 4

1.3 実験環境 5

1.4 数値例 5

1.4.1 数値例 1 6

1.4.2 数値例 2 6

1.4.3 数値例 3 6

1.4.4 数値例 4 9

2 Block Gram-Schmidt 法 **11**

2.1 Classical Gram-Schmidt 法 11

2.1.1 算法 11

2.1.2 再直交化 12

2.2 Block Gram-Schmidt 法 14

2.2.1 算法 15

2.2.2 再直交化 16

2.3 ブロックサイズ 16

2.3.1 計算量 16

2.3.2 計算時間 19

2.4 ブロックサイズの最適化 21

2.4.1 Scheme A 21

2.4.2 Scheme B 22

2.4.3 Scheme C 23
2.5 数値実験 25
2.5.1 数値実験 1 26
2.5.2 数値実験 2 27
2.5.3 数値実験 3 28
2.6 まとめ 28

3 Parallel Block Gram-Schmidt 法 30

3.1 並列化 30
3.1.1 分散方法 30
3.1.2 列方向分散を用いた Parallel Block Gram-Schmidt 法 31
3.2 計算量 31
3.3 最適なブロックサイズの検討 32
3.3.1 Scheme D 32
3.4 数値実験 32
3.4.1 数値実験 1 32
3.4.2 数値実験 2 34
3.5 まとめ 35

4 Block Symplectic Gram-Schmidt 法 37

4.1 Symplectic Gram-Schmidt 法 37
4.1.1 表記法 38
4.1.2 算法 39
4.1.3 J-直交性の崩れ 43
4.1.4 Symplectic Gram-Schmidt 法の再直交化 44
4.2 ブロック化 45
4.3 適応的なブロックサイズの決定手法 47
4.3.1 計算量 47
4.3.2 Scheme E 48
4.4 数値実験 48
4.4.1 数値実験 1 49

4.4.2 数値実験 2 55
4.5 まとめ 56

5 総括と結論 **58**

謝 辞 **61**

参考文献 **62**

付録A Block Gram-Schmidt 法 **66**

付録B Parallel Block Gram-Schmidt 法 **85**

付録C Block Symplectic Gram-Schmidt 法 **105**

付録D Scheme E **108**

第 1 章

序論

1.1　背景

理工学分野において，偏微分方程式によって表される様々な問題を有限差分法や有限要素法などを用いて離散化し，得られた大規模方程式の近似解を数値的に求めることが多々ある．その中で Gram-Schmidt 法 (GS 法) による直交化は，今まで様々な研究が行われ，多くの改良手法が提案されている [25]．その応用分野は幅広く，構造工学や電磁流体工学から生じる固有値問題の解法である Lanczos 法 [19, 28]，Arnoldi 法 [2]，Jacobi-Davidson 法 [11, 24, 30] や，大規模線形方程式の解法である GMRES 法 [16, 17, 20, 23] などの Klyrov 部分空間法の一部として使われている．また，Hamiltonian 行列のような特殊構造を持つ行列の固有値を求めることに適している Symplectic 法においても GS 法が用いられており，Symplectic Gram-Schmidt 法 (SGS 法) が提案されている．Lanzcos 法，Arnoldi 法，Jacobi-Davidson 法，GMRES 法は大規模疎行列問題に適した手法であり，偏微分方程式の境界値問題や固有値問題の解法に使われるなど，応用範囲が広く，重要である．

固有値計算は数値計算の分野において非常に重要なトピックの 1 つである．特に近年ビックデータ解析や様々なシミュレーションの大規模化に伴い，大規模な固有値計算手法が必要となっている．その中で，行列 A の全ての固有値を計算するために使われる QR 法は，基本的な固有値計算手法であり，これまでに多くの研究が行われてきた．QR 法の利点は全固有値・固有ベクトルが一度に計算出来ることである．一方，演算コストが非常に高いため大規模な問題に対しては適用しにくい難点があったが，並列化やプロセッサーの高性能化などにより利用可能となってきた．行列 A が密行列の場合，Hauseholder 法を用いて行列 A を Hessemberg 行列か三重対角行列に変換してから QR 法を適用させることにより演算量を大幅に減少させることが出来る．また，行列 A が疎行列の場合は Lanczos 法や Arnoldi 法を用いて行列 A を Hessemberg 行列か，または三重対角行列に変換し，QR

法を実行する．このように GS 法は，現実世界に起こる様々な現象の解析に広く用いられていて，数値計算における最も重要な手法の 1 つであると言える [7, 18, 25, 26, 32]．

GS 法は，正規直交系を生成する算法である．n 次元ベクトル $\boldsymbol{x}$ と，与えられた直交行列 Q に対して，GS 法による直交化は次式のようになる．

$$\boldsymbol{y} = \left(I - QQ^T\right)\boldsymbol{x} = \boldsymbol{x} - QQ^T\boldsymbol{x} \equiv \boldsymbol{x} - Q\boldsymbol{r} \tag{1.1}$$

式 (1.1) により，行列 X の各列を順次直交化すると次式が得られる．

$$X = QR \tag{1.2}$$

ただし，行列 R は上三角行列である．

GS 法は応用範囲の広い重要な手法であり，近年様々な研究が行われている．Stewart [27] は，ベクトルをブロック化した Block Gram-Schmidt 法 (BGS 法) を用いることにより，GS 法と比べ，QR 分解が高速に出来ることを示した．さらに，Stewart [27] は行列 $X \in \mathbb{R}^{n\times n}$ の条件数が大きい場合，列ベクトルをランダムな値から生成したベクトルに置き換える手法を提案した．行列 X の条件数が非常に大きい場合，再直交化を複数回行っても直交ベクトルを生成できないことがある．また最悪の場合は，$\boldsymbol{y} = \boldsymbol{0}$ となりブレイクダウンする．このとき，$\boldsymbol{x}$ を以下の式で置き直すことにより，ブレイクダウンを防ぐことが出来る．

$$\boldsymbol{x} = \frac{\boldsymbol{x}_{\text{new}}}{\|\boldsymbol{x}_{\text{new}}\|}\|\boldsymbol{x}\| \tag{1.3}$$

ただし，$\boldsymbol{x}_{new} \in \mathbb{R}^n$ は一様分布に従うランダム値列ベクトルである．

Vanderstraeten [29] や Katagiri [9] は，GS 法の並列化を提案した．Vanderstraeten [29] は，Modified Gram-Schmidt 法 (MGS 法) と同程度の数値的保証を与える BGS 法の並列化を提案した．また，Katagiri [9] は，GS 法，MGS 法，それらを組み合わせた Hybrid GS 法の並列化とその性能評価を行った．

しかし，BGS 法ではブロックサイズ m をユーザーが適宜決定する必要がある．これは最適な m の値が，扱う行列 X や計算に使うコンピュータによって影響を受けるためである．それゆえ，各問題ごとに最適と考えられるブロックサイズ m をユーザーが決定する必要がある．規模の小さい問題に対しては，このことは問題ではない．しかし大規模な問題に対して BGS 法を適用する場合，これは非効率である．さらに，近年活発に研究が行

われている Parallel Block Gram-Schmidt 法 (PBGS 法) においても同様のことが起きている．

このように様々な研究が行われている GS 法に対して，GS 法と類似している Symplectic Gram-Schmidt 法 (SGS 法) に関しての研究は少ない．SGS 法は，与えられた行列 X に対して $X = SR$ となる Symplectic 行列 S と，上三角行列 R を計算する算法である．SR 分解は，QR 分解に類似した手法で，QR 分解と同様，X の固有値・固有ベクトルを計算する手法に使われている．

応用分野において，特殊な構造を持つ行列の固有値を求めることがある．例えば，最適制御理論から導出される Riccati 方程式の解は，Hamiltonian 行列の固有値・固有ベクトルを求めることで数値的に解くことが出来る [6]．このように，Hamiltonian 行列の固有値の計算は重要であり，現在までにいくつかの研究が行われている [1, 3, 10]．その中で，SR 法は Hamiltonian 行列の構造を保持したまま，計算することが可能なので有用である．これによって，Hamiltonian 行列の共役な固有値対を同時に高速に計算することが可能になる．Van Loan [10] によると，Hamiltonian 行列に対しては，SR 法は QR 法と比べて，計算コスト，記憶領域ともに約 1/4 になることが報告されている．また，大規模疎行列に対しては，Symplectic Lanzcos 法が提案されている．この手法には，Symplectic なベクトル列を生成するために SGS 法が使われている．

現在までに複数の SR 分解の算法が提案されている．そのうちの 1 つが Symplectic Householder 法と Symplectic Givens 回転を用いた SR 分解である [10]．この手法は，Householder 法を用いた QR 分解と似ている算法である．Salam [22] は，この手法が Modified Symplectic Gram-Schmidt 法 (MSGS 法) による SR 分解と数学的に同値であることを証明している．他の SR 分解法は，Classical Symplectic Gram-Schmidt 法 (CSGS 法) や MSGS 法を用いたものである [21]．このように，CGS 法と同様，SGS 法も応用範囲の広い手法である．しかし，CGS 法を用いた QR 分解に関する研究に比べ，SGS 法を用いた SR 分解に関する研究は少ない．

Salam [21] によると，SGS 法はパラメーターの取り方により，J-直交性が大きく崩れることがあることが指摘されている．SGS 法の算法は，直交化されるベクトルの足し算と引き算を行うため桁落ちや丸め誤差の影響を受けやすい．また直交化されたベクトルは正規化をされていない，直交化されたベクトルのノルムが非常に大きくなる可能性がある．これらの事実により，SGS 法では GS 法が生成する直交基底の直交性と比べ，直交

性の崩れが頻繁に起きてしまう可能性がある．

本論文では，BGS 法，PBGS 法と SGS 法に対して，これらの手法の問題点を改善し，効率的に直交化を行うための算法を提案する．

1.2 本論文の目的と構成

本論文では，種々の直交化法を高速化する算法を 3 つ提案する．ただし，それらは根本的に新しい算法ではなく，既存手法の高速化とユーザーがより効率的に直交化法を使用出来るようにするための改良手法の提案である．BGS 法と PBGS 法に共通した問題点の解決と，SGS 法の高速化のための算法を提案する．

- Block Gram-Schmidt 法の適応的なブロックサイズの決定法の提案
- Parallel Block Gram-Schmidt 法の適応的なブロックサイズの決定法の提案
- Symplectic Gram-Schmidt 法のブロック化と，適応的なブロックサイズの決定法の提案

BGS 法において，適切なブロックサイズ m を決定することは非常に重要である．何故なら，ブロックサイズ m によって QR 分解の実行時間が変化するからである．小規模な問題では，計算時間の変化は数秒程度であるが，大規模な問題では大きく変化する．これまで，ブロックサイズ m はユーザーが任意に決定する必要があり，最適な値を調べるためには計算を何度も実行する必要があった．そこで本論文では，BGS 法の適切なブロックサイズ m を問題毎に自動的に決定する算法を提案する．

また BGS 法は並列計算を行うことが可能なので，様々な並列化手法を用いた PBGS 法の提案が行われている．そこで，本論文では適応的にブロックサイズ m を決定する手法を並列化し，PBGS 法に対して適用させた手法を提案する．

SGS 法は，特殊構造を持つ行列の固有値を計算するのに適している手法である．本論文では，SGS 法のブロック化を提案する．GS 法と同様に，ブロック化することによって高速化することが可能である．また，ブロックサイズ m の適応的な決定法を提案し，効率的に SR 分解を行うことが出来る算法を提案する．

2 章では，BGS 法について述べる．特にブロックサイズ m の変化による計算時間の変化と計算量について考察し，ブロックサイズ m の適応的な決定法を提案する．3 章では，

PBGS 法について述べる．BGS 法の並列化手法について述べたあと，ブロックサイズ m の適応的な決定法を並列化し，それを PBGS 法に対して適用する．4 章では，SGS 法について述べる．SGS 法と GS 法の違いを考察し，SGS 法の高速化のためにブロック化を提案する．そして，BGS 法と同様に，ブロックサイズ m の適応的な決定手法を提案する．2 章から 4 章の各章ではそれぞれ，章の最後に数値実験を行い，提案手法の有効性を示す．最後に 5 章において，結論と総括を述べる．

1.3 実験環境

本論文では，以下の 2 つの実験環境で数値実験を行った．

- 実験環境 1
  - OS : Ubuntu12.04.4 LTS
  - CPU : Intel(R) Xeon(R) CPU X5460 3.16GHz
  - Memory : 4GB
  - 精度 : 倍精度
- 実験環境 2
  - OS : Ubuntu14.04 LTS
  - CPU : Intel(R) Xeon(R) CPU E3-1270 V2 3.50GHz
  - Memory : 16GB
  - 精度 : 倍精度

プログラム言語は C 言語と Matlab2013b を用いた．

1.4 数値例

第 2 章から第 4 章における数値実験には，以下に挙げる問題を用いた．

1.4.1 数値例 1

BCSSTK02，06，15，18 は Matrix Market [5] から取得した例題であり，構造工学で現れる問題から生じる実正定値対称疎行列を係数とする固有値問題である．以下にそれぞれの行列サイズと，図 1.1 から図 1.8 に行列の非ゼロ成分とその値をプロットした図を示す．

- BCSSTK02：112×112
- BCSSTK06：420×420
- BCSSTK15：3948×3948
- BCSSTK18：11948×11948

1.4.2 数値例 2

CAVITY03，06，10，19 は Matrix Market [5] から取得した例題であり，cavity-flow から生じる偏微分方程式の境界値問題を離散化して生成される実非対称行列である．以下にそれぞれの行列サイズと，図 1.9 から図 1.16 に行列の非ゼロ成分とその値をプロットした図を示す．

- CAVITY03：317×317
- CAVITY06：1182×1182
- CAVITY10：2597×2597
- CAVITY19：4562×4562

1.4.3 数値例 3

3 つ目のテスト行列として，区間 $[1 : 10]$ の一様分布に従うランダム値からなる Hamiltonian 行列を用いる．行列サイズは，それぞれ $20 \times 20, 40 \times 40, \ldots, 200 \times 200$ の 10 個である．

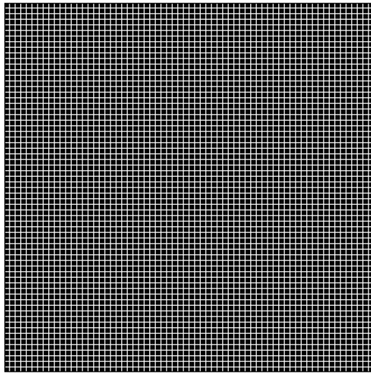


図 1.1　BCSSTK02 の構造パターン

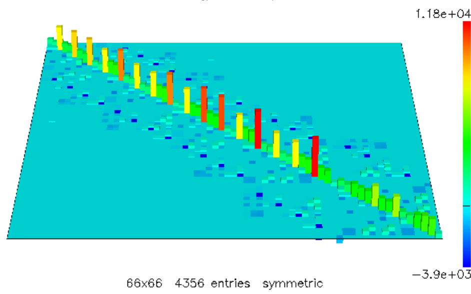


図 1.2　BCSSTK02 の構造パターン

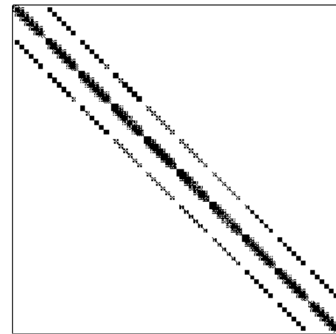


図 1.3　BCSSTK06 の構造パターン

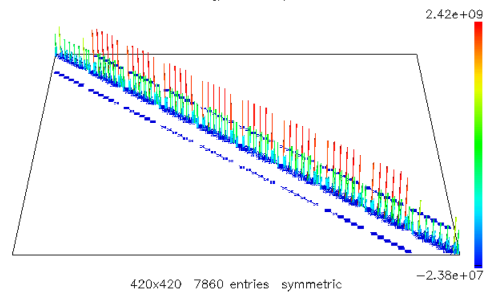


図 1.4　BCSSTK06 の構造パターン

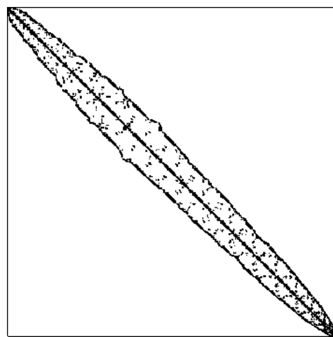


図 1.5　BCSSTK15 の構造パターン

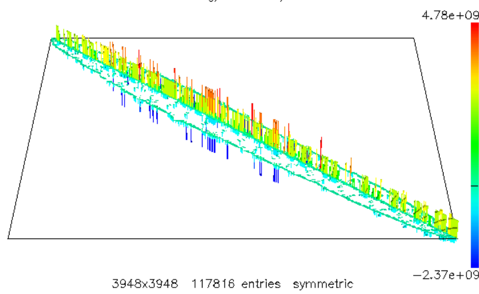


図 1.6　BCSSTK15 の構造パターン

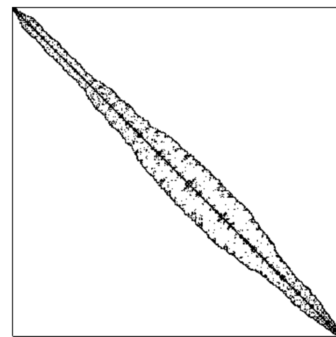


図 1.7　BCSSTK10 の構造パターン

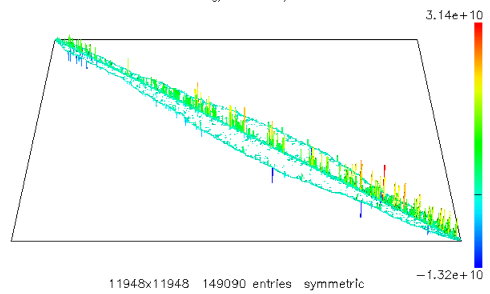


図 1.8　BCSSTK18 の構造パターン

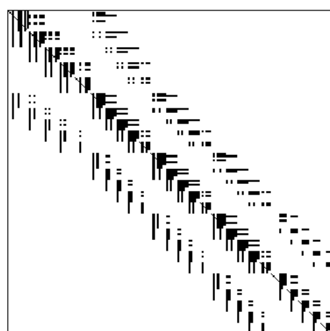


図 1.9　CAVITY03 の構造パターン

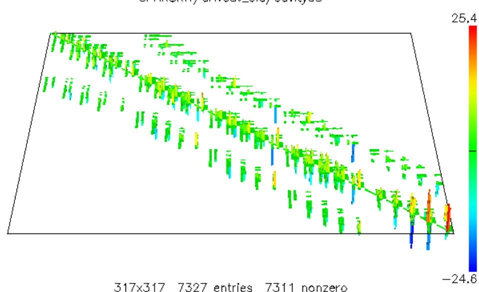


図 1.10　CAVITY03 の構造パターン

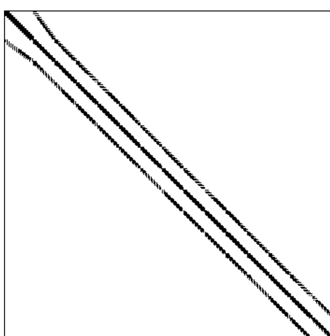


図 1.13　CAVITY10 の構造パターン

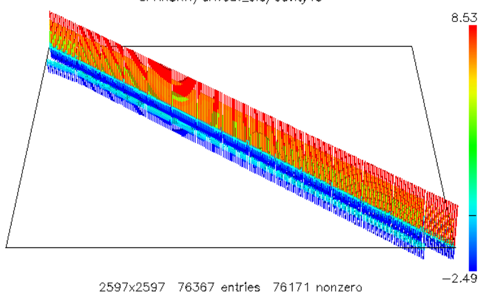


図 1.14　CAVITY10 の構造パターン

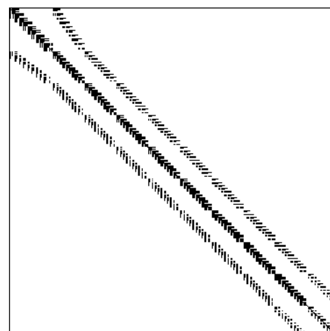


図 1.11　CAVITY06 の構造パターン

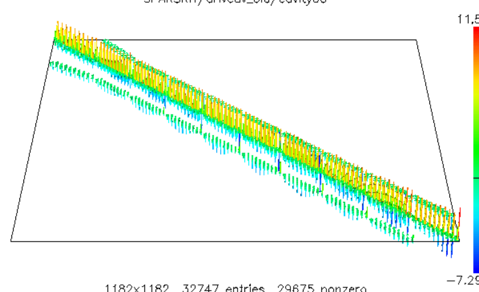


図 1.12　CAVITY06 の構造パターン

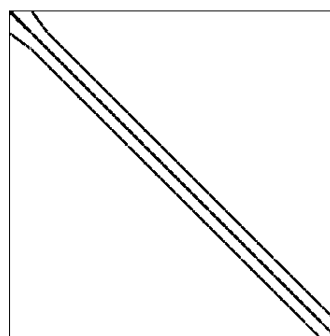


図 1.15　CAVITY19 の構造パターン

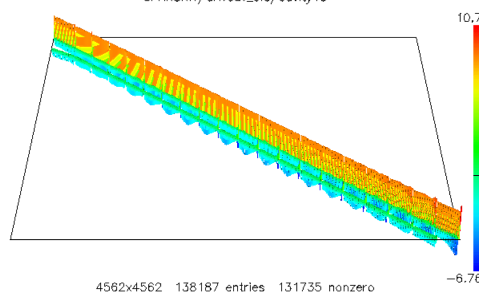


図 1.16　CAVITY19 の構造パターン

1.4.4 数値例 4

4 つ目のテスト行列として，行列サイズが 200×200 と 1000×1000 である，区間 $[1:10]$ の一様分布に従うランダム値からなる Hamiltonian 行列をテスト行列として用いる．それぞれの行列の要素の値をプロットしたものを図 1.17 と図 1.18 に示す．

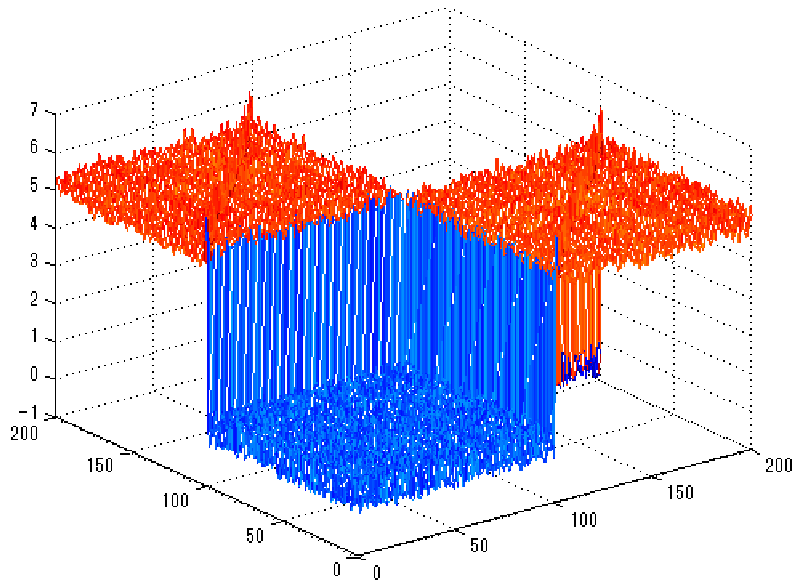


図 1.17　Hamiltonian 行列 1 の構造パターン

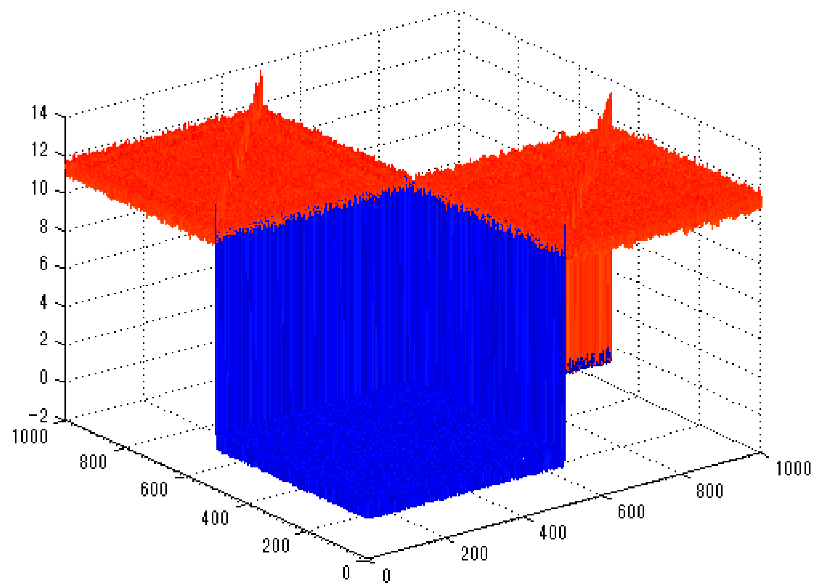


図 1.18　Hamiltonian 行列 2 の構造パターン

第 2 章

Block Gram-Schmidt 法

本章では，Block Gram-Schmidt (BGS) 法と，ブロックサイズ m による計算時間の変化について述べる．そして，適応的にブロックサイズ m を決定する手法を提案する．

2.1 Classical Gram-Schmidt 法

本節では，Classical Gram-Schmidt (CGS) 法について述べる．CGS 法は正規直交基底列を生成する算法である．

2.1.1 算法

今，CGS 法を用いて，1 次独立なベクトル列 $\boldsymbol{x}_i \in \mathbb{R}^n, i = 1, \ldots, n$ を直交化し，正規直交基底列 $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ を求めるとすると，Gram-Schmidt の直交化法より，次式が成立する．

$$\boldsymbol{y}_1 = \frac{\boldsymbol{x}_1}{\|\boldsymbol{x}_1\|} \tag{2.1}$$

$$\hat{\boldsymbol{y}}_2 = \boldsymbol{x}_2 - \boldsymbol{y}_1^{\mathrm{T}}\boldsymbol{x}_2\boldsymbol{y}_1, \quad \boldsymbol{y}_2 = \frac{\hat{\boldsymbol{y}}_2}{\|\hat{\boldsymbol{y}}_2\|} \tag{2.2}$$

$$\hat{\boldsymbol{y}}_3 = \boldsymbol{x}_3 - \boldsymbol{y}_1^{\mathrm{T}}\boldsymbol{x}_3\boldsymbol{y}_1 - \boldsymbol{y}_2^{\mathrm{T}}\boldsymbol{x}_3\boldsymbol{y}_2, \quad \boldsymbol{y}_3 = \frac{\hat{\boldsymbol{y}}_3}{\|\hat{\boldsymbol{y}}_3\|} \tag{2.3}$$

$$\vdots$$

$$\hat{\boldsymbol{y}}_n = \boldsymbol{x}_n - \boldsymbol{y}_1^{\mathrm{T}}\boldsymbol{x}_n\boldsymbol{y}_1 - \cdots - \boldsymbol{y}_{n-1}^{\mathrm{T}}\boldsymbol{x}_n\boldsymbol{y}_{n-1}, \quad \boldsymbol{y}_n = \frac{\hat{\boldsymbol{y}}_n}{\|\hat{\boldsymbol{y}}_n\|} \tag{2.4}$$

ただし，$^{\mathrm{T}}$ は転置を表す．ベクトル列 $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ をまとめて，$Q = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n]$ とすると Q は正規直交行列となる．CGS 法の算法を Algorithm 1 に示す．

ここで，式 (2.2) から式 (2.4) において，CGS 法の $k\,(k = 1, \ldots, n)$ ステップ目の計算

Algorithm 1 Classical Gram-Schmidt Algorithm

Require: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$
Ensure: Q, R
  $\boldsymbol{y}_1 = \boldsymbol{x}_1 / \|\boldsymbol{x}_1\|$
  $R(1,1) = \|\boldsymbol{x}_1\|$
  for $k = 2 : n$ **do**
    $\hat{\boldsymbol{y}}_k = \boldsymbol{x}_k - \boldsymbol{y}_1^{\mathrm{T}} \boldsymbol{x}_k \boldsymbol{y}_1 - \cdots - \boldsymbol{y}_{k-1}^{\mathrm{T}} \boldsymbol{x}_k \boldsymbol{y}_{k-1}$
    $R(1:k-1, k) = (\boldsymbol{y}_1^{\mathrm{T}} \boldsymbol{x}_k, \cdots, \boldsymbol{y}_{k-1}^{\mathrm{T}} \boldsymbol{x}_k)$
    $\boldsymbol{y}_k = \hat{\boldsymbol{y}}_k / \|\hat{\boldsymbol{y}}_k$
    $R(k,k) = \|\hat{\boldsymbol{y}}_k\|$
  end for
  $Q = [\boldsymbol{y}_1, \cdots, \boldsymbol{y}_n]$

の行列表記は，次式のようになる．

$$\hat{\boldsymbol{y}}_k = \left(I - Q_{k-1} Q_{k-1}^{\mathrm{T}}\right) \boldsymbol{x}_k = \boldsymbol{x}_k - Q_{k-1} Q_{k-1}^{\mathrm{T}} \boldsymbol{x}_k \equiv \boldsymbol{x}_k - Q_{k-1} \boldsymbol{r}_k, \quad \boldsymbol{y}_k = \frac{\hat{\boldsymbol{y}}_k}{\|\hat{\boldsymbol{y}}_k\|} \tag{2.5}$$

ただし，$Q_{k-1} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{k-1}]$ である．以上より，与えられた正規直交行列 $Q_{k-1} \in \mathbb{R}^{n \times k-1}$ に対して，CGS 法によるベクトル $\boldsymbol{x}_k \in \mathbb{R}^n$ の直交化のステップは，次式で表すことができる．

$$\boldsymbol{r}_k = Q_{k-1}^{\mathrm{T}} \boldsymbol{x}_k \tag{2.6}$$

$$\hat{\boldsymbol{y}}_k = \boldsymbol{x}_k - Q_{k-1} \boldsymbol{r}_k \tag{2.7}$$

$$\boldsymbol{y}_k = \frac{\hat{\boldsymbol{y}}_k}{\|\hat{\boldsymbol{y}}_k\|} \tag{2.8}$$

ここで，CGS 法の行列表記による算法を Algorithm 2 に示す．これによって，正規直交行列 Q_{k-1} に直交化したベクトル $\boldsymbol{y}_k$ を生成することが可能となる．

2.1.2 再直交化

本節では，GS 法の再直交化について述べる．GS 法において，与えられた正規直交行列 Q に対してベクトル $\boldsymbol{x}$ を直交化をする場合，直交化ベクトル $\boldsymbol{y}$ を正確に求められない場合がある．このような場合には，再直交化を行う．

Algorithm 2 Classical Gram-Schmidt Algorithm

Require: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$
Ensure: Q, R
  $Q_1 = \boldsymbol{x}_1/\|\boldsymbol{x}_1\|$
  $R(1,1) = \|\boldsymbol{x}_1\|$
  for $k = 2 : n$ **do**
    $\boldsymbol{r}_k = Q_{k-1}^{\mathrm{T}} \boldsymbol{x}_k$
    $R(1:k-1, k) = \boldsymbol{r}_k$
    $\hat{\boldsymbol{y}}_k = \boldsymbol{x}_k - Q_{k-1}\boldsymbol{r}_k$
    $\boldsymbol{y}_k = \hat{\boldsymbol{y}}_k/\|\hat{\boldsymbol{y}}_k\|$
    $R(k,k) = \|\hat{\boldsymbol{y}}_k\|$
    $Q_k = [Q_{k-1}, \boldsymbol{y}_k]$
  end for

まず，GS 法の再直交化の精度について述べる．今，ベクトル $\boldsymbol{x}_k \in \mathbb{R}^n$ と正規直交行列 $Q_{k-1} \in \mathbb{R}^{n\times k-1}$ が与えられていると仮定する．ベクトル $\boldsymbol{x}_k$ を以下のように分解する．

$$\boldsymbol{x}_k = \boldsymbol{x}_{k_{Q_{k-1}}} + \boldsymbol{x}_{k_\perp} \tag{2.9}$$

ただし，$\boldsymbol{x}_{k_{Q_{k-1}}}$ は $\boldsymbol{x}_k$ の Q_{k-1} 方向の成分，$\boldsymbol{x}_{k_\perp}$ は Q_{k-1} に直交する成分である．すなわち，$\boldsymbol{x}_{k_{Q_{k-1}}}$ は $\boldsymbol{x}_k$ の直交補空間方向成分であり，$\boldsymbol{x}_{k_\perp}$ は直交化方向の成分である．さらに，

$$\sigma = \frac{\|\boldsymbol{x}_{k_{Q_{k-1}}}\|}{\|\boldsymbol{x}_k\|}, \quad \gamma = \frac{\|\boldsymbol{x}_{k_{\perp}\|}}{\|\boldsymbol{x}_k\|} \tag{2.10}$$

とすると，正規直交行列 Q_{k-1} によって直交化されたベクトル $\hat{\boldsymbol{y}}_k$ は以下のように表すことができる．

$$\hat{\boldsymbol{y}}_k = \boldsymbol{x}_{k_\perp} + \boldsymbol{e}, \quad \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{x}_k\|} \equiv \varepsilon \tag{2.11}$$

ただし，ε は十分小さい数である．式 (2.11) より

$$\|\hat{\boldsymbol{y}}_k\| = \|\boldsymbol{x}_\perp + \boldsymbol{e}\| \geq \|\boldsymbol{x}_\perp\| - \|\boldsymbol{e}\| \tag{2.12}$$

が成り立つ．また，$\hat{\boldsymbol{y}}_k$ を $\boldsymbol{x}_k$ と同様に次のように分解することができる．

$$\hat{\boldsymbol{y}}_k = \hat{\boldsymbol{y}}_{k_{Q_{k-1}}} + \hat{\boldsymbol{y}}_{k_\perp}, \quad \|\hat{\boldsymbol{y}}_{k_{Q_{k-1}}}\| = \|\boldsymbol{x}_\perp\|, \quad \|\hat{\boldsymbol{y}}_{k_\perp}\| = \|\boldsymbol{e}\| \tag{2.13}$$

$\hat{\boldsymbol{y}}$ に対して $\hat{\sigma}, \hat{\gamma}$ を σ, γ と同様に定義すると，次式が成立する．

$$\hat{\sigma} = \frac{\|\hat{\boldsymbol{y}}_{k_{Q_{k-1}}}\|}{\|\hat{\boldsymbol{y}}_k\|} \leq \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{x}_\perp\| - \|\boldsymbol{e}\|} \leq \frac{\varepsilon}{\gamma - \varepsilon} = \frac{\varepsilon}{\sqrt{1-\sigma^2} - \varepsilon} \tag{2.14}$$

例えば，$\gamma \leq 3\varepsilon$ の時，2 回 GS 法を適用した後の Q 方向の成分 $\tilde{\sigma}$ は，式 (2.14) より次のようになる．

$$\tilde{\sigma} \leq \sqrt{\frac{4}{3}}\varepsilon \simeq 1.2\varepsilon \tag{2.15}$$

ただし，ε の 2 次の項は無視するものとする．式 (2.15) より再直交化が有効であることがわかる．さらに，次の式が成り立つ場合を考える．

$$\|\hat{\boldsymbol{y}}_k\| \geq \frac{1}{2}\|\boldsymbol{x}_k\| \tag{2.16}$$

ここで，式 (2.16) より，次式が成立する．

$$\gamma \geq \frac{1}{2} - \varepsilon \tag{2.17}$$

さらに，式 (2.14) を使うと，次式が得られる．

$$\hat{\sigma} \leq 2\varepsilon \tag{2.18}$$

ただし，ε の 2 次の項は無視するものとする．これより，式 (2.16) を満たせば，$\hat{\boldsymbol{y}}_k$ が直交化されていることがわかる．逆に，式 (2.16) をベクトル x の再直交化が必要かどうかを判断するための条件と考えることが出来る．そこで，もし式 (2.16) が成り立つならば，$\|\hat{\boldsymbol{y}}_k\|$ の直交性は保証されることになる．もし成り立たないのであれば，再び GS 法を用いて再直交化をする．この手順を繰り返すことによって，直交行列を正確に生成することが出来る．

2.2 Block Gram-Schmidt 法

次に BGS 法について述べる．

2.2.1 算法

BGS 法は，正則な行列 $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{n \times n}$ を列方向に m 分割し，分割して出来た行列 $X_{\mathrm{block}} \in \mathbb{R}^{n \times m}$ ごとに直交化する手法である．ただし，簡単のために m は n を割り切る任意の自然数とする．与えられた直交行列 $Q \in \mathbb{R}^{n \times h}$ を用いた，BGS 法による直交化のステップは次のようになる．ただし，h は任意の自然数とする．

$$R_{12} = Q^{\mathrm{T}} X_{\mathrm{block}} \tag{2.19}$$

$$\widehat{Y} = X_{\mathrm{block}} - Q R_{12} \tag{2.20}$$

これによって，行列 Q に対して直交化した行列 $\widehat{Y} \in \mathbb{R}^{n \times m}$ を計算することができる．しかし，式 (2.19) と式 (2.20) だけでは行列 $\widehat{Y}$ の列ベクトルどうしは直交していない．そこで今，$\widehat{Y}$，Y の $k\,(k = 1, \ldots, m-1)$ 列目までを $\widehat{Y}_{1:k}, Y_{1:k}$ とし，$k+1$ 列目の列ベクトルを $\hat{\boldsymbol{y}}_{k+1}, \boldsymbol{y}_{k+1}$ とする．ここで，CGS 法を用いると，次式が成立する．

$$\boldsymbol{r}_{k+1} = Y_{1:k}^{T} \hat{\boldsymbol{y}}_{k+1}, \quad \boldsymbol{y}_{k+1} = Y_{1:k} \boldsymbol{r}_{k+1} \tag{2.21}$$

以下これを繰り返すと，次のように $\hat{Y}$ を QR 分解することができる．

$$\widehat{Y} \rightarrow Y R_{22} \tag{2.22}$$

ただし，Y は正規直交行列，R_{22} は上三角行列である．ここで，式 (2.20) と式 (2.22) をまとめると，X_{block} は次式を満たす．

$$X_{\mathrm{block}} = Q R_{12} + Y R_{22} \tag{2.23}$$

Stewart [27] は，式 (2.23) を用いることにより，計算過程において，直交行列 Q を使う回数を $1/m$ に抑えることが出来ることを示した．また，BGS 法においては行列積を計算する必要が生じるが，Basic Liner Algebra Subprogram (BLAS) などの数値計算ライブラリを使うことによって，行列積を高速に計算出来ることが報告されている [31]．

2.2.2 再直交化

GS 法と同様に BGS 法においても，直交行列を正確に求められない場合がある．そのような場合には，GS 法と同様に再直交化を行う．BGS 法において，X_{block} は式 (2.23) で表すことができた．

$$X_{\mathrm{block}} = QR_{12} + YR_{22}$$

ここで，さらに Y の再直交化を行うと，行列 S_{12}，S_{22}，Z に対して，同様に次式が成り立つ．

$$Y = QS_{12} + ZS_{22} \tag{2.24}$$

ただし，S_{22} は上三角行列である．次に，式 (2.23) と式 (2.24) をまとめると，次式が成り立つ．

$$X_{\mathrm{block}} = Q\left(R_{12} + S_{12}R_{22}\right) + ZS_{22}R_{22} \tag{2.25}$$

この再直交化により，BGS 法においても，直交行列を正確に求められる．また 2.2.1 節で述べたように，BGS 法は式 (2.21) で GS 法を用いる．よって，式 (2.21) で得られた $\boldsymbol{y}_{k+1}$ に対して，2.1.2 節で述べた GS 法の再直交化条件 (2.16) を適用することで，BGS 法の再直交化を適切に行うことが出来る．これまで述べてきた BGS 法の算法を Algorithm 3 に示す．

2.3 ブロックサイズ

2.3.1 計算量

本節では，正則行列 X を BGS 法により直交化し，$X = QR$ を計算することにかかる計算量について考える．ただし，ブロックサイズ m は n を割り切る値とする．また全ての列に対し再直交化すると仮定する．掛け算の計算 1 回を 1 ユニットとする．$Q \in \mathbb{R}^{n\times h}$ と仮定し，$k+1$ ステップ目の BGS 法の計算量を考える．ただし，$h = km$ とする．まず，式 (2.19)，式 (2.20) の直交化の計算の部分が次のようになる．

$$\left(km^2n + km^2n\right) \times 2 = 4nm^2k \tag{2.26}$$

Algorithm 3 Block Gram-Schmidt Algorithm

Require: $X \in \mathbb{R}^{n\times n}, X_{\text{block}} \in \mathbb{R}^{n\times m}$
Ensure: Q, R
  $K := P/m$
  for $k = 0 : K-1$ **do**
    $X_{\text{block}} := X[:, km : (k+1)m]$
    $R_{12} := Q^{\mathrm{T}} X_{\text{block}}$
    $\widehat{Y} = X_{\text{block}} - QR_{12}$
    for $l = 1 : m$ **do**
      $\boldsymbol{r} := Y_{1:l-1}^{\mathrm{T}} \hat{\boldsymbol{y}}_l$
      $\boldsymbol{y} = Y_{1:(l-1)} \boldsymbol{r}$
      $R_{22}[1 : l-1, l] = \boldsymbol{r}$
      $R_{22}[k, k] = \|\boldsymbol{y}\|$
      $Y[:, k] = \boldsymbol{y}/\|\boldsymbol{y}\|$
    end for
    if $(\hat{\boldsymbol{y}}_k < 1/2\,\|\boldsymbol{x}_k\|)$ **then**
      Reorthogonalization
      $S_{12} := Q^{\mathrm{T}} \widehat{Y}$
      $\widehat{Y} = \widehat{Y} - QS_{12}$
      for l=1:m **do**
        $\boldsymbol{r} := Y_{1:(l-1)}^{\mathrm{T}} \hat{\boldsymbol{y}}_l$
        $\boldsymbol{y} = Y_{1:(l-1)} \boldsymbol{r}$
        $S_{22}[1 : l-1, l] = \boldsymbol{r}$
        $S_{22}[l, l] = \|\boldsymbol{y}\|$
        $Y[:, l] = \boldsymbol{y}/\|\boldsymbol{y}\|$
      end for
    end if
    $Q[:, km : (k+1)m] = Y$
    $R_{12} = S_{12} R_{22} + R_{12}$
    $R_{22} = S_{22} R_{22}$
    $R = R_{12} + R_{22}$
  end for

次に，$\widehat{Y}$ を直交化する計算の部分が，以下のようになる．

$$\sum_{k=1}^{m-1} (nk + kn) \times 2 = 2nm\,(m-1) \tag{2.27}$$

さらに，R_{12}，R_{22}，S_{12}，S_{22} から R を作るのにかかる計算量は，以下のようになる．

$$m^3 + km^3 \tag{2.28}$$

よって，式 (2.26)，式 (2.27)，式 (2.28) の計算量をまとめると，1 ステップあたりの BGS 法の計算量は以下のようになる．

$$4nm^2h + 2nm(m-1) + (k+1)m^3 \tag{2.29}$$

これより BGS 法全体の計算量 C_{bgs} は，次式で与えることが出来る．

$$C_{bgs} = \sum_{k=1}^{n/m-1} (4nm^2k + (k+1)m^3) \tag{2.30}$$

$$+2nm(m-1)\frac{n}{m}$$

$$= -m^3 + \frac{1}{2}nm^2 + \frac{1}{2}n^2m + 2n^3 - 2n^2 \tag{2.31}$$

ただし，簡単のために m は n を割り切る値としているので，n/m は自然数である．また，同様の仮定において GS 法全体の計算量 C_{gs} は，以下のようになる．

$$C_{gs} = 2n^2\,(n-1) \tag{2.32}$$

故に，次式が成り立つ．

$$C_{bgs} - C_{gs} = -m^3 + \frac{1}{2}nm(n+m) \tag{2.33}$$

式 (2.33) より，BGS 法を行うと，計算量が増加することがわかる．ここで，式 (2.33) は，R_{12}，R_{22}，S_{12}，S_{22} から R を生成する部分の計算量の和であることに注意する．従って，

もし 1 度も再直交化しないのであれば，次式が成り立つ．

$$C_{bgs} = C_{gs}$$

よって $m \geq 2$ のとき，1 度でも再直交を行うと BGS 法は，コスト高になると考えられる．しかし，実際には BGS 法を用いることで計算を高速に行うことが出来る．これは，先ほども述べたように，X をブロック化することによって，Q を読み込む回数を減らし，読み込むためのコストを減らすことが出来るからである．また，BLAS 等の数値計算ライブラリを使うことで，効率的に計算出来るからである．

一般に，ブロックサイズ m の値を大きくするにつれ計算時間は減少するが，ある値を境に計算時間は増加する．これは m が大きくなることで，パフォーマンスが向上する一方，計算量が大きくなるからである．

2.3.2 計算時間

この節では，BGS 法の実行に必要な計算時間について述べる．BGS 法の h ステップ目の計算量は，式 (2.3.2) で表すことができた．

$$4nm^2h + 2nm(m-1) + (k+1)m^3$$

ここで，式 (2.3.2) は，h の 1 次関数である．この式より，ブロックサイズ m が固定された場合，BGS 法の任意のステップにおいて BGS 法の計算量を求めることができる．また，式 (2.3.2) は，h の 1 次関数であるので，BGS 法の計算量は線形に増加する．以上より，BGS 法を実行した際の計算時間は，線形に増加すると予想できる．実際，BGS 法を実行した際の計算時間の変化を，図 2.1 を示す．

図 2.1 は，1.4.2 節で述べた数値例 2 の CAVITY19 であるサイズが 4562×4562 の実対称行列を，ブロックサイズ $m = 50$ で BGS 法により QR 分解したときに要する時間を，100 列ごとに計測した結果である．この図から，計算時間は線形に増加していることがわかる．

次に，図 2.2 にブロックサイズ m の変化による計算時間の変化を示す．図 2.2 より，BGS 法は，ブロックサイズ m によって計算時間が大きく変わることがわかる．

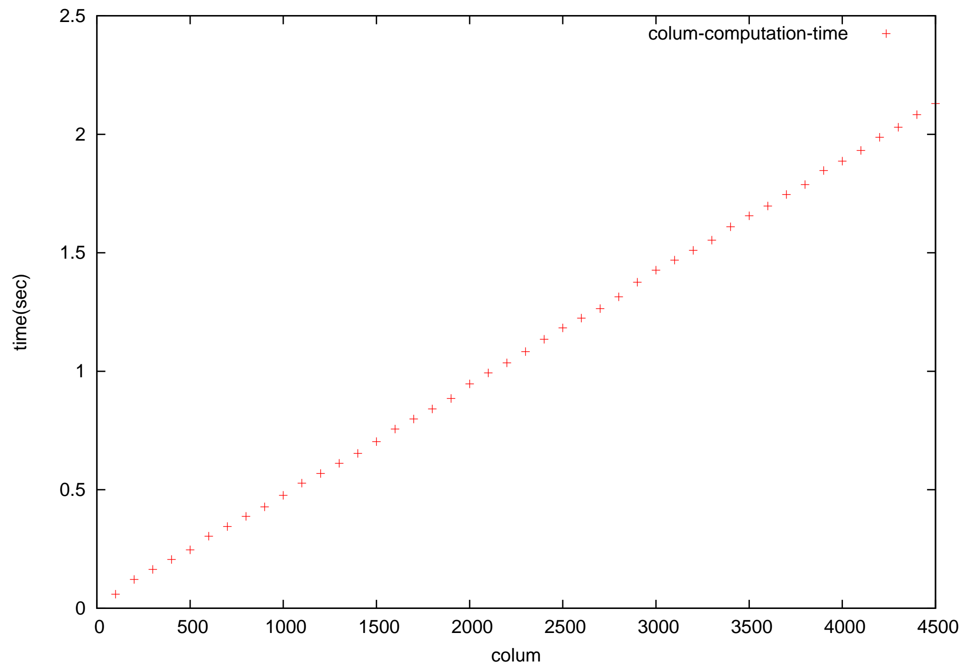


図 2.1　直交化と計算時間の関係

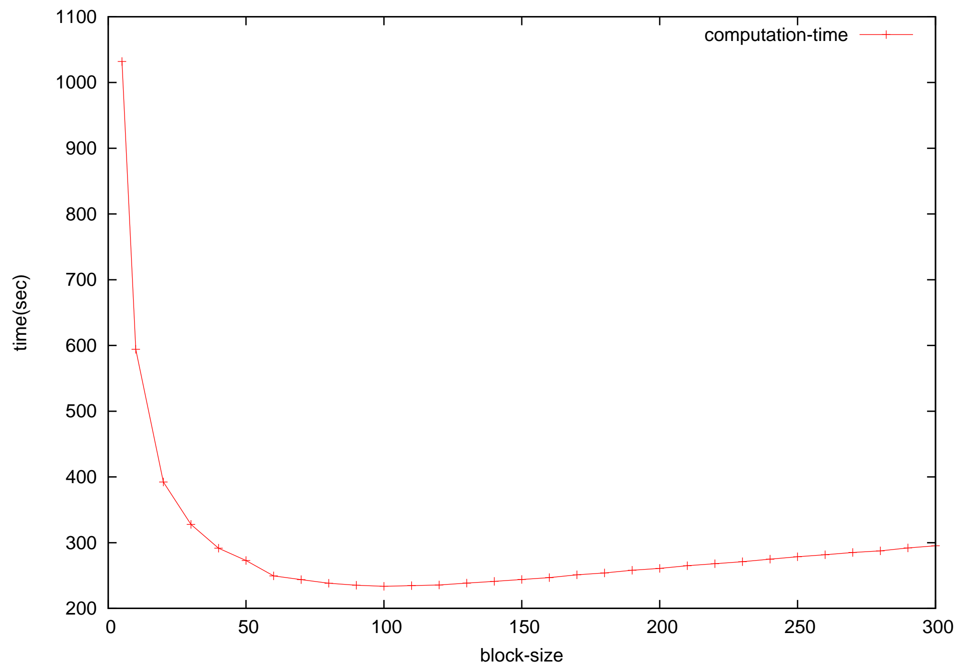


図 2.2　m の変化による計算時間の変化

2.4 ブロックサイズの最適化

BGS 法において最適なブロックサイズ m は扱う問題によって異なる．最適なブロックサイズ m を求めるためには，いくつかのブロックサイズで BGS 法を行い，最も良い m の値を決定する必要がある．規模の小さい問題においては，このことは大きな問題ではない．しかし，1 回の計算に長時間を要する問題においては，この手法は現実的ではない．そこで，計算量と計算時間の関係に着目し，適切なブロックサイズ m を BGS 法の中で適応的に決定する方法を 3 つ提案する [14, 12].

2.4.1 Scheme A

1 回の掛け算にかかる時間を比較することで，最適なブロックサイズを適応的に決定する方法を考える．2.3 節より，h 列目における BGS 法の計算量は次式であった．

$$4nmh + 2nm(m-1) + (h+m)m^2$$

h 列目におけるブロックサイズ m_i での BGS 法の 1 ステップの計算時間を t_i とすると，1 計算量あたりの計算時間 c_i は，次式のようになる．

$$c_i = \frac{t_i}{4nm_ih + 2nm_i(m_i-1) + (h+m_i)m_i^2} \tag{2.34}$$

同様に，$t_i, c_i, i = 1, \ldots, j$ を考え，次式を満たす m_i を最適なブロックサイズ m とする．ただし，j は任意の正の数であり，ユーザーが適宜決定する．

$$m = \{m_i; \min_i c_i\} \tag{2.35}$$

以上をまとめた Scheme A の算法を Algorithm 4 に示す．本論文において，Scheme A の m_i の初期値は $m_1 = 1$ とし，$m_i = 2^{i-1}$ とする．i の値を 1 から $j = 10$ まで行い BGS 法の 1 ステップを行う．また，式 (2.34) を用いて計算した c_i の中で，最も小さい c_i を求める手法として，次式を用いる．

$$\begin{cases} c_{i+1} > c_i \\ c_i < c_{i-1} \end{cases} \tag{2.36}$$

Algorithm 4 Scheme A

Require: $X \in \mathbb{R}^{n \times n}$
Ensure: m
  for $i = 1 : 10$ **do**
    $m_i := 2^{i-1}$
    $start := gettimeofday()$
    Block Gram-Schmidt method
    $end := gettimeofday()$
    $t_i := end - start$
    $s_i := t_i/(4nm_ih + 2nm_i(m_i - 1) + (h + m_i)m_i^2)$
    if $s_i \geq s_{i-1}$ **then**
      break
    end if
  end for

これは BGS 法において，前節でも述べたように，ブロックサイズ m が大きくなることで，パフォーマンスが向上する一方，計算量が大きくなるためである．よって，ブロックサイズ m の値が大きくなるにつれ計算時間が短くなり，あるサイズより大きくなると計算時間が増える．図 2.2 より，実際にあるブロックサイズ m を境に計算時間が減少から増加に転じている．よって，式 (2.36) を満たすような c_i を与える m_i を適切なブロックサイズ m とする．

2.4.2 Scheme B

次に，ブロックサイズの変化による計算時間の変化を予測し，計算時間の変化を関数で近似することで，最適なブロックサイズ m を自動的に決定する方法を考える．

まず式 (2.34) より，1 回の演算あたりの計算時間を求めることが出来る．また，2.3.2 節で述べたように，BGS 法全体の計算量は，式 (2.31) で与えることが出来る．これより，全体の予想される計算時間は，次式でのようになる．

$$t_i = C_{bgs_{m_i}} \times c_i \tag{2.37}$$

ここで，次式を考える．

$$
\begin{aligned}
f(x) &= ax^4 - bx^2 + c \\
&= a\left(x^2 - \frac{b}{2a}\right)^2 - \frac{b^2}{4a^2} + c
\end{aligned}
\tag{2.38}
$$

この関数 $f(x)$ の a, b, c を適当に定めることによって，関数 $f(x)$ を BGS 法のブロックサイズ m の変化による計算時間の変化を近似する関数とすることを考える．まず，式 (2.38) の a, b, c を用いて，以下のように行列 A，ベクトル $\boldsymbol{w}, \boldsymbol{r}$ を定める．

$$
\begin{aligned}
A[i,j] &= m_i^{6-j*2} \\
\boldsymbol{w} &= (a, b, c)^{\mathrm{T}} \\
\boldsymbol{r} &= (t_1, t_2, t_3)^T
\end{aligned}
\tag{2.39}
$$

そして，$A, \boldsymbol{w}, \boldsymbol{r}$ を用いて表された方程式を解く．

$$
\boldsymbol{u} = \min_{\boldsymbol{w} \in \mathbb{R}^3} \|A\boldsymbol{w} - \boldsymbol{r}\| \tag{2.40}
$$

関数 $f(x)$ の係数 a, b, c を，式 (2.40) より得られる $\boldsymbol{u} = (a, b, c)^{\mathrm{T}}$ で定めると，$f(x)$ は BGS 法のブロックサイズ m の変化による計算時間の変化を近似する関数となる．そして，関数 $f(x)$ の最小値をとる x の値を最適なブロックサイズ m とする．式 (2.38) より，関数 $f(x)$ の最小値を取る m は次式となる．

$$
m = \frac{b}{2a} \tag{2.41}
$$

この算法を Algorithm 5 に示す．

2.4.3 Scheme C

Matsuo ら [12] は，2.4.1 節，2.4.2 節のように，計算量から計算時間を予測する手法を提案した．しかし，m において 1 ステップしか実行していないので，計算時間は計測出来るが，h 列目から $h+m$ 列目になるときの計算時間の増加量が判別できなかった．また，計算量の変化から全体の計算時間を予測していたため，適切なものとは大きくずれたブ

Algorithm 5 Scheme B

Require: $X \in \mathbb{R}^{n \times n}$
Ensure: m
  for $i = 1 : 3$ **do**
    $m_i := 2^{i-1}$
    $start := gettimeofday()$
    Block Gram-Schmidt method
    $end := gettimeofday()$
    $t_i := end - start$
  end for
  $a = (t_{i0} - t_{i1})/m_i$
  $r[i] := (1/2)\, n^2 a + t_{i0} - a(h - m)$
  for $j = 5 : 1$ **do**
    $A[i, j] = m_i^{j-1}$
  end for
  solve $\boldsymbol{u} = \min_{\boldsymbol{w} \in \mathbb{R}^3} \|A\boldsymbol{w} - \boldsymbol{r}\|$
  $m := b/2a$

ロックサイズを決定することがしばしばあった．以上の事柄を改善するために，次のように適切なブロックサイズを決定する手法を提案する．m_i において，h 列目から $h + m_i$ 列目になるときの計算時間の増加量を l_i とすると，l_i は次式で表せる．

$$l_i = t_{i,(h+m_i)} - t_{i,h} \tag{2.42}$$

ただし，$t_{i,(h+m)}, t_{i,h}$ はそれぞれ，BGS 法の 1 ステップにかかる計算時間である．すると，1 列あたりの計算時間の増加量 a は，次式となる．

$$a = \frac{l_i}{m_i} \tag{2.43}$$

ここで，図 2.1 より，計算時間の増加量は線形であるから，ブロックサイズ m_i での，全体の計算時間 T_i は次式のように表すことが出来る．

$$T_i = \frac{1}{2} a n^2 + t_{i,h} - ah \tag{2.44}$$

式 (2.44) により，Scheme A と Scheme B よりも正確に全体の計算時間が予測出来る．得られた T_i を使って，Scheme B と同様に，以下のように最適なブロックサイズを決定することになる．

Step1 最初に，5 つのブロックサイズ $m_i, i = 1, 2, \ldots, 5$ を用意する．

Step2 m_i で BGS 法を 2 ステップ実行し計算時間の増加量 t_i を計る．

Step3 $m = m_i$ の全体の計算時間 T_i を t_i から予測する．

Step4 ブロックサイズ m の変化による計算時間の変化を，5 つの T_i から 4 次関数で近似する．

Step5 4 次関数の最小値をとるブロックサイズを適切なブロックサイズとする．

この手法を用いることで，QR 分解の実行時間に影響を与えることなく，適切なブロックサイズを見つけることが出来る．m_i を小さい値にすれば，Step2 における 2 ステップの計算時間は短く，増加量も正確にわかる．よって問題が大規模な場合，計算時間全体に対する予測に必要な時間はとても小さい．この算法を Algorithm 6 に示す．

2.5 数値実験

本節では，BGS 法の最適なブロックサイズ m_{opt} を決定する 3 つの提案手法の有用性を示すために，実験環境 1 のもとで C 言語を用いて数値実験を行った．本節において，数値実験の結果を示す表に使われる記号は，それぞれ次で表す．

- BGS : Block Gram-Schmidt 法
- m_{opt} : BGS 法のブロックサイズ $m, m = 10, 20, \ldots, 300,$ の中で最適なブロックサイズ
- m_{A} : 2.4.1 節の Scheme A によって決められたブロックサイズ
- m_{B} : 2.4.2 節の Scheme B によって決められたブロックサイズ
- m_{C} : 2.4.3 節の Scheme C によって決められたブロックサイズ
- t : 計算時間（秒）

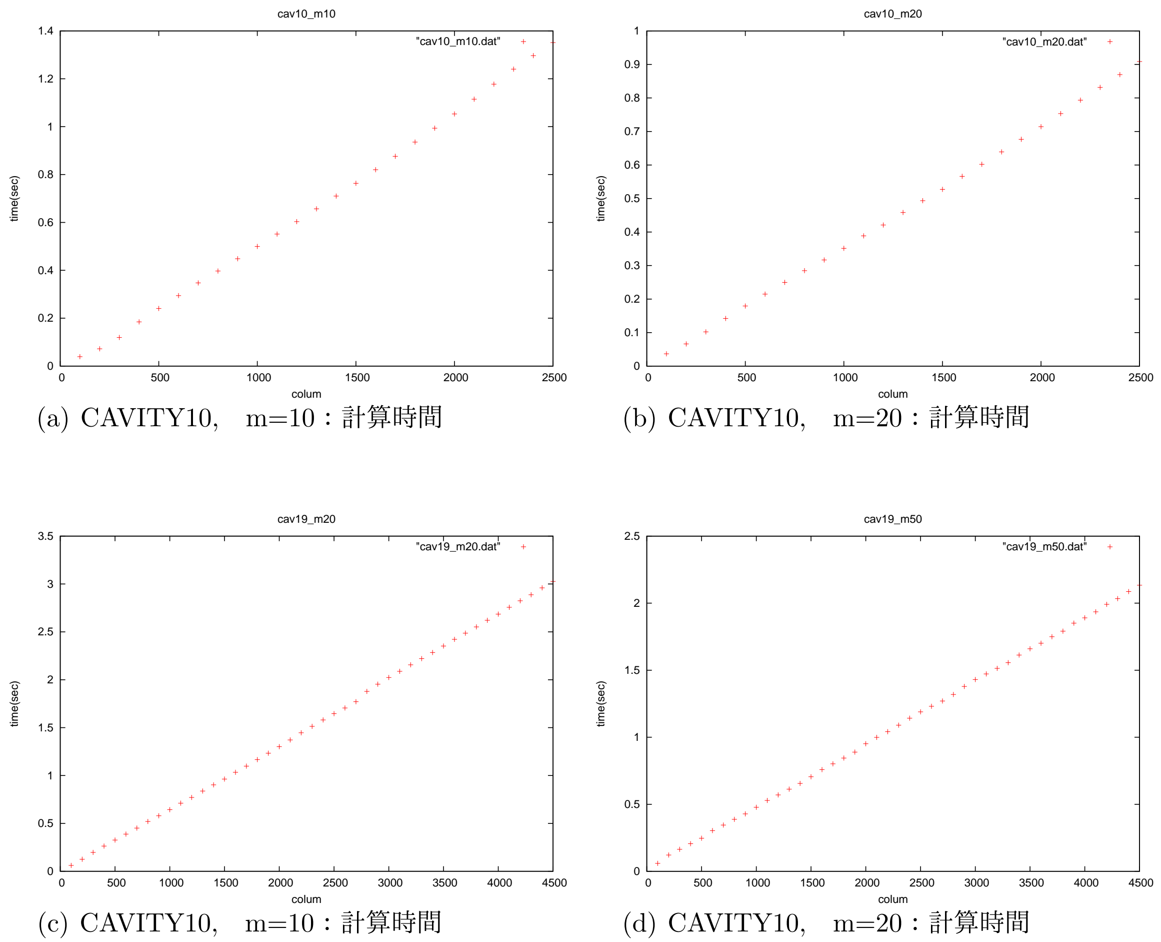
Algorithm 6 Scheme C

Require: :$X \in \mathbb{R}^{n\times n}$
Ensure: m
  for $i = 1:5$ **do**
    $m_i := 2^{i-1}$
    for $j = 0:1$ **do**
      $start := gettimeofday()$
      Block Gram-Schmidt method
      $end := gettimeofday()$
      $t_{ij} := end - start$
    end for
    $a = (t_{i0} - t_{i1})/m_i$
    $b[i] := (1/2)\, n^2 a + t_{i0} - a(h-m)$
    for $j = 5:1$ **do**
      $A[i,j] = m_i^{j-1}$
    end for
  end for
  solve $A\boldsymbol{x} = \boldsymbol{b}$
  $f(m) := x_1 m^4 + x_2 m^3 + x_3 m^2 + x_4 m + x_5$
  solve $m := \min_{m\in[0,\frac{1}{2}n]} f(m)$

2.5.1 数値実験 1

本節では 2.3.2 節で述べた，計算時間が一定に増加することを数値実験を確認する．1.4.2 節においては，数値例 2 の CAVITY19 のみの数値結果を述べた．そこで本節では他の問題に対しても，同様になることを示す．数値例 2 の CAVITY10 の $m = 10$，$m = 20$ の場合と，CAVITY19 の $m = 20$，$m = 50$ の場合の 100 列ごとの計算時間を測った結果を図 2.3 に示す． この図 2.3 からわかるように，それぞれの問題，ブロックサイズにおいて計算時間は 1 次関数的に増えている．つまり，問題のサイズ，ブロックサイズによらず計算時間の増加量は一定で，式 (2.3.2) の h に比例していると考えられる．

以上より，2.3.2 節で述べたように，計算時間が一定に増加することを示した．



(a) CAVITY10,　m=10：計算時間　(b) CAVITY10,　m=20：計算時間

(c) CAVITY10,　m=10：計算時間　(d) CAVITY10,　m=20：計算時間

図 2.3　100 列ごとの計算時間

2.5.2　数値実験 2

本節では，1.4.1 節の数値例 1 を用いて，BGS 法と Scheme A，Scheme B，Scheme C のそれぞれの手法で，これを QR 分解する際のブロックサイズと計算時間を比較した．結果を表 2.1 に示す．表 2.1 より，Scheme A は問題が大きくなるにつれて，m_{A} の値が非常に大きくなっている．逆に，Scheme B は問題が変化しても m_{B} の値はほぼ変化していない．Scheme C の m_{C} の値もあまり変化はないように見える．しかし，m と $m_{\mathrm{A}}, m_{\mathrm{B}}, m_{\mathrm{C}}$ を比較すると，BCSSTK02，BCSSTK15，BCSSTK18 に対しては，m_C が近い値を取っていることがわかる．同様に計算時間も Scheme C による手法が一番速いことがわかる．

以上より，この問題に対しては Scheme C が一番良い手法だということがわかる．

表 2.1　提案手法によるブロックサイズと計算時間の比較:BCSSTK

Problem	Matrix size	m_{opt}	t_{opt}	m_{A}	t_{A}	m_{B}	t_{B}	m_{C}	t_{C}
BCSSTK02	112×112	60	0.0008	32	0.0013	41	0.0014	51	0.0009
BCSSTK06	420×420	40	0.065	128	0.076	41	0.066	15	0.077
BCSSTK15	3948×3948	80	35.01	1024	64.93	43	39.27	53	37.3316
BCSSTK18	11948×11948	100	872.63	1024	1137.3	43	1035.5	55	959.9

表 2.2　提案手法によるブロックサイズと計算時間の比較:CAVITY

Problem	Matrix size	m_{opt}	t_{opt}	m_{A}	t_{A}	m_{B}	t_{B}	m_{C}	t_{C}
CAVITY03	317×317	90	0.035	128	0.041	41	0.037	55	0.038
CAVITY06	1182×1182	40	1.092	128	1.309	43	1.175	13	1.601
CAVITY10	2597×2597	80	10.57	256	15.25	46	11.47	55	11.00
CAVITY19	4562×4562	100	53.38	256	62.36	46	59.26	55	56.95

2.5.3　数値実験 3

本節では 1.4.2 節の数値例 2 を用いた．それぞれの手法で QR 分解する際のブロックサイズと計算時間を比較した．計算結果を表 2.2 に示す．表 2.2 より，数値例 2 に対しても，それぞれの手法は数値例 1 の場合と同じような結果になっていることがわかる．Scheme A は問題のサイズが大きくなると，m_{A} がとても大きくなっている．それに伴い，計算時間も増大している．逆に，Scheme B はほぼ同じ m_B を決定している．これは，最適に決定されているとは言えない．

これらに対し，Scheme C は CAVITY10，CAVITY19 に対して良いブロックサイズ m_{C} を決定している．計算時間 t_{C} も t_{opt} と比べて，約 110% 程度と非常に良い値を示している．しかし，まだ m_{C} が m と大きく異なっていたり，55 前後に集中している部分もある．

2.6　まとめ

本章では，BGS 法を効率的に使用するために，適切なブロックサイズ m の決定法を用いた BGS 法を提案した．適切なブロックサイズ m の決定法は，いくつかの小さいブロックサイズ m をサンプルとして実行し，計算量と計算時間の関係より，計算速度が最速に

なるブロックサイズ m を決定する手法である．この手法により，扱う問題によって異なる適切なブロックサイズ m を自動的に求めることが出来るようになり，効率的に BGS 法を使用することが可能となった．

本論文では，サンプルとして得られた計算時間から計算時間全体を予測する手法を 3 つ提案した．それぞれの手法を数値実験を用いて比較した結果，Scheme A は問題が大きくなるにつれて，m_{A} の値が非常に大きくなった．Scheme B は，問題が異なっても決定されるブロックサイズが変わらず，適応的に決定できるとは言えないことがわかった．これは，2.4.3 節で述べたように，サンプルブロックサイズ m において 1 ステップしか実行していないため，計算時間は計測出来るが，h 列目から $h+m$ 列目にいたるときの計算時間の増加量を求めることができなかったためである．

一方で，Scheme C は問題ごとにブロックサイズが変わっており，適応的にブロックサイズ m を決定できることが示された．また，最適なブロックサイズ m_{opt} の計算速度と比べ，90 %程度の計算速度が実現できた．

このように，Scheme C の手法により適応的にブロックサイズ m を決定する BGS 法が有効であることが示された．

第 3 章

Parallel Block Gram-Schmidt 法

本章では，まず BGS 法の並列化手法について述べる．次に BGS 法に対して用いた適応的にブロックサイズ m を決定する手法の PBGS 法への応用について述べる．

3.1 並列化

近年，スーパーコンピュータ“京”などに代表される，高い計算処理能力を持った計算機は，気象予報，気候予測，データマイニングなど，最先端の研究に幅広く利用されている．しかし，そこで扱われる問題は複雑で，大規模なことが多く，高速に計算するために並列化が必要となる．

Vanderstraeten [29] と Gudula ら [8] は，BGS 法の並列化手法について提案した．BGS 法は，MGS 法と比べ，通信回数が少ないため高い並列性を持っているが，CGS 法を基にしているため，直交行列の計算誤差が大きいという数値的に不安定な部分がある．そこで，数値的に安定になるようにブロックサイズ m を決定する PBGS 法を提案した．

本論文では，高速化に焦点を当てた PBGS 法を提案する [13, 15]．

3.1.1 分散方法

分散メモリ型並列計算機において，様々な行列の分散手法がある．

・**行方向分散 (Row-Wise Distribution (RWD))** 行方向分散は行列を行方向に分割し，各 Processor Element (PE) に記憶させる方法である．この場合，各 PE は行方向に分割された直交行列 Q と，直交化されるブロック行列 X_{block} を持つ．

・**列方向分散 (Column-Wise Distribution)** 列方向分散は行列を列方向に分割し，記

Algorithm 7 Parallel Block Gram-Schmidt Algorithm

Require: $X \in \mathbb{R}_{n\times n}$
Ensure: Q, R
  $K := P/m$
  for $k = 0 : K-1$ **do**
    if myrank == 0 **then**
      Broadcast (X_{block})
    else
      Receive (X_{block})
    end if
    $R_{12} := Q^T X_{\text{block}}$
    $\widehat{Y} = X_{\text{block}} - QR_{12}$
    if myrank == 0 **then**
      Receive ($\widehat{Y}$) from each PE
    else
      send ($\widehat{Y}$)
    end if
  end for

憶させる手法である．この場合，PE は分割された直交行列 Q と，直交化されるブロック行列 X_{block} を保持する．

3.1.2 列方向分散を用いた Parallel Block Gram-Schmidt 法

本論文では，Column-Wise Distribution (CWD) を使って，BGS 法を並列化した手法 [9] を PBGS 法と呼ぶことにする．CWD は，RWD と比べて，実装が行いやすく，反復手法などに対して応用しやすいという利点を持っている．Algorithm 7 に列方向分散を用いた PBGS 法の算法を示す．

3.2 計算量

PBGS 法の計算量について述べる．$X \in \mathbb{R}^{n\times n}, X_{\text{block}} \in \mathbb{R}^{n\times m}$ とし，ブロックサイズは m とする．簡単のために m は n を割り切る数とする．PBGS 法の計算量は，BGS 法と同じである．これは，並列化によって各 PE にデータを分散する必要が生じるが，新しい計算過程が生じるわけではないことからわかる．

3.3 最適なブロックサイズの検討

PBGS 法において，プロセッサー数の変化により適切なブロックサイズが変化することが予想される．そこで，2 章の提案手法を拡張し，PBGS 法に対する適切なブロックサイズ m の決定法を考える．

3.3.1 Scheme D

2 章でも述べたように BGS 法においては，1 つの CPU 上で最も性能が発揮されるようなブロックサイズが最適だと考えられる．すなわち，Scheme C において BGS 法を用いていた部分を PBGS 法に置き換えることで，ブロックサイズ m の決定手法を適用させることが可能になる．Scheme D の算法を Algorithm 8 に示す．

3.4 数値実験

本節では，PBGS 法の有用性を示すために，実験環境 1 で C 言語を用いて，数値実験を行った．通信ライブラリは MPI を用いた．本節において，数値実験の結果を示す表に使われる記号はそれぞれ次を表す．

- PE : プロセッサ．PE の後ろの数字は PE 数を示している
- $m_{\rm opt}$: BGS 法のブロックサイズ $m, m = 10, 20, \ldots, 300,$ の中で最適なブロックサイズ
- $m_{\rm D}$: 3.3.1 節の Scheme D によって決められたブロックサイズ
- t : 計算時間（秒）

3.4.1 数値実験 1

本節では 1.4.1 節の数値例 1 の BCSSTK15，BCSSTK18 に対して，BGS 法と Scheme D を適用した PBGS 法を用いて，QR 分解にかかる時間を比較した．結果を表 3.1，表 3.2，表 3.3 に示す．

表 3.1　Block Gram-Schmidt 法と Parallel Block Gram-Schmidt 法の比較 1

Problem	Matrix size	m_{opt}	t_m	m_{D}(PE2)	t_{D}(PE2)	m_{D}(PE4)	t_{D}(PE4)
BCSSTK15	3948×3948	80	35.01	150	28.40	210	18.15
BCSSTK18	11948×11948	100	872.6	180	524.0	230	489.3

表 3.2　Block Gram-Schmidt 法と Parallel Block Gram-Schmidt 法の比較 2

Problem	Matrix size	m_{opt}	t_m	m_{D}(PE6)	t_{D}(PE6)	m_{D}(PE8)	t_{D}(PE8)
BCSSTK15	3948×3948	80	35.01	250	15.18	260	15.39
BCSSTK18	11948×11948	100	872.6	290	451.3	320	440.2

表 3.3　Block Gram-Schmidt 法と Parallel Block Gram-Schmidt 法の比較 3

Problem	Matrix size	m_{opt}	t_m	m_{D}(PE10)	t_{D}(PE10)	m_{D}(PE12)	t_{D}(PE12)
BCSSTK15	3948×3948	80	35.01	280	14.83	310	15.55
BCSSTK18	11948×11948	100	872.6	380	392.6	420	376.3

BCSSTK15，BCSSTK18 のどちらの問題に対しても，並列化し複数の PE で計算することで，BGS 法と比べて，PBGS 法が高速に直交行列を計算出来ていることがわかる．また，PBGS 法に Scheme D を適用して決定したブロックサイズ m_{D} は，BCSSTK15 と BCSSTK18 の問題に対して，BGS 法の最適なブロックサイズ m_{opt} よりも大きくなっている．これは，複数の PE で並列計算することで，一度に扱える行列のサイズが大きくなるためであると考えられる．すなわち，PBGS 法の 1 ステップにおいて，各 PE が m_{opt} サイズの X_{block} を直交化することで最速になる，ということである．それによって PBGS 法全体としては，一度に扱える行列のサイズが大きくなる．故に，PBGS 法に Scheme D を適用して決定したブロックサイズ m_{D} は m_{opt} よりも大きいと考えられる．しかし，ブロックサイズ m が大きくなることによって，通信にかかるコストも大きくなるため，PE 数に比例してはいないと考えられる．同様に，PE 数を増やしても速度向上率は比例していない．これは通信コストや，ブロックサイズを決定する部分の計算などの並列化できていない計算部分の影響と考えられる．

しかし，どちらの問題に対しても PBGS 法は BGS 法を高速化している．特に PE 数

表 3.4　Scheme D を用いた Parallel Block Gram-Schmidt 法の比較：BCSSTK15

Method	m	t	Raito of t
PBGS(PE8)	50	35.67	1.96
PBGS(PE8)	100	21.22	1.17
PBGS(PE8)	200	15.62	0.86
PBGS(PE8)	260	15.39	0.85
PBGS(PE8)	300	16.20	0.89
PBGS-Scheme D(PE8)	180	18.16	1.00

が 2 と 4 の時は，PE 数に比例した計算速度が実現できているので，良く並列化出来ていると考えられる．

以上より，PBGS 法は BGS 法の高速化に有効であると言える．またブロックサイズ m の適応的な決定法が，問題ごと，PE 数ごとに適応的にブロックサイズ m を決定し，効率的に計算が行われていることを示している．

3.4.2　数値実験 2

本節では 1.4.1 節の数値例 1 の BCSSTK15 に対して，Scheme D を適用した PBGS 法と，任意に決めた様々なブロックサイズを用いた PBGS 法をそれぞれ適用して，QR 分解した際ににかかる時間を比較した．PBGS 法は PE 数が 8 の場合を用いた．計算結果を表 3.4 に示す．ただし，Raito of t は提案手法の速度を 1 とした時の，他の手法との実行時間の割合である．

表 3.4 より，Scheme D を用いた PBGS 法は，最速でないことがわかる．しかしながら，Raito of t を見ると分かるように，$m = 50$ の場合に対しては 2 倍程度，$m = 100$ の場合に対しても高速化できている．ただし，$m = 200$，$m = 300$ の場合に対しては，10 から 15%程度遅くなっている．このように，Scheme D を適用させた PBGS 法は最速ではないが，常に最速な場合の 90%程度の計算速度で直交化が行われている．これより，適応的なブロックサイズ m の決定法を適用した PBGS 法は $m = 50$ などの計算速度が最速な場合と比べて 2 倍以上遅くなってしまうようなブロックサイズ m を選択してしまうことを回避することができる手法である，と考えられる．

3.5 まとめ

本章では，CWD を用いた BGS 法の並列化，BGS 法の高速化と，PBGS 法の実装について述べた．さらに BGS 法の場合と同様，ブロックサイズ m によって計算時間が異なるため，適応的にブロックサイズ m を決定する手法を提案した．

2 章において提案した，適応的にブロックサイズ m を決定する 3 つの手法のうち，Scheme C が最も良い手法であったため，本章においては Scheme C を PBGS 法に適用した．3.3 節において述べたように，Scheme C の算法において，BGS 法を PBGS 法に変更するだけで，PBGS 法に適用することができ，応用しやすい手法であることがわかった．

そして数値実験において，BGS 法を並列化することで高速化出来ることと，適応的なブロックサイズ m の決定法が，PBGS 法に適用することができることを示した．そして，PBGS 法に対しても，ブロックサイズ m の決定手法が有効であることを示した．

Algorithm 8 Scheme D

```
Require: X ∈ ℝ^{n×n}
Ensure: m
  for i = 1 : 5 do
    m_i := 2^{i−1}
    for j = 0 : 1 do
      if myrank==0 then
        start := gettimeofday()
      end if
      Parallel Block Gram-Schmidt method
      if myrank==0 then
        end := gettimeofday()
        t_{ij} := end − start
      end if
    end for
    if myrank==0 then
      a = (t_{i0} − t_{i1})/m_i
      b[i] := (1/2) n^2 a + t_{i0} − a(h − m_i)
      for j = 5 : 1 do
        A[i, j] = m_i^{j−1}
      end for
    end if
  end for
  if myrank==0 then
    solve Ax = b
    f(m) := x_1 m^4 + x_2 m^3 + x_3 m^2 + x_4 m + x_5
    solve m := min_{m∈[0, (1/2)N]} f(m)
  end if
```

第 4 章

Block Symplectic Gram-Schmidt 法

本章では，まず Symplectic Gram-Schmidt(SGS) 法について述べる．そして，SGS 法のブロック化とブロックサイズ m の適切な決定手法を提案し，数値実験によって有効性を示す．

4.1 Symplectic Gram-Schmidt 法

Symplectic Gram-Schmidt (SGS) 法は，与えられた行列 X に対して $X = SR$ を満たす Symplectic 行列 S と，上三角行列 R を計算する．SR 分解は，QR 分解と類似した手法で，QR 法と同様に X の固有値，固有ベクトルを計算することが出来る．

様々な応用分野において，特殊な構造を持つ行列の固有値を求めることはよくある．例えば，最適制御理論から導出される Riccati 方程式の解は Hamiltonian 行列の固有値・固有ベクトルを求めることで数値的に解くことが出来る [6]．このように，Hamiltonian 行列の固有値問題は非常に重要であり，現在までにいくつかの研究が行われている [1, 3, 10]．その中で，SR 法は QR 法とは異なり行列 X の重要な構造を保持したまま計算することが可能である．Van Loan [10] によれば，特に Hamiltonian 行列に対しては，QR 法と比べて，SR 法は約 1/4 の計算コストと記憶領域だけを必要とすることが報告されている．また大規模疎行列の固有値問題に対しては，Symplectic Lanzcos 法が提案されている [3]．この手法では，Symplectic なベクトル列を生成するために SGS 法が使われている．

これまで複数の SR 分解のアルゴリズムが提案されている．そのうちの 1 つが Symplectic Hauseholder 法と Symplectic Givens 回転を用いた SR 手法である [10]．この手法は，Hauseholder 法を用いた QR 法と似ている．Salam [22] は，この手法が Modified SGS による SR 分解と数学的に同値であることを証明した．他の SR 分解手法は，Classical あるいは Modified SGS 法を用いたものである [21]．

このように，CGS 法と同様，SGS 法も応用範囲の広い手法である．ただ，CGS 法を用いた QR 分解と比べ，SGS 法を用いた SR 分解に関する研究は少ない．Salam [21] によると，SGS 法はパラメーターの取り方により J-直交精度が大きく変化することが指摘されているが，手法自体の誤差解析などは報告されていない．

本論文では，まず SGS 法を用いた SR 分解の誤差の伝播について考える．SGS はアルゴリズム内において，直交化されるベクトルから足し算と引き算を行うため桁落ちなど丸め誤差の影響を受けやすい傾向がある．また直交化されたベクトルで正規化をしないため，直交化されたベクトルノルムが大きくなる可能性がある．

これらより SGS 法は，GS 法が生成する直交基底の直交性と比べ直交性の悪化が頻繁に起きる恐れがある．そこで，本章では J-直交行列の計算誤差を解析するとともに，再直交化の条件に関する検討を行った．そして，CSGS 法を高速化するために，ブロック化した Block Symplectic Gram-Schmidt(BSGS) 法とそれに対する適応的なブロックサイズ m の決定法を提案する．最後に，数値実験を用いて提案手法の有効性を示す．

4.1.1 表記法

本節では，SGS 法に関わる手法についての表記について述べる．行列サイズが $n \times n$ の単位行列 I_n と零行列 0_n に対して，行列 $J \in \mathbb{R}^{2n \times 2n}$ を次式で定義する．

$$J = \begin{bmatrix} 0_n & I_n \\ -I_n & 0_n \end{bmatrix} \tag{4.1}$$

ただし，$J^{\mathrm{T}} = J^{-1} = -J$ である．

そして，2 つのベクトル $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{2n}$ に対して，J-内積を次のように定める．

$$< \boldsymbol{x}, \boldsymbol{y} >_J = \boldsymbol{x}^{\mathrm{T}} J \boldsymbol{y} \tag{4.2}$$

また，行列 M に対して，M^{J} を次のように定義する．

$$M^{\mathrm{J}} = J^{\mathrm{T}} M^{\mathrm{T}} J \tag{4.3}$$

Algorithm 9 Elementary SR Factorization

Require: $A_1 = [\boldsymbol{a}_1, \boldsymbol{a}_2]$
Ensure: $S_1 = [\boldsymbol{s}_1, \boldsymbol{s}_2], R_1 = [r_{11}, r_{12}, r_{21}, r_{22}]$
Choose $r_{11} \in \mathbb{R}$, $\boldsymbol{s}_1 = \boldsymbol{a}_1 / r_{11}$
Choose $r_{12} \in \mathbb{R}$, $\boldsymbol{y} = \boldsymbol{a}_2 - r_{12}\boldsymbol{s}_1$
$r_{22} = \boldsymbol{s}_1^{\mathrm{T}} J \boldsymbol{y}$
$\boldsymbol{s}_2 = \boldsymbol{y} / r_{22}$

そして，行列 S が次式を満たすとき，S は Symplectic，あるいは J-直交であるという．

$$S^{\mathrm{J}} S = J^{\mathrm{T}} S^{\mathrm{T}} J S = I \tag{4.4}$$

4.1.2　算法

本節では，与えられたベクトル列から J-直交ベクトル列を生成する算法について述べる．まず，与えられた 2 つのベクトル列 $X_1 = [\boldsymbol{x}_1, \boldsymbol{x}_2], \boldsymbol{x}_i \in \mathbb{R}^n, i = 1, 2$ を J-直交させる Elementary SR Factorization(ESR) 法を Algorithm 9 に示す．ESR 法は与えられた行列 X_1 に対して，次のように計算することになる．

$$\begin{aligned} \boldsymbol{s}_1 &= \frac{\boldsymbol{x}_1}{r_{11}} \\ \boldsymbol{y} &= \boldsymbol{x}_2 - r_{12}\boldsymbol{s}_1 \\ r_{22} &= \boldsymbol{s}_1^{\mathrm{T}} J \boldsymbol{y} \\ \boldsymbol{s}_2 &= \frac{\boldsymbol{y}}{r_{22}} \end{aligned} \tag{4.5}$$

ただし，r_{11}, r_{12} は任意の実数である．以上をまとめると次式が成り立つ．

$$X_1 = S_1 R_1 \tag{4.6}$$

ただし，行列 $S_1 = [\boldsymbol{s}_1, \boldsymbol{s}_2]$ は J-直交行列であり，$R_1 = [r_{11}, r_{12}, r_{21}, r_{22}]$ は上三角行列である．ここで，r_{11}, r_{12} は任意の実数である．Salam [21] は，このパラメーターの選び方についての提案を行っている．

- ESR1 法: $r_{11} = \|\boldsymbol{x}_1\|, r_{12} = 0$
- ESR2 法: $r_{11} = \|\boldsymbol{x}_1\|, r_{12} = \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2$

- ESR3 法: $r_{11} = \|\boldsymbol{x}_1^T J \boldsymbol{x}_2\|, r_{12} = 0$

特に ESR2 法において，$r_{11} = \|\boldsymbol{x}_1\|, r_{12} = \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2$ であるので，式 (4.5) より次式が成り立つ．

$$\boldsymbol{s}_1 = \frac{\boldsymbol{x}_1}{r_{11}} = \frac{\boldsymbol{x}_1}{\|\boldsymbol{x}_1\|} \tag{4.7}$$

$$\boldsymbol{s}_2 = \frac{\boldsymbol{y}}{r_{22}} = \frac{\boldsymbol{x}_2 - \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2 \boldsymbol{s}_1}{\boldsymbol{s}_1^{\mathrm{T}} J \boldsymbol{x}_2 - \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2 \boldsymbol{s}_1} = \frac{\boldsymbol{x}_2 - \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2 \boldsymbol{s}_1}{\boldsymbol{s}_1^{\mathrm{T}} J \boldsymbol{x}_2} \tag{4.8}$$

式 (4.7) と式 (4.8) より，次式が成立する．

$$\boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{s}_2 = \frac{\boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2 - \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2 \boldsymbol{s}_1}{\boldsymbol{s}_1^{\mathrm{T}} J \boldsymbol{x}_2} = 0 \tag{4.9}$$

従って ESR2 法を用いると，$\boldsymbol{s}_1$ と $\boldsymbol{s}_2$ は $\boldsymbol{s}_1 \perp \boldsymbol{s}_2$ が成り立つ．これは $\|\boldsymbol{s}_2\|$ が最小になる選び方であるので，これらの中で最も数値的に安定していることが報告されている [21]．本論文では，ESR2 法を使用することにする．

与えられた行列 $A \in \mathbb{R}^{2n \times 2n}$ に対して，次のような分解をする手法の 1 つに Classical Symplectic Gram-Schmidt(CSGS) 法がある．

$$A = SR \tag{4.10}$$

ただし，S は J-直交行列であり，R は上三角行列である．与えられた J-直交行列 S に対して，CSGS 法によるベクトル列 $X = [\boldsymbol{x}_1, \boldsymbol{x}_2]$ の J-直交化は次のようになる．

$$H_{12} = S^J X \tag{4.11}$$

$$Y = X - SH_{12} \tag{4.12}$$

$$Y \to SR \quad \text{(by ESR)} \tag{4.13}$$

式 (4.12) と式 (4.13) からわかるように，CSGS 法の計算は CGS 法に似ている．しかし，CGS 法と異なり正規化を行わず，代わりに ESR 法を用いて $W_i = [\boldsymbol{w}_{2i-1}, \boldsymbol{w}_{2i}]$ の関係を定めている．以下これを繰り返すことによって，式 (4.10) を得る．Algorithm 10 に CSGS 法の算法を示す．

Algorithm 10 Classical Symplectic Gram-Schmidt Algorithm

Require: $X_1, \cdots, X_n$
Ensure: $S = [S_1, \cdots, S_n], R$
  $X_1 = S_1 R(1:2, 1:2)$
  for $i = 2:n$ **do**
    for $j = 1:i-1$ **do**
      $H_{i,j} = S_j^{\mathrm{T}} X_i$
    end for
    $Y_i = X_i - \sum_{j=1}^{j=i-1} S_j H_{i,j}$
    $R(1:2(i-1), 2i-1:2i) = H_{j,i}$
    if $(Y_i \neq 0)$ **then**
      $Y_i = S_i \acute{R}$ by ESR algorithm
    else
      exit
    end if
    $R(2i-1:2i, 2i-1:2i) = \acute{R}$
  end for

CSGS 法において，式 (4.12) と式 (4.13) が主となる部分であり，CGS 法の次式に対応している．

$$\boldsymbol{r} = Q^T \boldsymbol{x}$$
$$\hat{\boldsymbol{y}} = \boldsymbol{x} - Q\boldsymbol{r}$$

また，この算法によって生成されたベクトル列 $S_1, \ldots, S_n, S_i = [\boldsymbol{s}_{2i-1}, \boldsymbol{s}_{2i}]$ は次式を満たす．

$$\boldsymbol{s}_{2i-1}^{\mathrm{T}} J \boldsymbol{s}_{2i} = 1, \quad i = 1, \ldots, n \tag{4.14}$$

$$S_i^{\mathrm{J}} S_j = 0, \quad i \neq j, \quad S_i^{\mathrm{J}} S_j = 1, \quad i = j \tag{4.15}$$

よって，次式が成立する．

$$\dim \operatorname{span}\{S_1, \ldots, S_n\} = 2n \tag{4.16}$$

Algorithm 11 Modified Symplectic Gram-Schmidt Algorithm

Require: $X_1, \cdots, X_n$
Ensure: $S_1, \cdots, S_n, R$
  $X_1 = S_1 R(1:2, 1:2)$
  for $i = 2 : n$ **do**
    $Y_i = X_i$
    for $j = 1 : i-1$ **do**
      $H_{i,j} = S_j^{\mathrm{J}} Y_i$
      $Y_i = Y_i - \sum_{j=1}^{j=i-1} S_j H_{i,j}$
    end for
    $R(1:2(i-1), 2i-1:2i) = H_{j,i}$
    if $(Y_i \neq 0)$ **then**
      $Y_i = S_i \acute{R}$ by ESR algorithm
    else
      exit
    end if
    $R(2i-1:2i, 2i-1:2i) = \acute{R}$
  end for

このように CSGS 法は与えられた行列と同様の空間を張るベクトル列を生成する．

次に，Modified Symplectic Gram-Schmidt(MSGS) 法について述べる．MSGS 法は，与えられた J-直交行列 S に対して，次式で J-直交化を行う．

$$Y = X \tag{4.17}$$

for $i = 1, \ldots, j-1$

$$H_i = S_i^J W \tag{4.18}$$

$$Y = Y - S_i H_i \tag{4.19}$$

end for

$$Y \to SR, \quad \text{(by ESR)} \tag{4.20}$$

CSGS 法の場合と同様，MSGS 法も MGS 法と似た手法である．また，正規化を行わずに ESR 法を用いて $Y_i = [\boldsymbol{y}_{2i-1}, \boldsymbol{y}_{2i}]$ の関係を定めている．

Algorithm 11 に MSGS 法の算法を示す．MGS 法の場合と同じように MSGS 法にお

いても，CSGS 法よりも MSGS 法の方が，J-直交行列の計算精度が高いことがわかっている [22].

ここで，GS 法と SGS 法の違いについて述べる．

- Gram-Schmidt 法
  - $X \in \mathbb{R}^{n\times n} \rightarrow QR$
    $\mathrm{span}(X) = \mathrm{span}(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n)$, where $< \boldsymbol{q}_i, \boldsymbol{q}_j >= 0$, $i \neq j$
- Symplectic Gram-Schmidt 法
  - $X \in \mathbb{R}^{2n\times 2n} \rightarrow SR$
    $\mathrm{span}(X) = \mathrm{span}(S_1, \ldots, S_n)$
    $\Leftrightarrow \mathrm{span}(X) = \mathrm{span}([\boldsymbol{s}_1, \boldsymbol{s}_2], [\boldsymbol{s}_3, \boldsymbol{s}_4], \ldots, [\boldsymbol{s}_{2n-1}, \boldsymbol{s}_{2n}])$
    $< S_i, S_j >_J = 0$, $i \neq j$

GS 法は与えられた行列 $X \in \mathbb{R}^{n\times n}$ に対して，n 本の正規直交基底列を生成する算法である．このとき行列 X が張る空間と正規直交基底列が張る空間は同じになる．一方で SGS 法は，与えられた行列 $X \in \mathbb{R}^{2n\times 2n}$ に対して，n 個の J-直交行列 S_i を生成する．このとき行列 X が張る空間と J-直交行列 S_i が張る空間は同じになる．また，ESR2 法を用いると，$S_i = [\boldsymbol{s}_{2i-1}, \boldsymbol{s}_{2i}]$ において $\boldsymbol{s}_{2i-1}$ と $\boldsymbol{s}_{2i}$ は直交している．

4.1.3 J-直交性の崩れ

本節では，SGS 法による SR 分解の J-直交行列の数値的性質について述べる．一般的に QR 分解によって得られた直交行列 Q の直交性の精度は，次式のようになる．ただし，行列 I は単位行列である．

$$||I - Q^T Q||_2 \tag{4.21}$$

Björck [4] によると，丸め誤差による影響は QR 分解された行列 A の条件数に比例することを示した．また，Stewart [27] によって再直交化するための指標が導入された．本節では，SGS 法に対して，J-直交化の精度について考える．

まず，J-直交行列の計算精度を次式で表すことにする．

$$||I - S^J S||_2 \tag{4.22}$$

Salam [21] により，次の指摘がなされた．ESR 法において r_{11} と r_{12} の選択の仕方により，J-直交性が大きく変化して崩れることがあり，また，行列 X の条件数が 1 のように小さかったとしても，SGS 法においては J-直交性が崩れることもある．この詳細な理由は述べられていないが，一般的な条件数の定義を用いる代わりに $\mathrm{cond}(X) = \sqrt{(X^{\mathrm{T}}JX)}$ を用いることを提案している．

いま，CSGS 法の $i = k+1$ ステップ目が行われていると仮定する．新たに生成される J-直交ベクトル列 Y_{k+1} は，次式のようになる．

$$Y_{k+1} = X_{k+1} - (S_1 H_{i,1} + S_2 H_{i,2} + \cdots + S_k H_{i,k}) \tag{4.23}$$

$$\begin{aligned} &= X_{k+1} - \left(\left(\boldsymbol{s}_2^{\mathrm{T}} J X_{k+1} \right) \boldsymbol{s}_1 + \cdots + \left(\boldsymbol{s}_{2k}^{\mathrm{T}} J X_{k+1} \right) \boldsymbol{s}_{2k-1} \right) \\ &\quad - \left(\left(\boldsymbol{s}_1^{\mathrm{T}} J X_{k+1} \right) \boldsymbol{s}_2 + \cdots + \left(\boldsymbol{s}_{2k-1}^{\mathrm{T}} J X_{k+1} \right) \boldsymbol{s}_{2k} \right) \end{aligned} \tag{4.24}$$

SGS 法は，CGS 法と異なり J による変換によって行列 Y_{k+1} に加算と減算を繰り返す．これにより，桁落ちのような丸め誤差の影響を大きく受けてしまう可能性がある．

ESR アルゴリズムにおいて，$r_{11} = \|\boldsymbol{x}_1\|, r_{12} = \boldsymbol{s}_1^{\mathrm{T}} \boldsymbol{x}_2$ と取ると，$\boldsymbol{s}_{2i}$ は取り得る値の最小値となるが，$1 = \|\boldsymbol{s}_1\| \leq \|\boldsymbol{s}_2\|$ となり，$\|\boldsymbol{s}_{2i}\|$ の大きさは常に 1 より大きくなる [21]．このことより，$\|\boldsymbol{s}_{2i}\|$ が行列 X のベクトル列の大きさと比較して，非常に大きくなる可能性がある．今，$\|\boldsymbol{s}_{2k}\| \gg \|\boldsymbol{s}_j\|$ とし，$\boldsymbol{s}_i^{\mathrm{T}} J \boldsymbol{s}_i = \varepsilon$ が成り立つとする．ただし，ε はマシンイプシロンで，$i \neq j$ とする．すると，式 (4.24) より次式が成立する．

$$\begin{aligned} \boldsymbol{s}_{2k-1}^{\mathrm{T}} J Y_{k+1} &= \boldsymbol{s}_{2k-1}^{\mathrm{T}} J X_{k+1} - \boldsymbol{s}_{2k-1}^{\mathrm{T}} J \left(S_1 H_{i,1} + S_2 H_{i,2} + \cdots + S_k H_{i,k} \right) \\ &= \boldsymbol{s}_{2k-1}^{\mathrm{T}} J X_{k+1} - (\varepsilon - \varepsilon + \cdots + \boldsymbol{s}_{2k-1}^{\mathrm{T}} J X_{k+1} - (\boldsymbol{s}_{2k}^{\mathrm{T}} J X_{k+1})\varepsilon) \\ &= \boldsymbol{s}_{2k-1}^{\mathrm{T}} J X_{k+1} - \boldsymbol{s}_{2k-1}^{\mathrm{T}} J X_{k+1} + (\boldsymbol{s}_{2k}^{\mathrm{T}} J X_{k+1})\varepsilon \\ &\simeq (\boldsymbol{s}_{2k}^{\mathrm{T}} J X_{k+1})\varepsilon \neq 0 \end{aligned} \tag{4.25}$$

4.1.4 Symplectic Gram-Schmidt 法の再直交化

GS 法の再直交化も重要な研究の 1 つである [4, 27]．GS 法は直交化されるベクトル列が互いに平行に近い場合や丸め誤差の影響などにより，直交行列を正確に求められない場

合がある．このような場合には，再直交化を行う．しかし，再直交化することで計算コストが増加するので，良い再直交化の条件を検討することが非常に重要になる．GS 法のステップによって直交化されたベクトル $\hat{\boldsymbol{y}}$ が，次式を満たすものとする．

$$\|\hat{\boldsymbol{y}}\| \geq \frac{1}{2}\|\boldsymbol{x}\| \tag{4.26}$$

このことより，$\hat{\boldsymbol{y}}$ の直交化されていない成分の大きさが 2ε 以下となる．逆に，この条件が満たされない場合は，再直交化すれば良いことがわかる [27]．

しかし，SGS 法に対しては，これがうまく機能しないことが予想される．4.1.3 節で示したように，S のノルムが大きくなる傾向があるので，式 (4.26) を満たしているが，J-直交性が崩れてしまうことが予測される．そのような場合においても再直交化が行われるように，次のような新しい再直交化条件を考える．

$$\|\boldsymbol{y}\| \leq \frac{1}{2}\|\boldsymbol{x}\| \tag{4.27}$$

この式を用いると，$\boldsymbol{y}$ のノルムを上からの有界性が得られ，J-直交化されたベクトルのノルムが大きくなることを防ぎ，J-直交性が崩れることを回避することが出来る．

4.2 ブロック化

本節では，CSGS 法のブロック化を提案する．1 章と 2 章で述べたように，Stewart [27] は，GS 法をブロック化することにより，正規直交基底列を高速に計算することが出来ることを示した．そこで，CSGS 法を高速化するのブロック化を考察する．まず，CSGS 法は式 (4.12) と式 (4.13) である．これらの式中の X を $X_{\text{block}} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]$ で置き換えると，式 (4.12) と式 (4.13) は次式になる．

$$H_{12} = S^J X_{\text{block}} \tag{4.28}$$

$$Y = X_{\text{block}} - SH_{12}. \tag{4.29}$$

式 (4.28) と式 (4.29) により，X_{block} は J-直交行列 S を用いて J-直交化することが出来る．しかし BGS 法と同様に，これだけでは Y 中のベクトル列どうしは J-直交化できて

Algorithm 12 Block Symplectic Gram-Schmidt Algorithm

Require: X=$[X_{\text{block}_1}, \cdots, X_{\text{block}_n}]$
Ensure: $S = [S_1, \cdots, S_n], R$
  $X_{\text{block}_1} = S_1 R(1:2, 1:2)$
  for $i = 2:n$ **do**
    for $j = 1:i-1$ **do**
      $H_{i,j} = S_j^J X_{\text{block}_i}$
    end for
    $Y_i = X_{\text{block}_i} - \sum_{j=1}^{j=i-1} S_j H_{i,j}$
    $R(1:2(i-1), 2i-1:2i) = H_{j,i}$
    $Y_i = S_i \acute{R}$ by CSGS method
    $R(2i-1:2i, 2i-1:2i) = \acute{R}$
  end for

いない．そこで Y を，次式のように CSGS 法によって，J-直交化させる．

$$Y \rightarrow SR, \quad (\text{ by CSGS}) \tag{4.30}$$

そうすることで，CSGS 法をブロック化することが可能となる．Block Symplectic Gram-Schmidt(BSGS) 法の算法を Algorithm 12 に示す．CSGS 法のブロック化によって，BLAS を用いて高速に演算することが可能になる．

次に，CSGS 法，MSGS 法，BSGS 法の性質を比較する．

- Classical Symplectic Gram-Schmidt 法
  - 計算精度：低い → 再直交化
  - 並列性：高い
- Modified Symplectic Gram-Schmidt 法
  - 計算精度：高い
  - 並列性：低い
- Block Symplectic Gram-Schmidt 法
  - 計算精度：低い

– 並列性：高い

GS 法の場合と同様に，MSGS 法は CSGS 法より計算精度が高い．BSGS 法は CSGS 法をブロック化しているため，CSGS 法と類似の性質を持っている．しかし，BSGS 法の X_{block_i} の列ベクトルは，式 (4.28) と式 (4.29) によって，まず行列 S に対して J-直交化を計算した後，式 (4.30) によって，X_{block_i} 中のベクトル列どうしを J-直交化しているため，一部 MSGS 法を取り入れた形になっていると考えられる．よって，わずかではあるが，CSGS 法より計算精度が高くなっている可能性がある．

4.3 適応的なブロックサイズの決定手法

本節では，BSGS 法に対して，適応的にブロックサイズ m を決定する手法を提案する．

4.3.1 計算量

まず，本節では，BSGS 法の計算量について考察する．今，$X \in \mathbb{R}^{2n\times 2n}$ とし，ブロックサイズを m とする．掛け算の計算 1 回を 1 ユニットとする．$S \in \mathbb{R}^{2n\times h}$ と仮定し，$k+1$ ステップ目の BSGS 法の計算量を考える．ただし，$h = k \times m$ とする．まず，式 (4.28) の計算の部分が次のようになる．

$$4n \times 2n^2 \times 2nm = 16n^4 m \tag{4.31}$$

次に，式 (4.29) の計算の部分が次のようになる．

$$2k \times n \times m = 2knm \tag{4.32}$$

そして，Y を J-直交化させる計算の部分が，以下のようになる．

$$\sum_{k=1}^{m-1} 4nk\left(8n^3+1\right) = 2nm(m-1)\left(8n^3+1\right) \tag{4.33}$$

よって，式 (4.31)，式 (4.32)，式 (4.33) の計算量をまとめると，1 ステップあたりの BSGS 法の計算量は次のようになる．

$$16n^4m + 2knm + 2nm(m-1)\left(8n^3+1\right) = 2nm\left(8mn^3+k+m-1\right) \tag{4.34}$$

これより BSGS 法全体の計算量 C_{bsgs} は，次式で与えることが出来る．

$$\begin{aligned} C_{bsgs} &= \sum_{k=1}^{n/m-1} 2nm\left(8mn^3+k+m-1\right) \\ &= n\left(\frac{n}{m}-1\right)\left(16m^2n^3+m^2-m+n\right) \end{aligned} \tag{4.35}$$

以上より，式 (4.34) と式 (4.35) を用いて，BSGS 法に対して，適応的にブロックサイズ m を決定する手法を提案する．

4.3.2 Scheme E

1 章と 2 章で述べたように，BGS 法ではブロック化することにより高速化することが出来るが，ブロックサイズ m によって計算時間が変化するという問題点があった．BSGS 法においても同様に，ブロックサイズ m によって計算時間が異なると考えられる．

そこで，2 章のにおいて最も有効な提案手法であった Scheme C を，BSGS 法に対しても適用し，適応的にブロックサイズ m を決定する手法，Scheme E，を提案する．3 章において述べたように，Scheme C は他の手法に対して適応させやすい手法である．BSGS 法に対しても，3 章と同様に Algorithm 6 において，BGS 法を BSGS 法に置き換えることで容易に適用させることが出来る．適応的にブロックサイズを決定する BSGS 法の算法を Algorithm 13 に示す．

4.4 数値実験

本節では，4.1.3 節から 4.3 節で述べた J-直交性の崩れ，CSGS のブロック化の有効性，BSGS 法に対するブロックサイズの適応的な決定手法の有効性を数値実験を用いて示す．実験環境 2 において C 言語を用いて数値実験を行った．数値実験の結果を示す表に使われる記号はそれぞれ次を表す．

Algorithm 13 Scheme E

Require: :$X \in \mathbb{R}^{n\times n}$
Ensure: m
  for $i = 1:5$ **do**
    $m_i := 2^{i-1}$
    for $j = 0:1$ **do**
      $start := gettimeofday()$
      Block Symplectic Gram-Schmidt method
      $end := gettimeofday()$
      $t_{ij} := end - start$
    end for
    $a = (t_{i0} - t_{i1})/m_i$
    $b[i] := (1/2)\, n^2 a + t_{i0} - a(h - m)$
    for $j = 5:1$ **do**
      $A[i,j] = m_i^{j-1}$
    end for
  end for
  solve $A\boldsymbol{x} = \boldsymbol{b}$
  $f(m) := x_1 m^4 + x_2 m^3 + x_3 m^2 + x_4 m + x_5$
  solve $m := \min_{m\in[0,\frac{1}{2}N]} f(m)$

- CSGS : Classical Symplectic Gram-Schmidt 法
- MSGS : Modified Symplectic Gram-Schmidt 法
- BSGS : Block Symplectic Gram-Schmidt 法
- BSGS-m : Block Symplectic Gram-Schmidt 法に Scheme E を適用させた手法
- t : 計算時間（秒）
- Accuracy : 式 (4.22) を用いて計算した J-直交行列の計算精度

4.4.1 数値実験 1

本節では，SGS 法の J-直交性の崩れと，再直交化について，1.4.3 節の数値例 3 を用いて，Matlab 上で数値実験を行った．CSGS 法と MSGS 法を用いて SR 分解を行い，式 (4.22) を用いて J-直交行列の計算精度をそれぞれを 5 回ずつ測った．

計算結果を図 4.2，図 4.3 に示す．図 4.2 において，横軸は行列サイズ，縦軸は式 (4.22) を用いて計算した J-直交行列の計算誤差の値である．それぞれの折れ線は，各 1 回目から 5 回目の試行に対応している．比較のために，CGS 法を用いて同様の行列に対して QR 分解を行い，式 (4.21) を用いて直交行列の計算精度を調べた結果を図 4.1 に示す．

図 4.1 からわかるように，10 から 200 程度のサイズ行列に対しての CGS 法は，10^{-15} 程度の非常に高い精度で直交行列が計算できている．一方，図 4.2 からわかるように，SGS 法による J-直交行列の計算は，行列サイズが小さいものに対しては，GS 法と同程度の精度で計算できているが，行列サイズが 100 程度まで行くと，J-直交行列が正確に計算できていないことがわかる．また，図 4.3 より，MSGS 法においても GS 法と同程度の直交精度は計算できていないことがわかる．しかしながら，CSGS 法よりも良く J-直交行列が計算できている．このことからも MSGS 法が，CGSG 法よりも数値的に安定であることがわかる．

また，図 4.4，図 4.5，図 4.6 は，5 回目の試行の行列サイズ 200 のテスト行列を，それぞれの手法で分解した際に求められた上三角行列 R の対角成分をプロットしたものを示している．これらは，それぞれ正規化される前の直交ベクトルの大きさを表している．

図 4.4 は GS 法の正規化される前の直交ベクトルのノルムであり，ステップが進むにつれノルムが小さくなっている．一方，CSGS 法のノルムは，CGS 法のノルムと比べ異常に大きくなっていることがわかる．これにより，4.1.3 節で検討したことが発生し，J-直交性が CGS 法のそれに比べ崩れやすくなっていると考えられる．そして，MSGS 法のノルムも大きくなっているが，CSGS 法のそれよりは小さいので，CSGS 法よりも良い J-直交計算精度を持っていると考えることが出来る．

次に，4.1.3 節で述べた再直交化条件の有用性を検討する．テスト行列には上記と同様のものを使う．図 4.5 と図 4.6 より，J-直交化されるベクトルのノルムは増大していくことがわかったので，GS 法の再直交化条件は常に満たされると予想される．よって，GS 法の再直交化条件だけでは J-直交性が崩れるようなベクトルが計算されたとしても再直交化は行われない．そこでこれまでの条件に加えて，4.1.3 節で提案した式 (4.27) を用いて再直交化をする CSGS 法と MSGS 法を用いて SR 分解を行った．結果を図 4.7 と図 4.8 に示す．

図 4.7 より，再直交化付きの CSGS 法は図 4.2 と比べて，非常に計算精度が高くなっていることがわかる．一方で，図 4.8 より，再直交化付きの MSGS 法は図 4.3 と比べて，

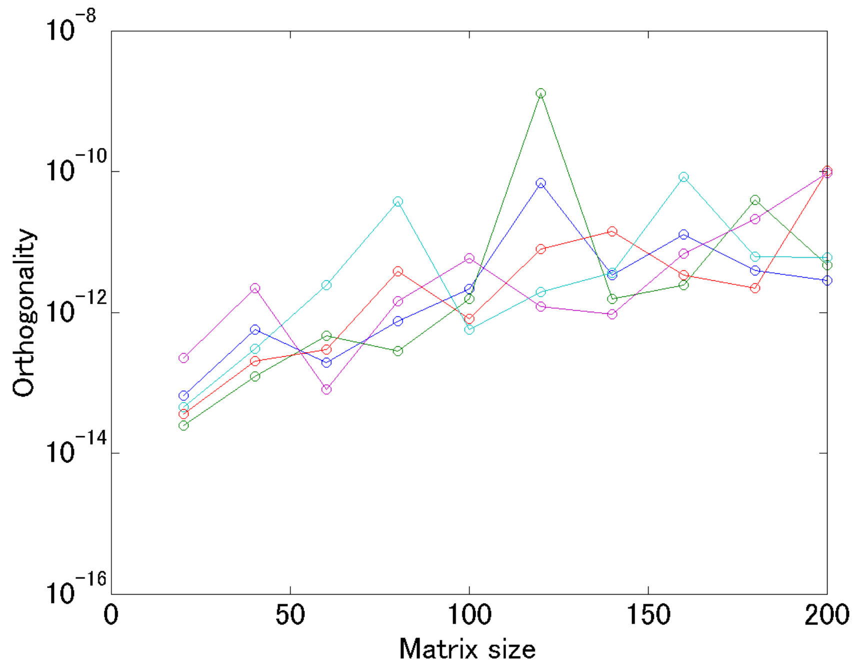


図 4.1　CGS 法の直交行列の計算誤差

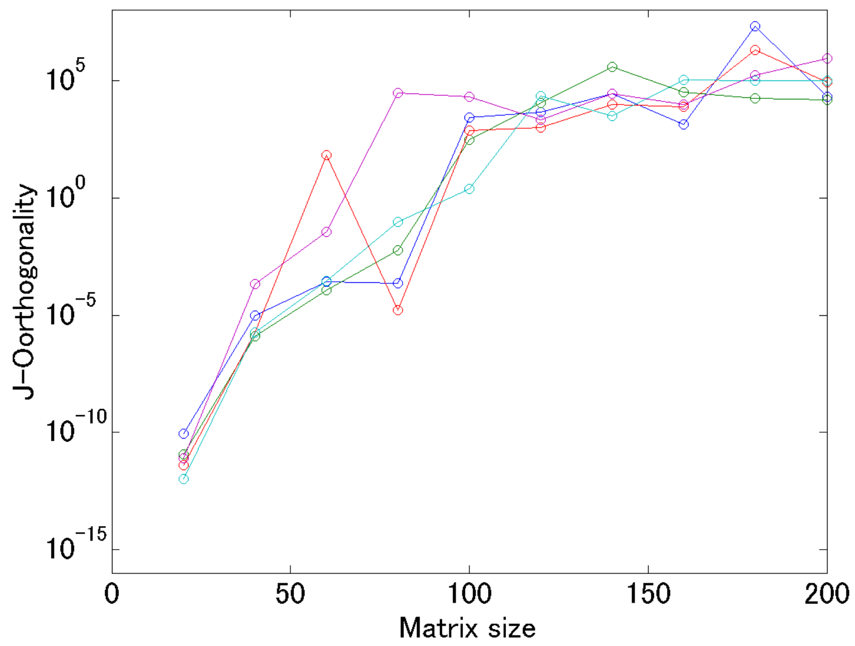


図 4.2　CSGS 法の J-直交行列の計算誤差

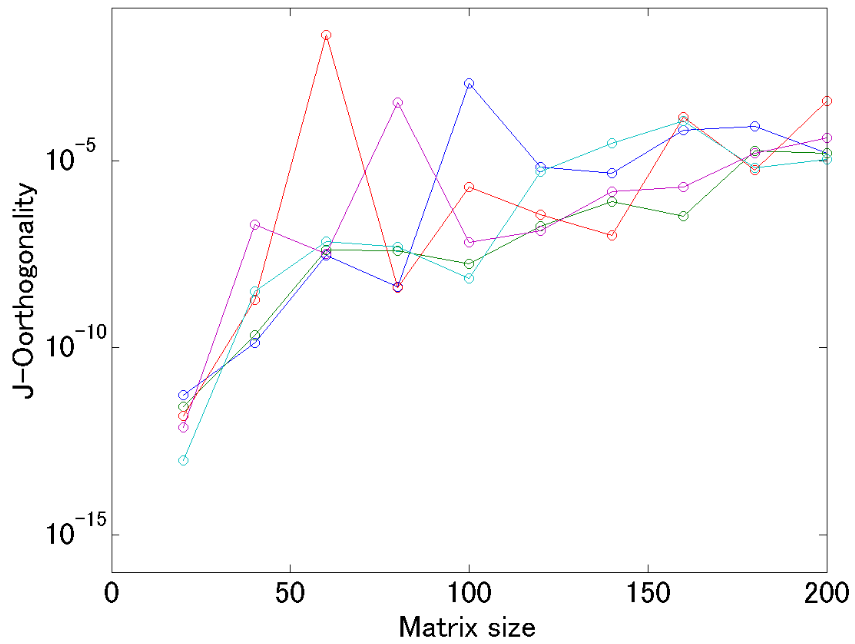


図 4.3　MSGS 法の J-直交行列の計算誤差

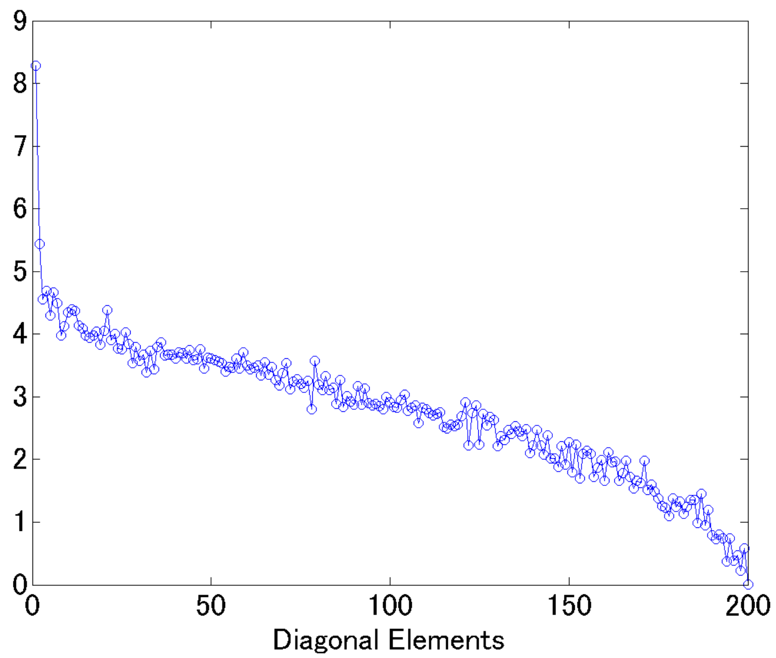


図 4.4　CGS 法により生成された直交ベクトルのノルムの大きさ

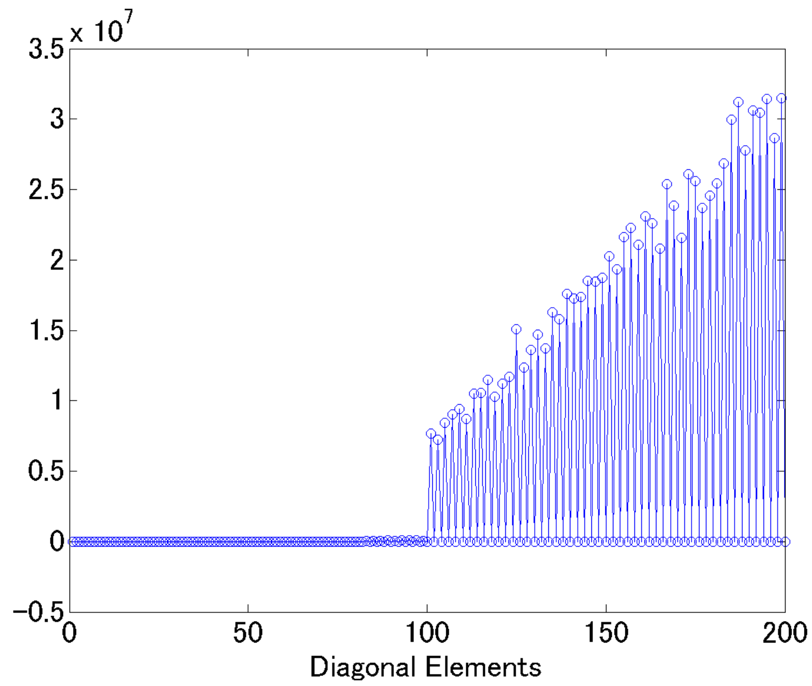


図 4.5　CSGS 法により生成された J-直交ベクトルのノルムの大きさ

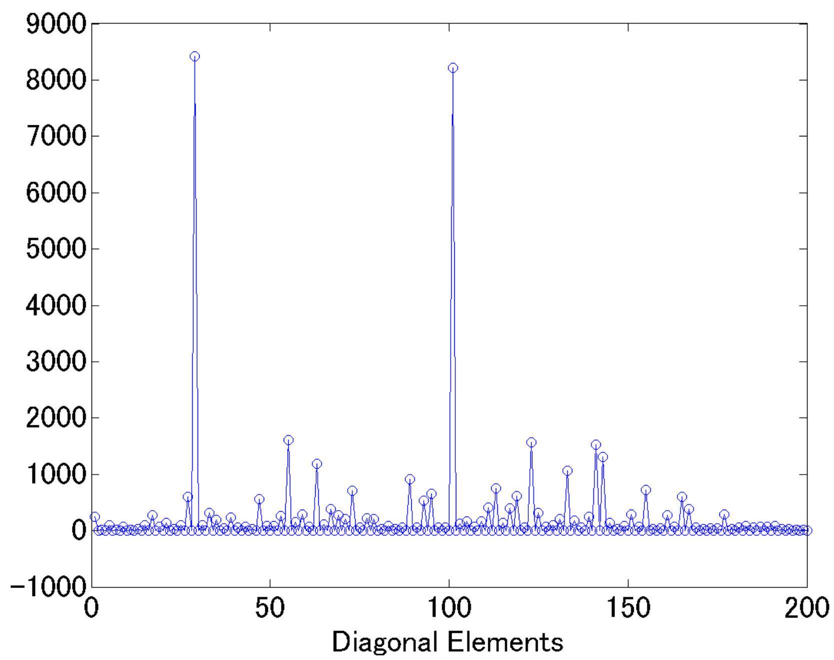


図 4.6　MSGS 法により生成された J-直交ベクトルのノルムの大きさ

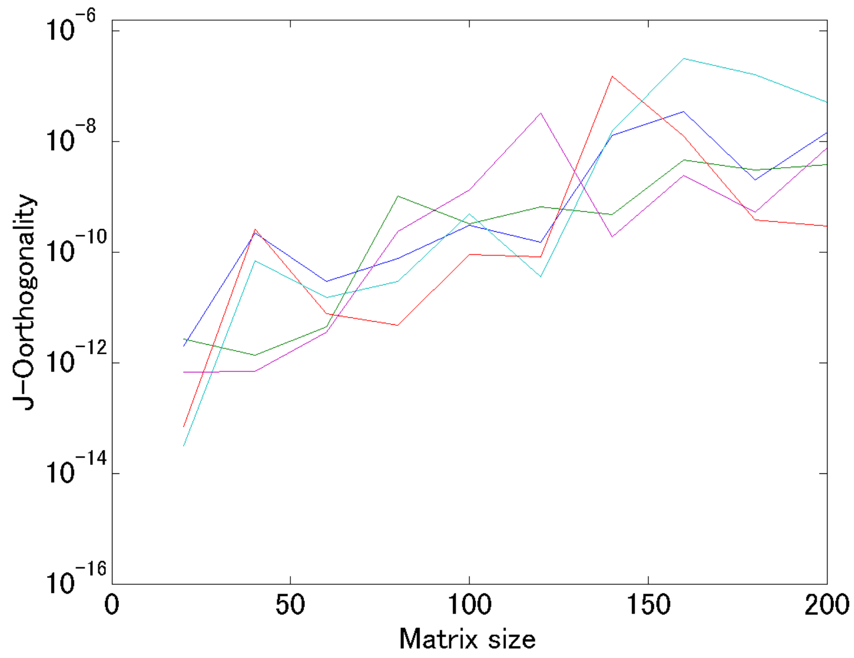


図 4.7 CSGS 法による J-直交行列の計算誤差の様子

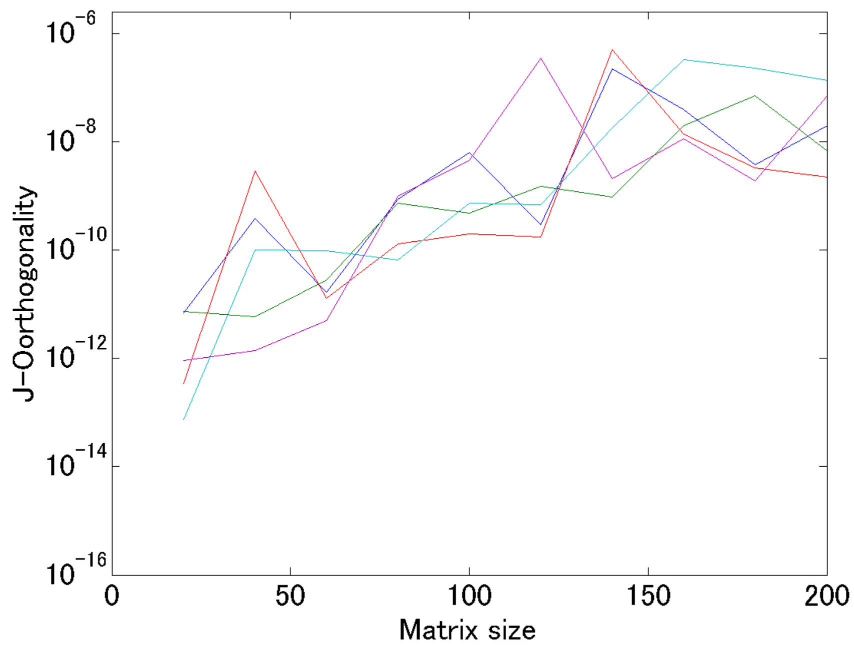


図 4.8 MSGS 法による J-直交行列の計算誤差の様子

表 4.1　CSGS 法，MSGS 法，BSGS 法の比較：Hamiltonian 行列 1

Method	m	t_m	Accuracy
CSGS	2	0.104	8.70e-06
MSGS	2	0.107	4.65e-06
BSGS	10	0.039	2.80e-06

表 4.2　CSGS 法，MSGS 法，BSGS 法の比較：Hamiltonian 行列 2

Method	m	t_m	Accuracy
CSGS	2	21.24	1.55e-04
MSGS	2	21.40	1.03e-04
BSGS	40	2.50	3.74e-05
BSGS	100	3.21	7.14e-05
BSGS-m	72	2.72	4.48e-05

計算精度は高くない．また，どちらの手法も GS 法と比べて，未だに同程度の精度は得られていないことがわかる．これらの結果より，CSGS 法は著しく J-直交精度が低くなりやすいので，式 (4.27) は有用であるが，より厳しい条件が必要であるといえる．

4.4.2　数値実験 2

本節では，CSGS 法のブロック化と，ブロックサイズの決定手法の有効性を数値実験を用いて検討する．CSGS 法，MSGS 法，BSGS 法を用いて 1.4.4 節の数値例 4 の行列を SR 分解し，計算時間と式 (4.22) を用いて J-直交精度を計算した．ただし，全ての手法において再直交化を一度行った．ここで，BSGS-m は適応的なブロックサイズの決定手法を用いた BSGS 法を表す．数値実験の結果を表 4.1 と表 4.2 に示す．

表 4.1 より，BSGS 法は他の 2 つの手法と比べて，SR 分解を約 2 倍程度高速に計算することが出来ることがわかる．これはブロック化することにより，J-直交行列を計算に使用する回数が減り，BLAS を用いることで高速に計算が出来るためであると考えられる．また，4.2 節で述べたように，CSGS 法と比べ，わずかではあるが J-直交行列の計算精度が向上している．しかしながら表 4.1 では，MSGS 法に対しても BSGS 法の J-直交計算精度が高くなっている．

表 4.2 より，同様に BSGS 法は他の 2 つの手法と比べて，約 10 倍程度高速になっている．特に表 4.1 と比べて，CSGS 法との速度比が約 5 倍程度大きくなっている．これは扱う行列サイズが大きくなり，BLAS によるパフォーマンスの向上が大きくなったためである．また，BSGS-m 法は，BSGS 法と比べて約 90 %程度の速度比が実現できている．また，J-直交性の計算精度については，表 4.1 と比べて，より大きい幅で精度が向上していることがわかる．これは，4.4.1 節からわかるように，行列サイズが大きい問題の方が，J-直交行列の計算が不安定であるためと考えられる．

4.5 まとめ

本章では，SGS 法と GS 法の違い，J-直交性の崩れについて述べたあと，再直交化条件の検討を行った．そして，CSGS 法のブロック化を提案し，ブロックサイズの決定手法を適用した．

GS 法と異なり，SGS 法は行列 J の影響によって，加算と減算を繰り返すため丸め誤差の影響を大きく受ける可能性があること，また，式 (4.25) より，$\boldsymbol{s}_{2i}$ が X の列ベクトルに比べて非常に大きくなることがあることがわかり，これらにより，直交性が崩れる可能性があることを示した．そして，数値実験を用いて実際に J-直交性が崩れやすいことを確認した．特に CSGS 法では，200 次元程度のサイズの行列の SR 分解でさえも，まったく正確に計算できていないことがわかった．

さらに CSGS 法においては，直交化されたノルムが非常に大きくなることを確認した．これらのことより，CSGS 法においては，再直交化が非常に重要であることがわった．GS 法には，再直交化の適用に関する判定条件が存在しているが，それをそのまま SGS 法に適用しても，直交化されたベクトルノルムが大きくなるので，有用でないことを示した．そこで，本章では新しい再直交化の条件を提案し，数値実験を行った．4.4.1 節の数値実験の結果より，CSGS 法，MSGS 法に対して，この再直交化条件が有用であることを示した．

また，CSGS 法を高速化するために，CSGS 法のブロック化とブロックサイズ m の決定法を提案した．CSGS 法は，CGS 法と算法が似ているため，容易にブロック化することが可能であった．また，ブロック化することにより，CSGS 法と比べて，計算順序が多少入れ替わることがわかった．これにより，わずかではあるが，BSGS 法の方が計算され

た J-直交行列の計算精度が向上する可能性があることを示した．そして，4.4.2 節の数値実験により，ブロック化することで CSGS 法を高速化し，計算精度も向上したことを検証し，提案手法の有効性を示した．また，ブロックサイズの決定手法を適用することで，適応的にブロックサイズを決定することができた．BSGS 法と比べ，BSGS-m 法は約 90 %程度の速度比を実現しており，提案手法の有効性を示した．

第 5 章

総括と結論

本論文では，3 つの Gram-Schmidt(GS) 法について，ブロック化と適応的なブロックサイズ m の決定法とその実装について提案した．演算を高速に処理できるようにするため，様々な Gram-Schmidt 法に対して，ブロック化することは重要である．そこで本論文では，Symplectic Gram-Schmidt(SGS) 法のブロック化の提案と，Block Gram-Schmidt(BGS) 法，Parallel Block Gram-Schmidt(PBGS) 法に対して，ブロックサイズを適応的に決定する手法を提案した．

2 章では BGS 法について考察した．BGS は Classical Gram-Schmidt(CGS) 法において，1 列ベクトルごとに直交化していた部分を，m 列のブロックベクトルごとに一度に直交化する算法である．それにより，BLAS などの数値計算ライブラリのパフォーマンスを向上させることができ，高速に数値計算を行うことができるようになる．

しかし，行列 X を分割するブロックサイズ m の値の取り方によって計算速度が変わるので，ユーザーが問題ごとにブロックサイズ m を検討し，適宜決定する必要がある．そこで本論文では，計算時間の増加と計算量の増加に着目した．ブロックサイズ m を固定して，BGS 法を行うと，各ステップにかかる計算時間が線形に増加していくことが，計算量と数値実験により確認できた．

そこで，この計算量と計算時間の関係を用いて，ブロックサイズごとの BGS 法の計算実行時間を予測することによって，適応的にブロックサイズ m を決定する 3 つの手法 Scheme A, B, C を提案した．そして，数値実験において Scheme A, B, C を用いて，様々なサイズの問題を計算した．その結果，Scheme A, B は適応的にブロックサイズ m を決定することができず，どの問題においても似たような値を示した．一方で，Scheme C は問題ごとに適応的にブロックサイズ m を決定し，最速な場合と比較して，90 %程度の速度比で計算を行えることが示された．

3 章では，BGS 法の並列化と PBGS 法のブロックサイズの決定手法について述べた．

PBGS 法の分散手法には様々な手法があるが，本論文では列方向分散を用いた．その理由は，列分散手法が列ごとにブロック化していく BGS 法に適応させやすいからである．PBGS 法は，BGS 法の場合と同様に，ブロックサイズ m によって計算時間が異なるため，適応的なブロックサイズを決定する手法を提案した．BGS 法と比べて，分散処理が加わっているものの計算回数は同じであると考え，そのまま PBGS 法に適用させた．そして，数値実験において，並列化することで高速化できることと，ブロックサイズ m を自動的に決定出来ることを示し，PBGS 法に対しても，ブロックサイズ m の決定手法が有効であることを示した．並列化することで，各プロセッサがそれぞれ BGS 法において最速となるブロックサイズの行列を扱うため，並列化した手法では最適なブロックサイズ m は BGS 法と比べ大きくなった．特に，並列化による速度向上率がプロセッサ数に比例しているところでは，最適なブロックサイズ m も比例した値に近いサイズとなった．このことからも，ブロックサイズ m の決定法が適応的にブロックサイズを決定できることを示した．また，このブロックサイズ m を決定する提案手法が，1 ステップの計算時間を測る部分の手法を変更するだけで，他の手法にも適用させやすいことも示した．

4 章では，CSGS 法のブロック化を提案し，ブロックサイズの決定手法を適用した．SGS 法は GS 法と異なり，行列 J の影響によって，加算と減算を繰り返すため丸め誤差の影響を大きく受ける可能性があること，直交化される列ベクトルのノルムと比べて，計算された J-直交ベクトルのノルムが大きくなることがあり，これによっても直交性が乱れる可能性があることを示した．また，数値実験により，実際に J-直交性が乱れやすいことを確認した．特に CSGS 法では，小規模な問題でさえも，正確に計算できないことを示した．

さらに，CSGS 法を高速化するために，CSGS 法のブロック化とブロックサイズの決定手法を提案した．CSGS 法は CGS 法とアルゴリズムが似ているため，容易にブロック化することが可能であること，ブロック化することにより，CSGS 法と比べて，計算順序が多少入れ替わること，これにより，わずかではあるが BSGS 法の方が計算された J-直交行列の J-直交性が向上する可能性があることを示した．4.4.2 節の数値実験により，ブロック化することで CSGS 法を高速化し，計算精度も向上したことを検証し，提案手法の有効性を示した．また，ブロックサイズ m の決定手法を適用した結果，適応的にブロックサイズ m を決定することができた．BSGS 法と比べ，ブロックサイズ m をユーザーが決定することなく，BSGS-m 法は最速な場合と比べて約 90 %程度の速度で計算を行うことができ，提案手法の有効性を示した．

以上，本論文では，BGS 法に対して適応的なブロックサイズ m の決定手法，PBGS 法に対して適応的なブロックサイズ m の決定手法，SGS 法に対してブロック化と適応的なブロックサイズ m の決定手法を提案し，その有効性を示した．

謝 辞

本論文の作成にあたり，慶應義塾大学理工学部の野寺 隆教授には，研究や論文執筆についてさまざまなアドバイスをいただき，大変お世話になりました．また，同大学同学部の谷 温之名誉教授，萩原 将文教授，田村 明久教授，小田 芳彰准教授にはお忙しい中，副査として本論文を査読していただき，論文執筆に関して有益なご指摘をいただきました．そして，野寺研究室のみなさんのおかげでとても充実した生活を送ることができました．この場を借りて心より感謝申し上げます．

参 考 文 献

[1] Ammar, G. and Benner, P.: On Hamiltonian and Symplectic Hessenberg Forms, *Linear Algebra Appl.*, Vol. 149, No. 1, pp. 55–72 (1991).

[2] Arnoldi, W. E.: The Principle of Minimized Iteration in the Solution of the Matrix Eigenvalue Problem, *Quarterly of Applied Mathematics*, Vol. 9, pp. 17–29 (1951).

[3] Benner, P. and Fassbender, H.: An Implicitly Restarted Symplectic Lanczos Method for the Hamiltonian Eigenvalue Problem, *Linear Algebra Appl.*, Vol. 263, pp. 75–111 (1997).

[4] Björck, A.: Solving Linear Least Squares Problems by Gram-Schmidt Orthogonalization, *BIT*, Vol. 7, pp. 1–21 (1967).

[5] Boisvert, R.: A Web Resource for Test Matrix Collections, Townmeeting on Online NIST Reference Data, Gaithersburg, MD. (1977
). http://math.nist.gov/MatrixMarket/.

[6] Bunse-Gerstner, A. and Mehrmann, V.: A Symplectic QR Like Algorithm for the Solution of the Real Algebraic Riccati Equation, *IEEE Trans. Autom. Control*, Vol. AC31, pp. 1104–1113 (1986).

[7] Elden, L. and Park, H.: Block Downdating of Least Squares Solutions, *SIAM J. Matrix Anal. Appl.*, Vol. 15, pp. 1018–1034 (1994).

[8] Gudula, R. and Michael, S.: Comparison of Different Parallel Modified Gram-Schmidt Algorithm, *Euro-Par 2005, LNCS*, Vol. 3648, pp. 826–836 (2005).

[9] Katagiri, T.: Performance Evaluation of Parallel Gram-Schmidt Reorthogonalization Methods, *VECPAR 2002, LNCS*, Vol. 2565, pp. 302–314 (2003).

[10] Loan, C. V.: A Symplectic Method for Approximating All the Eigenvalues of a Hamiltonian Matrix, *Linear Algebra Appl.*, Vol. 61, pp. 223–251 (1984).

[11] Matsuo, Y., Guo, H. and Arbenz, P.: Experiments on a Parallel Nonlinear Jacobi-Davidson Algorithm, *Procedia Computer Science*, Vol. 29, pp. 566–575 (2014).

[12] Matsuo, Y. and Nodera, T.: The Oprimal Block-Size for the Block Gram-Schmidt Orthogonalization, *J. Sci. Tech.*, Vol. 49, pp. 569–584 (2011).

[13] Matsuo, Y. and Nodera, T.: An Efficient Implementation of the Block Gram–Schmidt Method, *Anziam J.*, Vol. 54, pp. C394–409 (2013).

[14] 松尾洋一，野寺　隆：Block Gram-Schmidt 法の適切な block-size の決定法，情報処理学会第 74 回全国大会, 講演番号 1K-1 (2012).

[15] 松尾洋一，野寺　隆 : 適切な block-size による Block Gram-Schmidt 法の並列化，研究報告ハイパフォーマンスコンピューティング，Vol. 2012-HPC-134(2), pp. 1–4 (2012).

[16] Moriya, K. and Nodera, T.: The DEFLATED-GMRES(m, k)Method with Switching the Restart Frequency Dynamically, *Numer. Linear Algebra Appl.*, Vol. 7, pp. 569–584 (2000).

[17] Moriya, K. and Nodera, T.: Usage of the Convergence Test of the Residual Norm in the GMRES($\leq m_{\max}$) Algorithm, *ANZIAM J.*, Vol. 49, pp. 293–308 (2007).

[18] Qiaohua, L.: Modified Gram-Schmidt-based Methods for Downdating the Cholesky Factorization, *J. Comput. Appl. Math.*, Vol. 235, pp. 1897–1905 (2011).

[19] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, SIAM Second Editiono (2003).

[20] Saad, Y. and Schultz, M. H.: GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. Sci. Stat. Comput.*, Vol. 7, pp. 856–869 (1986).

[21] Salam, A.: On Theorerical and Numerical Aspects of Symplectic Gram-Schmid-like Algorithms, *Numer. Algo.*, Vol. 39, pp. 437–462 (2005).

[22] Salam, A. and Al-Aidarous, E.: Equivalence Between Modified Symplectic Gram-Schmidt and Householder SR Algorithms, *BIT Numer. Math.*, Vol. 54, pp. 283–302 (2014).

[23] Shiroishi, J. and Nodera, T.: A GMRES(m) Method with Two Stage Deflated Preconditioners, *ANZIAM J.*, Vol. 52, pp. C222–C236 (2011).

[24] Sleijpen, G. L. G. and van der Vorst, H. A.: A Jacobi–Davidson iteration method for linear eigenvalue problems, *SIAM Rev.*, Vol. 40, pp. 267–293 (2000).

[25] Steven, J. L., Ake, B. and Walter, G.: Gram-Schmidt Orthogonalization: 100 years and more, *Numer. Linear Algebra Appl.*, Vol. 20, pp. 492–532 (2013).

[26] Stewart, G. W.: The Effect of Rounding Errors on an Algorithm for Downdating a Cholesky Factorization, *J. Inst. Math. Appl.*, Vol. 23, pp. 203–213 (1979).

[27] Stewart, G. W.: Block Gram-Schmidt Orthogonalization, *SIAM J. Sci. Comput.*, Vol. 31, pp. 761–775 (2008).

[28] Teramoto, K. and Nodera, T.: A Note on Lanczos Algorithm for Computing PageRank, *CMCGS 2014*, pp. 12–15 (2014).

[29] Vanderstraeten, D.: An Accurae Parallel Block Gram-Schmidt Algorithm without Reorthogonalization, *Numer. Linear Algebra Appl.*, Vol. 7, pp. 302–314 (2000).

[30] Voss, H.: A Jacobi–Davidson Method for Nonlinear and Nonsymmetric Eigenproblems, *Comput. Struct.*, Vol. 85, No. 17-18, pp. 1284–1292 (2007).

[31] Yokozawa, T., Takahashi, D., Boku, T. and Sato, M.: Parallel Implementation of Classical Gram-Schmidt Orthogonalization Using Matrix Multiplication, 情報処理学会研究報告ハイパフォーマンスコンピューティング, Vol. 2006-HPC-106, pp. 31–36 (2006).

[32] Yoo, K. and Park, H.: Accurate Downdating of a Modified Gram-Schmidt OR Decomposition, *BIT*, Vol. 36, pp. 223–251 (1996).

付 録 A

Block Gram-Schmidt法

Scheme C

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include<sys/time.h>
#include<string.h>

double gettimeofday_sec();
void input_size(int *pN,int *pP, FILE *fin);
void input_matrix(double **X, FILE *fin);
double **dmatrix(int nr,int nl);
void free_dmatrix(double **a,int nr,int nl);
double *dvector(int i);
void free_dvector(double *a);
void mnorm(double *A,int nr,int nl,double *nu);
double norm(int N,double *y);
double QRSD(double *Q,int N,int P);
double XRSD(double *Q,int N,int P,double *R,double *X);
void Bgssro(double *X,double *Q,double *R,int N,int P,int m);
void gss(double *Y,int N,int m,double *R22);
void gssro(double *Y,int N,int m,double *R22);
```

```
void bgssro_m(int N,int P,int m,int h,double *X,double *Q,double *R);
void decide_m(int N,int P,int *m,int *h,double *X,double *Q,double *R);

int main()
{
  int N,P,*pN,*pP,m,i,j,jN;
  double qrsd,xrsd,start,end;
  FILE *fin;
  pN=&N;pP=&P;
  m=4;
  start=gettimeofday_sec();
  if((fin = fopen("filename","r"))==NULL)
    {
      printf("no such file\n");
      exit(1);
    }
  input_size(pN,pP,fin);
  double **A,*X,*Q,*R;
  A=dmatrix(N,P);
  X=dvector(N*P);
  Q=dvector(N*P);
  R=dvector(P*P);
  for(i=0;i<N;++i){
    for(j=0;j<P;++j)
      A[i][j]=0.0;
  }
  input_matrix(A,fin);
  fclose(fin);
  /*
  int N=5000,P=5000,*pN,*pP,m,i,j,jN;
```

```
double qrsd,xrsd,start,end;
FILE *fin;
pN=&N;pP=&P;
m=1;
double **A,*X,*Q,*R;
A=dmatrix(N,P);
X=dvector(N*P);
Q=dvector(N*P);
R=dvector(P*P);
for(i=0;i<N;++i){
  for(j=0;j<P;++j)
    A[i][j]=(double)rand()/2147483647;
}
*/
for(j=0;j<P;++j){
  jN=j*N;
  for(i=0;i<N;++i)
    X[i+jN]=A[i][j];
}
free_dmatrix(A,N,P);
/*blockgramschmidt*/
start=gettimeofday_sec();
Bgssro(X,Q,R,N,P,m);
end=gettimeofday_sec();
/*result*/
printf("%10.4f\n",end-start);
/*
qrsd=QRSD(Q,N,P);
printf("qrsd=%4.2e\n",qrsd);
xrsd=XRSD(Q,N,P,R,X);
```

```
  printf("xrsd=%4.2e\n",xrsd);
  */
  free_dvector(X);
  free_dvector(Q);
  free_dvector(R);
}

void input_size(int *pN, int *pP, FILE *fin)
{
  fscanf(fin, "%d", pN);
  fscanf(fin, "%d", pP);
}

void input_matrix(double **X, FILE *fin)
{
  int i,j,k,l;
  fscanf(fin, "%d", &k);
  for(l=0;l<k;++l){
    fscanf(fin, "%d",&i);
    fscanf(fin, "%d",&j);
    fscanf(fin, "%lf",&X[i-1][j-1]);
    //   printf("X[%d][%d]=%5.5e\n", i, j, X[i-1][j-1]);
  }
}

double **dmatrix(int nr,int nl)
{
  double **a;
  int i;
  if((a=(double **)malloc(nr*sizeof(double *)))==NULL)
```

```
    {
      printf("no free memory\n");
      exit(1);
    }
  for(i=0;i<nr;++i)
    a[i]=(double *)malloc(nl*sizeof(double));

  return (a);
}

void free_dmatrix(double **a,int nr,int nl)
{
  int i;
  for(i=0;i<nr;++i)
    free((void *)(a[i]));
  free((void *)(a));
}

double *dvector(int i)
{
  double *a;
  if((a=(double *)malloc((i)*sizeof(double)))==NULL)
    {
      printf("no free memory\n");
      exit(1);
    }
  return(a);
}

void free_dvector(double *a)
```

```
{
  free((void *)(a));
}

void mnorm(double *A,int nr,int nl,double *nu)
{
  int i,j,cn;
  double s;
  for(j=0;j<nl;++j){
    s=0.0;
    cn=nr*j;
    for(i=0;i<nr;++i)
      s=A[i+cn]*A[i+cn]+s;
    nu[j]=s;
  }
}

double QRSD(double *Q,int N,int P)
{
  char transf[]="T",transc[]="N";
  double *Qrsd,s,max,alpha=1.0,beta=0.0;
  int i,j,jP;
  max=0.0;
  Qrsd=dvector(P*P);
  dgemm_(transf,transc,&P,&P,&N,&alpha,Q,&N,Q,&N,&beta,Qrsd,&P);
  for(j=0;j<P;++j){
    jP=P*j;
    for(i=0;i<P;++i){
      if(i!=j)
s=Qrsd[i+jP];
```

```
      else{
s=Qrsd[i+jP]-1.0;
      }
      if(s<0.0)
s=-s;
      if(max<s)
max=s;
    }
  }
  return max;
}

double XRSD(double *Q,int N,int P,double *R,double *X)
{
  double *Xrsd,s,max,alpha=1.0,beta=0.0;
  char transf[]="N",transc[]="N";
  int i,j,NP;
  max=0.0;
  Xrsd=dvector(N*P);
  dgemm_(transf,transc,&N,&P,&P,&alpha,Q,&N,R,&P,&beta,Xrsd,&N);
  NP=N*P;
  for(i=0;i<NP;++i){
    s=X[i]-Xrsd[i];
    if(s<0.0)
      s=-s;
    if(max<s)
      max=s;
  }
  return max;
}
```

```
void Bgssro(double *X,double *Q,double *R,int N,int P,int m)
{
  double *Y1,*R221,st,en,aa,bb,KK,cc,dd;
  int *mm,*hh,h,k,i,j,Rem,Nm,jP,jm,hN,M,l,K,jN,PP,mi;
  double *mt,bmt,gamma,dN,dP,dm,pp;
  m=10;
  h=m;
  hh=&h;
  mm=&m;
  //  Rem=P%m;
  l=100;
  Y1=dvector(N*m);
  R221=dvector(m*m);
  PP=P*P;
  for(i=0;i<PP;++i)
    R[i]=0.0;
  Nm=N*m;
  /*0 - m-1colum*/
  st=gettimeofday_sec();
  for(i=0;i<Nm;++i)
    Y1[i]=X[i];
  gssro(Y1,N,m,R221);
  for(i=0;i<Nm;++i)
    Q[i]=Y1[i];
  for(j=0;j<m;++j){
    jP=j*P;
    jm=j*m;
    for(i=0;i<m;++i)
      R[i+jP]=R221[i+jm];
```

```
  }
  free_dvector(Y1);
  free_dvector(R221);
  /*decide m*/
  decide_m(N,P,mm,hh,X,Q,R);
  K=(P-h)/m;
  Rem=(P-h)%m;
  printf("m=%d\n",m);
  /*km - (k+1)m colum*/
  for(k=0;k<K;++k){
    bgssro_m(N,P,m,h,X,Q,R);
    h=h+m;
    if(h>=l){
      //      printf("%d colum\n",h);
      en=gettimeofday_sec();
      //      printf("%10.4f\n",en-st);
      st=gettimeofday_sec();
      l=l+100;
    }
  }
  /*rem colum*/
  M=Rem;
  bgssro_m(N,P,M,h,X,Q,R);
  h=h+M;
}

void gss(double *Y,int N,int m,double *R22)
{
  int k,i,j,km,Nk,incx=1;
  double *y,*r,*z,*s,nu,alpha=1.0,beta=0.0;
```

```
char transc[]="N";
for(i=0;i<m*m;++i)
  R22[i]=0.0;
y=dvector(N);
r=dvector(N);
z=dvector(N);
for(i=0;i<N;++i)
  y[i]=Y[i];
nu=norm(N,y);
R22[0]=nu;
for(i=0;i<N;++i)
  Y[i]=y[i]/nu;
for(k=1;k<m;++k){
  km=k*m;
  Nk=N*k;
  for(i=0;i<N;++i)
    y[i]=Y[i+Nk];
  transc[0]='T';
  dgemv_(transc,&N,&k,&alpha,Y,&N,y,&incx,&beta,r,&incx);
  transc[0]='N';
  dgemv_(transc,&N,&k,&alpha,Y,&N,r,&incx,&beta,z,&incx);
  for(i=0;i<N;++i)
    y[i]=y[i]-z[i];
  nu=norm(N,y);
  for(i=0;i<N;++i)
    Y[i+Nk]=y[i]/nu;
  for(i=0;i<k;++i)
    R22[km+i]=r[i];
  R22[km+k]=nu;
}
```

```
    free_dvector(y);
    free_dvector(r);
    free_dvector(z);
}

void gssro(double *Y,int N,int m,double *R22)
{
  int k,i,j,km,Nk,incx=1,l;
  double *y,*r,*z,*s,nu,nu1,alpha=1.0,beta=0.0;
  char transc[]="N";
  y=dvector(N);
  r=dvector(N);
  s=dvector(N);
  z=dvector(N);
  for(i=0.0;i<m*m;++i)
    R22[i]=0.0;
  for(i=0;i<N;++i)
    y[i]=Y[i];
  nu=norm(N,y);
  R22[0]=nu;
  for(i=0;i<N;++i)
    Y[i]=y[i]/nu;
  for(k=1;k<m;++k){
    km=k*m;
    Nk=N*k;
    for(i=0;i<N;++i){
      y[i]=Y[i+Nk];
    }
    nu=norm(N,y);
    for(i=0;i<k;++i)
```

```
      s[i]=0.0;
    while(1){
      transc[0]='T';
      dgemv_(transc,&N,&k,&alpha,Y,&N,y,&incx,&beta,r,&incx);
      transc[0]='N';
      for(i=0;i<k;++i)
s[i]=s[i]+r[i];
      dgemv_(transc,&N,&k,&alpha,Y,&N,r,&incx,&beta,z,&incx);
      for(i=0;i<N;++i)
y[i]=y[i]-z[i];
      nu1=norm(N,y);
      if(nu1>0.5*nu)
break;
      else
nu=nu1;
    }
    for(i=0;i<N;++i)
      Y[i+Nk]=y[i]/nu1;
    for(i=0;i<k;++i)
      R22[km+i]=s[i];
    R22[km+k]=nu1;
  }
  /*
    free_dvector(y);
    free_dvector(r);
    free_dvector(s);
    free_dvector(z);
  */
}
```

```
double norm(int N,double *y)
{
  int i;
  double s;
  s=0.0;
  for(i=0;i<N;++i)
    s=y[i]*y[i]+s;
  s=sqrt(s);
  return s;
}

double gettimeofday_sec()
{
  struct timeval tv;
  gettimeofday(&tv,NULL);
  return tv.tv_sec + (double)tv.tv_usec*1e-6;
}

void bgssro_m(int N,int P,int m,int h,double *X,double *Q,double *R)
{
  int i,j,hN,Nm,Ph,p,jm,jN,hj,Pj;
  char transf[]="N",transc[]="N";
  double *nu,*W22,alpha=1.0,beta=0.0,*Y,*Z,*R12,*R22,*S12,*S22;
  Y=dvector(N*m);
  Z=dvector(N*m);
  R12=dvector(P*m);
  R22=dvector(m*m);
  S12=dvector(P*m);
  S22=dvector(m*m);
  hN=h*N;
```

```
  Nm=N*m;
  Ph=P*h;
  for(j=0;j<m;++j){
    jN=j*N;
    for(i=0;i<N;++i){
      Y[i+jN]=X[i+jN+hN];
    }
  }
  nu=dvector(m);
  W22=dvector(m*m);
  mnorm(Y,N,m,nu);
  transf[0]='T';
  dgemm_(transf,transc,&h,&m,&N,&alpha,Q,&N,Y,&N,&beta,R12,&h);
  transf[0]='N';
  dgemm_(transf,transc,&N,&m,&h,&alpha,Q,&N,R12,&h,&beta,Z,&N);
  for(i=0;i<Nm;++i)
    Y[i]=Y[i]-Z[i];
  gss(Y,N,m,R22);
  p=0;
  for(j=0;j<m;++j){
    jm=j*m;
    if(R22[jm+j]<0.5*nu[j])
p=1;
  }
  if(p==1){
    transf[0]='T';
    dgemm_(transf,transc,&h,&m,&N,&alpha,Q,&N,Y,&N,&beta,S12,&h);
    transf[0]='N';
    dgemm_(transf,transc,&N,&m,&h,&alpha,Q,&N,S12,&h,&beta,Z,&N);
    for(i=0;i<Nm;++i)
```

```
      Y[i]=Y[i]-Z[i];
    gssro(Y,N,m,S22);
    beta=1.0;
    dgemm_(transf,transc,&h,&m,&m,&alpha,S12,&h,R22,&m,&beta,R12,&h);
    beta=0.0;
    dgemm_(transf,transc,&m,&m,&m,&alpha,S22,&m,R22,&m,&beta,W22,&m);
    for(i=0;i<m*m;++i)
      R22[i]=W22[i];
  }
  for(j=0;j<m;++j){
    jN=j*N;
    for(i=0;i<N;++i)
      Q[hN+jN+i]=Y[i+jN];
  }
  for(j=0;j<m;++j){
     hj=h*j;
     Pj=P*j;
      for(i=0;i<h;++i)
R[Ph+i+Pj]=R12[i+hj];
    }
    for(j=0;j<m;++j){
      jm=j*m;
      Pj=P*j;
      for(i=0;i<m;++i)
R[Ph+h+i+Pj]=R22[i+jm];
    }
}

void decide_m(int N,int P,int *m,int *h,double *X,double *Q,double *R)
{
```

```
double aa,bb,cc,dd;
double dm,dN;
double mt1,mt2,s,ss;
double *A,*w,*b,*work;
int l,i,j,pp,m1,g,bcol,iwork=g*2,info,acol;
char trans='N';
g=5;
bcol=1;
A=dvector(g*5);
b=dvector(g);
acol=5;
dN=(double)N;
*m=0;
for(i=0;i<g;++i){
  m1=2;
  bgssro_m(N,P,m1,*h,X,Q,R);
  *h=*h+m1;
  for(l=0;l<i;++l)
    m1=m1*2;
  dm=(double)m1;
  for(j=0;j<2;++j){
  cc=gettimeofday_sec();
  bgssro_m(N,P,m1,*h,X,Q,R);
  dd=gettimeofday_sec();
  if(j==0)
    aa=dd-cc;
  if(j==1)
    bb=dd-cc;
  *h=*h+m1;
  }
```

```
    cc=(bb-aa)/dm;
    printf("cc=%f\n",cc);

    b[i]=(dN/dm-1.0)*((0.5*cc*dN/dm)+2.0*aa-bb);

    A[i]=dm*dm*dm*dm;
    A[i+g]=dm*dm*dm;
    A[i+(g*2)]=dm*dm;
    A[i+(g*3)]=dm;
    A[i+(g*4)]=1.0;
  }
  iwork=g+5;
  work=dvector(iwork);
  for(i=0;i<g;++i)
    printf("b[]=%4.2e\n",b[i]);
  dgels_(&trans,&g,&acol,&bcol,A,&g,b,&g,work,&iwork,&info);
  printf("\n");
  for(i=0;i<g;++i)
    printf("b[]=%4.2e\n",b[i]);
  double x1=0.0,x2,fx,dfx,d=1.0;
  double y1,y2,z1,z2,EPS=1.0e-10;
  int max=10;
  i=1;
  while(fabs(d) > EPS && i<max){
    fx=(4.0*b[0]*x1*x1*x1)+(3.0*b[1]*x1*x1)+(2.0*b[2]*x1)+b[3];
    dfx=(12.0*b[0]*x1*x1)+(6.0*b[1]*x1)+2.0*b[2];
    d=-(fx/dfx);
    x2=x1+d;
    x1=x2;
    printf("x2=%4.2e\n",x2);
```

```
    ++i;
  }
  if(i==max){
    printf("no1\n");
  }
  y1=x2;
  z1=(b[0]*x1*x1*x1*x1)+(b[1]*x1*x1*x1)+(b[2]*x1*x1)+(b[3]*x1)+b[4];
  printf("z1=%4.2e\n",z1);
  d=1.0;
  x1=100.0;
  i=1;
  while(fabs(d) > EPS && i<max){
    fx=(4.0*b[0]*x1*x1*x1)+(3.0*b[1]*x1*x1)+(2.0*b[2]*x1)+b[3];
    dfx=(12.0*b[0]*x1*x1)+(6.0*b[1]*x1)+(2.0*b[2]);
    d=-(fx/dfx);
    x2=x1+d;
    x1=x2;
    printf("x2=%4.2e\n",x2);
    printf("fx=%4.2e\n",fx);
    ++i;
  }
  if(i==max){
    printf("no2\n");
  }
  y2=x2;
  z2=(b[0]*x1*x1*x1*x1)+(b[1]*x1*x1*x1)+(b[2]*x1*x1)+(b[3]*x1)+b[4];
  printf("z2=%4.2e\n",z2);
  if(z1>=z2)
    dm=y2;
  else
```

```
    dm=y1;
  while((double)*m<dm+1)
    *m=*m+1;
  printf("m=%f\n",dm);
}
```

付 録 B

Parallel Block Gram-Schmidt法

Scheme D

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <math.h>
#include "mpi.h"
#define MSG_LEN 100

double gettimeofday_sec();
void input_size(int *pN, int *pP, FILE *fin);
void input_matrix(double **X, FILE *fin,int posi_s,int posi_e);
double **dmatrix(int nr,int nl);
void free_dmatrix(double **a,int nr,int nl);
double *dvector(int n);
void free_dvector(double *n);
void mnorm(double *A,int nr,int nl,double *nu);
double norm(int N,double *y);
double QRSD(double *Q,int N,int P);
void bgs(int N,int P,int p,int m,double *X,double *Q,int my_rank,
int process_num,int posi_s,int posi_e);
```

```
void gss(double *Y,int N,int m);
void bgs_m(int N,int P,int p,int m,int h,double *X,double *Q,
int my_rank,int process_num,int posi_s,int posi_e);
void decide_m(int N,int P,int p,int *m,int *h,double *X,double *Q,
int my_rank,int process_num,int posi_s,int posi_e)
void gssro(double *Y,int N,int m);
int main(int argc,char* argv[])
{
  int N,P,*pN,*pP,n,p,nre,pre,m,i,j,jN;
  double start,end,qrsd;
  FILE *fin;
  pN=&N;
  pP=&P;
  m=30;
  int my_rank;
  int process_num; /* プロセス数 */
  int posi_s,posi_e;
  /* 全ての MPI 関数を呼び出す前に一回だけ呼び出す */
  MPI_Init(&argc, &argv);
  /* 自分のランクを調べる */
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  /* プロセス数を調べる */
  MPI_Comm_size(MPI_COMM_WORLD, &process_num);
  if((fin = fopen("filename","r"))==NULL)
    {
      printf("no such file\n");
      exit(1);
    }
  input_size(pN,pP,fin);
  p=P/process_num;
```

```
pre=p+(P%process_num);
posi_s=my_rank*p;
if(my_rank!=(process_num-1)){
  posi_e=posi_s+p-1;
}
else{
  posi_e=posi_s+pre-1;
}
//  printf("posis=%d,posie=%d\n",posi_s,posi_e);
double **A,*X,*Q,*R;
if(my_rank!=(process_num-1)){
  A=dmatrix(N,p);
  X=dvector(N*p);
}
else{
  A=dmatrix(N,pre);
  X=dvector(N*pre);
}
for(i=0;i<N;++i){
  for(j=0;j<p;++j)
    A[i][j]=0.0;
}
/*
srand(time(NULL));
for(i=0;i<N;++i){
  for(j=0;j<p;++j)
    A[i][j]=(double)rand()/2147483647;
}
*/
input_matrix(A,fin,posi_s,posi_e);
```

```
  fclose(fin);
  if(my_rank!=(process_num-1)){
    for(j=0;j<p;++j){
      jN=j*N;
      for(i=0;i<N;++i)
X[i+jN]=A[i][j];
    }
  }
  else{
    for(j=0;j<pre;++j){
      jN=j*N;
      for(i=0;i<N;++i)
X[i+jN]=A[i][j];
    }
  }
  if(my_rank!=(process_num-1)){
    free_dmatrix(A,N,p);
  }
  else{
    free_dmatrix(A,N,pre);
  }
    Q=dvector(N*P);
  /*blockgramschmidt*/
  if(my_rank==0){
    start=gettimeofday_sec();
  }
  bgs(N,P,p,m,X,Q,my_rank,process_num,posi_s,posi_e);
  if(my_rank==0){
    end=gettimeofday_sec();
    /*result*/
```

```
    printf("%10.4f\n",end-start);
    qrsd=QRSD(Q,N,P);
    printf("qrsd=%4.2e\n",qrsd);
  }
  /*
    xrsd=XRSD(Q,N,P,R,X);
    printf("xrsd=%4.2e\n",xrsd);
  */
  free_dvector(X);
  free_dvector(Q);
  /* MPI 関数を呼び出した後, 最後に一回だけ呼び出す */
  MPI_Finalize();
  exit(EXIT_SUCCESS);
}

void bgs(int N,int P,int p,int m,double *X,double *Q,int my_rank,
int process_num,int posi_s,int posi_e)
{
  int tag = 0;
  double *Y1;
  int i,j,k,l,Nm,jP,jm,*mm,*hh,h;
  int K,Rem,point,*pointer;
  h=m;
  mm=&m;
  hh=&h;
  Nm=N*m;
  Y1=dvector(Nm);
  if(my_rank==0){
    for(i=0;i<Nm;++i){
      Y1[i]=X[i];
```

```
    }
    gssro(Y1,N,m);
  }
  MPI_Bcast(Y1,Nm,MPI_DOUBLE, 0, MPI_COMM_WORLD);
  for(i=0;i<Nm;++i)
    Q[i]=Y1[i];
  free_dvector(Y1);
  K=(P-h)/m;
  Rem=(P-h)%m;
  if(Rem<process_num){
    K=K-1;
    Rem=Rem+m;
  }
  if(my_rank==0){
  decide_m(N,P,mm,hh,X,Q,R);
  }
  MPI_Bcast(&m,1,MPI_INT,0,MPI_COMM_WORLD);
  l=100;
  for(k=0;k<K;++k){
    bgs_m(N,P,p,m,h,X,Q,my_rank,process_num,posi_s,posi_e);
    h=h+m;
    if(h>=l && my_rank==0){
      printf("%d column\n",h);
      l=l+100;
    }
  }
  bgs_m(N,P,p,Rem,h,X,Q,my_rank,process_num,posi_s,posi_e);
}

void input_size(int *pN,int *pP,FILE *fin)
```

```
{
  fscanf(fin, "%d",pN);
  fscanf(fin, "%d",pP);
}

void input_matrix(double **X,FILE *fin,int posi_s,int posi_e )
{
  int i,j,k,l;
  fscanf(fin,"%d",&k);
  for(l=0;l<k;++l){
    fscanf(fin, "%d",&i);
    fscanf(fin, "%d",&j);
    if((j>=(posi_s+1))&&(j<=(posi_e+1))){
      fscanf(fin, "%lf",&X[i-1][j-1-posi_s]);
    }
  }
}

double **dmatrix(int nr,int nl)
{
  double **a;
  int i;
  if((a=(double **)malloc(nr*sizeof(double *)))==NULL)
    {
      printf("no memory\n");
      exit(1);
    }
  for(i=0;i<nr;++i){
    a[i]=(double *)malloc(nl*sizeof(double));
  }
```

```
  return a;
}

void free_dmatrix(double **a,int nr,int nl)
{
  int i;
  for(i=0;i<nr;++i)
    free((void *)(a[i]));
  free((void *)(a));
}

double *dvector(int i)
{
  double *a;
  if((a=(double *)malloc((i)*sizeof(double)))==NULL)
    {
      printf("no free memory\n");
      exit(1);
    }
  return(a);
}

void free_dvector(double *a)
{
  free((void *)(a));
}

void bgs_m(int N,int P,int p,int m,int h,double *X,double *Q,
int my_rank,int process_num,int posi_s,int posi_e)
{
```

```
  int i,j,k,hN,Nm,ph,q,jm,jN,hj,pj,s,point,u,v,tag=0,mdv,Ni,mdvr;
  char transf[]="N",transc[]="N";
  double *nu,*W22,alpha=1.0,beta=0.0,*Y,*Z,*R12,*R22,*S12,*S22,*ZZ,*Y1,*Y2;
  MPI_Status recv_status;
  Nm=N*m;
  Y=dvector(Nm);
  hN=h*N;
  mdv=m/process_num;
  if(my_rank!=(process_num-1)){
    Y1=dvector(N*mdv);
    mdvr=mdv+(m%process_num);
    Y2=dvector(N*mdvr);
  }
  else{
    mdv=mdv+(m%process_num);
    Y1=dvector(N*mdv);
  }
  Z=dvector(N*mdv);
  R12=dvector(h*mdv);
  ph=p*h;
  s=h/p;
  s=s+1;
  if(h>(posi_s-m) && h<=posi_e){
    if((h>=posi_s-1 && h<(posi_e-m+1))||(P-m<=h)){
      for(j=0;j<m;++j){
jN=j*N;
for(i=0;i<N;++i){
  Y[i+jN]=X[((h-posi_s)*N)+jN+i];
}
      }
```

```
    }
    else{
      if(my_rank==(s-1)){
u=posi_e-h;
v=m-u;
// printf("u=%d,v=%d\n",u,v);
for(j=0;j<u;++j){
  jN=j*N;
  for(i=0;i<N;++i){
    Y[i+jN]=X[((h-posi_s)*N)+jN+i];
  }
}
MPI_Recv(&Y[u*N],v*N,MPI_DOUBLE,s,tag,MPI_COMM_WORLD,&recv_status);
      }
      else{
v=m+h-posi_s+1;
MPI_Send(&X[0],v*N,MPI_DOUBLE,s-1,tag,MPI_COMM_WORLD);
      }
    }
  }
  MPI_Bcast(Y,Nm,MPI_DOUBLE,s-1,MPI_COMM_WORLD);
  for(i=0;i<mdv;++i){
    Ni=N*i;
    for(j=0;j<N;++j){
      Y1[Ni+j]=Y[(my_rank*Nm/process_num)+Ni+j];
    }
  }
  for(k=0;k<2;++k){
    transf[0]='T';
    dgemm_(transf,transc,&h,&mdv,&N,&alpha,Q,&N,Y1,&N,&beta,R12,&h);
```

```
    transf[0]='N';
    dgemm_(transf,transc,&N,&mdv,&h,&alpha,Q,&N,R12,&h,&beta,Z,&N);
    for(i=0;i<mdv;++i){
      Ni=N*i;
      for(j=0;j<N;++j){
Y1[Ni+j]=Y1[Ni+j]-Z[Ni+j];
      }
    }
  }
  if(my_rank!=0){
    MPI_Send(Y1,N*mdv,MPI_DOUBLE,0,tag,MPI_COMM_WORLD);
  }
  else{
    for(j=0;j<N*mdv;++j){
      Y[j]=Y1[j];
    }
    for(i=1;i<process_num-1;++i){
      MPI_Recv(Y1,N*mdv,MPI_DOUBLE,i,tag,MPI_COMM_WORLD,&recv_status);
      for(j=0;j<N*mdv;++j){
Y[j+(i*N*mdv)]=Y1[j];
      }
    }
    MPI_Recv(Y2,N*mdvr,MPI_DOUBLE,i,tag,MPI_COMM_WORLD,&recv_status);
      for(j=0;j<N*mdvr;++j){
Y[j+((process_num-1)*N*mdv)]=Y2[j];
      }
       gssro(Y,N,m);
  }
  MPI_Bcast(Y,Nm,MPI_DOUBLE,0,MPI_COMM_WORLD);
  for(j=0;j<m;++j){
```

```
    jN=j*N;
    for(i=0;i<N;++i)
      Q[hN+jN+i]=Y[i+jN];
  }
}

void gssro(double *Y,int N,int m)
{
 int k,i,j,km,Nk,incx=1,l;
  double *y,*r,*z,*s,nu,nu1,alpha=1.0,beta=0.0;
  char transc[]="N";
  y=dvector(N);
  r=dvector(N);
  s=dvector(N);
  z=dvector(N);
  for(i=0;i<N;++i)
    y[i]=Y[i];
  nu=norm(N,y);
  //  R22[0]=nu;
  for(i=0;i<N;++i)
    Y[i]=y[i]/nu;
  for(k=1;k<m;++k){
    km=k*m;
    Nk=N*k;
    for(i=0;i<N;++i){
      y[i]=Y[i+Nk];
    }
    nu=norm(N,y);
    for(i=0;i<k;++i)
      s[i]=0.0;
```

```
    // while(1){
      transc[0]='T';
      dgemv_(transc,&N,&k,&alpha,Y,&N,y,&incx,&beta,r,&incx);
      transc[0]='N';
      for(i=0;i<k;++i)
s[i]=s[i]+r[i];
      dgemv_(transc,&N,&k,&alpha,Y,&N,r,&incx,&beta,z,&incx);
      for(i=0;i<N;++i)
y[i]=y[i]-z[i];
      nu1=norm(N,y);
      /*
      if(nu1>0.5*nu)
break;
      else
nu=nu1;
      */
      //    }
    for(i=0;i<N;++i)
      Y[i+Nk]=y[i]/nu1;
    //    for(i=0;i<k;++i)
    // R22[km+i]=s[i];
    // R22[km+k]=nu1;
  }
  /*
    free_dvector(y);
    free_dvector(r);
    free_dvector(s);
    free_dvector(z);
  */
}
```

```
void mnorm(double *A,int nr,int nl,double *nu)
{
  int i,j,cn;
  double s;
  for(j=0;j<nl;++j){
    s=0.0;
    cn=nr*j;
    for(i=0;i<nr;++i)
      s=A[i+cn]*A[i+cn]+s;
    nu[j]=s;
  }
}

double norm(int N,double *y)
{
  int i;
  double s;
  s=0.0;
  for(i=0;i<N;++i)
    s=y[i]*y[i]+s;
  s=sqrt(s);
  return s;
}

void gss(double *Y,int N,int m)
{
  int k,i,j,km,Nk,incx=1;
  double *y,*r,*z,*s,nu,alpha=1.0,beta=0.0,nu1;
  char transc[]="N";
```

```
  //  for(i=0;i<m*m;++i)
  //R22[i]=0.0;
  y=dvector(N);
  r=dvector(N);
  z=dvector(N);
  for(i=0;i<N;++i)
    y[i]=Y[i];
  nu=norm(N,y);
  //R22[0]=nu;
  for(i=0;i<N;++i)
    Y[i]=y[i]/nu;
  for(k=1;k<m;++k){
    Nk=N*k;
    for(i=0;i<N;++i)
      y[i]=Y[i+Nk];
    nu=norm(N,y);
    //   while(1){
    for(j=0;j<2;++j){
      transc[0]='T';
      dgemv_(transc,&N,&k,&alpha,Y,&N,y,&incx,&beta,r,&incx);
      transc[0]='N';
      dgemv_(transc,&N,&k,&alpha,Y,&N,r,&incx,&beta,z,&incx);
      for(i=0;i<N;++i)
y[i]=y[i]-z[i];
      nu1=norm(N,y);
      if(nu1>0.5*nu)
break;
      else
nu=nu1;
    }
```

```
    for(i=0;i<N;++i)
      Y[i+Nk]=y[i]/nu1;
  }
    free_dvector(y);
    free_dvector(r);
    free_dvector(z);
}

double gettimeofday_sec()
{
  struct timeval tv;
  gettimeofday(&tv,NULL);
  return tv.tv_sec + (double)tv.tv_usec*1e-6;
}

double QRSD(double *Q,int N,int P)
{
  char transf[]="T",transc[]="N";
  double *Qrsd,s,max,alpha=1.0,beta=0.0;
  int i,j,jP;
  max=0.0;
  Qrsd=dvector(P*P);
  dgemm_(transf,transc,&P,&P,&N,&alpha,Q,&N,Q,&N,&beta,Qrsd,&P);
  for(j=0;j<P;++j){
    jP=P*j;
    for(i=0;i<P;++i){
      if(i!=j)
s=Qrsd[i+jP];
      else{
s=Qrsd[i+jP]-1.0;
```

```
      }
      if(s<0.0)
s=-s;
      if(max<s){
max=s;
printf("max=%4.2e,i=%d,j=%d,\n",max,i,j);
      }
    }
  }
  return max;
}

void decide_m(int N,int P,int p,int *m,int *h,double *X,double *Q,
int my_rank,int process_num,int posi_s,int posi_e)
{
  double aa,bb,cc,dd;
  double dm,dN;
  double mt1,mt2,s,ss;
  double *A,*w,*b,*work;
  int l,i,j,pp,m1,g,bcol,iwork=g*2,info,acol;
  char trans='N';
  g=5;
  bcol=1;
  A=dvector(g*5);
  b=dvector(g);
  acol=5;
  dN=(double)N;
  *m=0;
  for(i=0;i<g;++i){
    m1=2;
```

```
    bgs_m(N,P,p,m1,h,X,Q,my_rank,process_num,posi_s,posi_e);
    *h=*h+m1;
    for(l=0;l<i;++l)
      m1=m1*2;
    dm=(double)m1;
    for(j=0;j<2;++j){
    cc=gettimeofday_sec();
    bgssro_m(N,P,p,m1,h,X,Q,my_rank,process_num,posi_s,posi_e);
    dd=gettimeofday_sec();
    if(j==0)
      aa=dd-cc;
    if(j==1)
      bb=dd-cc;
    *h=*h+m1;
    }
    cc=(bb-aa)/dm;
    printf("cc=%f\n",cc);
    b[i]=(dN/dm-1.0)*((0.5*cc*dN/dm)+2.0*aa-bb);
    A[i]=dm*dm*dm*dm;
    A[i+g]=dm*dm*dm;
    A[i+(g*2)]=dm*dm;
    A[i+(g*3)]=dm;
    A[i+(g*4)]=1.0;
  }
  iwork=g+5;
  work=dvector(iwork);
  for(i=0;i<g;++i)
    printf("b[]=%4.2e\n",b[i]);
  dgels_(&trans,&g,&acol,&bcol,A,&g,b,&g,work,&iwork,&info);
  printf("\n");
```

```
for(i=0;i<g;++i)
  printf("b[]=%4.2e\n",b[i]);
double x1=0.0,x2,fx,dfx,d=1.0;
double y1,y2,z1,z2,EPS=1.0e-10;
int max=10;
i=1;
while(fabs(d) > EPS && i<max){
  fx=(4.0*b[0]*x1*x1*x1)+(3.0*b[1]*x1*x1)+(2.0*b[2]*x1)+b[3];
  dfx=(12.0*b[0]*x1*x1)+(6.0*b[1]*x1)+2.0*b[2];
  d=-(fx/dfx);
  x2=x1+d;
  x1=x2;
  printf("x2=%4.2e\n",x2);
  ++i;
}
if(i==max){
  printf("no1\n");
}
y1=x2;
z1=(b[0]*x1*x1*x1*x1)+(b[1]*x1*x1*x1)+(b[2]*x1*x1)+(b[3]*x1)+b[4];
printf("z1=%4.2e\n",z1);
d=1.0;
x1=100.0;
i=1;
while(fabs(d) > EPS && i<max){
  fx=(4.0*b[0]*x1*x1*x1)+(3.0*b[1]*x1*x1)+(2.0*b[2]*x1)+b[3];
  dfx=(12.0*b[0]*x1*x1)+(6.0*b[1]*x1)+(2.0*b[2]);
  d=-(fx/dfx);
  x2=x1+d;
  x1=x2;
```

```
    printf("x2=%4.2e\n",x2);
    printf("fx=%4.2e\n",fx);
    ++i;
  }
  if(i==max){
    printf("no2\n");
  }
  y2=x2;
  z2=(b[0]*x1*x1*x1*x1)+(b[1]*x1*x1*x1)+(b[2]*x1*x1)+(b[3]*x1)+b[4];
  printf("z2=%4.2e\n",z2);
  if(z1>=z2)
    dm=y2;
  else
    dm=y1;
  while((double)*m<dm+1)
    *m=*m+1;
  printf("m=%f\n",dm);
}
```

付 録 C

Block Symplectic Gram-Schmidt 法

```
#include"sgsoperation.h"

void bbsgs(Matrix *A, Matrix *V, Matrix *R)
{

  int N=A->row, P=A->column, *pN, *pP, m=2, i, j, k,kN, kp,K,km, jN,n,p,inc=1;

  /* check size of A */
  if((N%2 != 0)||(P%2 != 0)){
    printf("can not copute because the size of A is odd\n");
    exit(1);
  }
  pN=&N; pP=&P; n=N/2; p=P/2;
  double minusone=-1.0, ONE=1.0;
  printf("input block-size m\n");
  scanf("%d",&k);
  if(k%2==1){
  printf("invalid value, m=2\n");
  }else{
  m=k;
  }

K=P/m;kp=P%m;
```

```
Matrix *AA, *VV, *RR, W;
AA=CreateMatrix(N,m);
VV=CreateMatrix(N,m);
RR=CreateMatrix(m,m);
  //  W=CreateMatrix(N,2);
  /* A1=V1R1 via ESR */

copyMatrix(A, 0, m, AA, 0);
// ESR2(AA, VV, RR);
bsgs(AA, VV, RR);
copyMatrix(VV, 0, m, V, 0);
copyMatrixRR(RR, R, 0);

   /* start roop */
printf("start operation\n");
for(k=1; k<K; ++k){
km=k*m;
kN=N*km;
copyMatrix(A, kN, m, AA, 0);
Matrix *H;
H=CreateMatrix(km,m);
H=GetJMatrixMatrixProductH(V, km, AA, m);
dgemm_("N","N", &N, &m, &km, &minusone, V->a, &N, H->a, &km, &ONE, AA->a, &N);
msgs(AA, VV, RR);

//     ESR2(AA, VV, RR);
reorthobbsgs(VV, V, R, H, RR, km);

copyMatrix(VV, 0, m, V, kN);
copyMatrixRR(RR, R, km*(P+1));
```

```
copyMatrixH(H, R, km*P);



}
if(kp!=0){
printf("start rest of operation\n");

km=K*m;
kN=km*N;
AA=CreateMatrix(N,kp);
VV=CreateMatrix(N,kp);
RR=CreateMatrix(kp,kp);

copyMatrix(A, kN, kp, AA, 0);
Matrix *H;
H=CreateMatrix(km,kp);
H=GetJMatrixMatrixProductH(V, km, AA, kp);
dgemm_("N","N", &N, &kp, &km, &minusone, V->a, &N, H->a, &km, &ONE, AA->a, &N);
bsgs(AA,VV,RR);
copyMatrix(VV, 0, kp, V, kN);
copyMatrixRR(RR, R, km*(P+1));
copyMatrixH(H, R, km*P);
reorthobbsgs(VV, V, R, H, RR, km);

}
}
```

付 録 D

Scheme E

```
#include"sgsoperation.h"

int determineblocksize(Matrix *X, Matrix *V, Matrix *R)
{
printf("start determine block-size \n");
int N=X->row,P=X->column,n=N/2,p=P/2;
int i,j,k=2,l=5, tempx;
double tempm[l+1],tempt[l+1],comp[l+1],t1,t2;
double start,end;
tempm[0]=1;

Matrix *A; Vector *b, *ipiv;
A=CreateMatrix(l,l);
b=CreateVector(l);
ipiv=CreateVector(l);
// printf("start taking samples \n");

for(i=1; i<=l; ++i){
tempm[i]=tempm[i-1]*2;

start=gettimeofday_sec();
bsgsblocksize(X, V, R, tempm[i],k);
end=gettimeofday_sec();
```

```
t1=end-start;
k=k+tempm[i];

start=gettimeofday_sec();
bsgsblocksize(X, V, R, tempm[i],k);
end=gettimeofday_sec();
t2=end-start;
k=k+tempm[i];

b->v[(i-1)]=((((2*N*tempm[i]*k)+(2.0*N*tempm[i]*(tempm[i]-1.0)))
*(8.0*N*N*N+1))+N*(N*N+3.0)*(1+(tempm[i-1])))/tempm[i];
b->v[(i-1)]=(t2-t1)/b->v[(i-1)];
tempx=tempm[i];
for(j=0; j<l; ++j){
A->a[(i-1)+(j*l)]=tempx;
tempx=tempx*tempx;
}
}

// printf("finish taking samples\n");
int info=N+1, ONE=1, m;
// dgels_("N", &l, &l, )  /* dgels solves least square problem. */
dgesv_(&l, &ONE, A->a, &l, ipiv->v, b->v, &l, &info);
if(info != 0){
printf("error to determine block size. info of dgesv = %d\n ", info);
exit(1);
}
printf("start netwton method\n");
m=newtonmethod(b);
printf("finish netwton method\n");
```

```
if(m>N || m<0){
printf("invalied block size. set m=2\n");
m=70;
}
V->column=k;
printf("finish determine block-size.\n m=%d\n",m);

return(m);
}

void bsgsblocksize(Matrix *A, Matrix *V, Matrix *R, int tempm, int tempcolumn)
{

  int N=A->row, P=A->column, *pN, *pP, m=tempm, i, j, k,kN, kp,K,km, jN,n,p,inc=1;

  pN=&N; pP=&P; n=N/2; p=P/2;
  double minusone=-1.0, ONE=1.0;

Matrix *AA, *VV, *RR;
AA=CreateMatrix(N,m);
VV=CreateMatrix(N,m);
RR=CreateMatrix(m,m);
km=tempcolumn;
kN=N*km;
copyMatrix(A, kN, m, AA, 0);
Matrix *H;
H=CreateMatrix(km,m);
H=GetJMatrixMatrixProductH(V, km, AA, m);
dgemm_("N","N", &N, &m, &km, &minusone, V->a, &N, H->a, &km, &ONE, AA->a, &N);
```

```
bsgs(AA, VV, RR);
reorthobbsgs(VV, V, R, H, RR, km);
printf("m=%d,km=%d\n",m,km);

copyMatrix(VV, 0, m, V, kN);
copyMatrixRR(RR, R, km*(P+1));
copyMatrixH(H, R, km*P);

}
```