Vol.2020-HPC-173 No.2
2020/3/16

ロバストなSA-AMG法に向けた ニアカーネルベクトル抽出手法に関する研究

野村 直也[1]　中島 研吾[1]　藤井 昭宏[2]

概要：大規模連立一次方程式 $A\boldsymbol{x} = b$ に対する高速かつスケーラブルな解法として SA-AMG 法がある．SA-AMG 法は与えられた問題行列から，階層的に粗い問題と階層移動に必要な補間演算子を再帰的に生成し，それらを用いて問題を解く．これにより，様々な波長の誤差成分を効率よく減衰させ，高速で安定な収束やスケーラビリティを実現している．ここで，補間演算子生成に問題に応じたニアカーネルベクトル（行列 A との行列ベクトル積の結果が 0 ベクトルに近くなるようなベクトル，0 固有値に近い固有ベクトルに対応）を用いることで，収束性をさらに高められることが知られている．ニアカーネルベクトルの設定に関しては，αSA 法を含めいくつかの関連研究が存在する．著者等は，粗いレベルでニアカーネルベクトルを複数本抽出し，それらを用い追加的に設定本数を増やす手法の提案を行ってきた．この手法を用いることで，スケーラブルな収束性を実現し，実行時間の改善がみられることもわかった．本発表では，ニアカーネルベクトルのより効率的な抽出手法の検討，および様々な問題に対して手法を用いた際の結果を報告する．本発表では抽出のさらなる効率化のため，固有値計算法を用いた粗レベル補間近似ニアカーネルベクトル抽出法を提案した．この手法では，行列サイズが小さい粗いレベルの係数行列に対して固有値解析を実施する．これにより，0 固有値に近い固有ベクトル，つまりニアカーネルベクトルを低コストで抽出できると考えられる．本発表では，本手法を Texas A&M 大学の SuiteSparse Matrix Collection より取得した問題に対して適用し，様々な問題に対する有用性の検証を行った．この手法により，従来手法と比べ抽出時間を抑えつつ，従来手法と同等の高い収束性を実現できることがわかった．

キーワード：Multigrid 法，SA-AMG 法，ニアカーネルベクトル抽出手法

1. はじめに

コンピュータによるシミュレーションに基づく計算科学は，有限要素法，差分法のような手法が広く用いられる．これらの手法では，最終的に連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解くことに帰着される．近年では，より複雑な問題を正確にシミュレートすることが求められており，それにより問題の大規模化が進んでいる．そのため，大規模連立一次方程式を高速かつ安定に解く手法の開発は急務となっている．

そこで，大規模連立一次方程式を高速に解く手法として Multigrid 法が使用されている．Multigrid 法は階層的に粗い問題を生成することで，様々な波長の誤差成分を効率よく減衰させ，高速で安定な収束やスケーラビリティを実現している．Multigrid 法は大きく分けて，幾何的 Multigrid（GMG）法と代数的 Multigrid（AMG）法 [1], [2] の 2 つに分類される．本研究ではその中で，AMG 法を対象とする．AMG 法の中でも階層行列の生成方法により，さらに様々な派生解法が存在する [3]．本研究ではその中で特に，Smoothed Aggregation に基づく AMG（SA-AMG）法と呼ばれる手法を対象とした [4], [5]．この手法は多くの問題に対して有用であることが知られており，広く用いられている解法となっている．

SA-AMG 法は，収束しにくい誤差成分を粗いレベルで効率よく減衰させることで，高い収束性を実現している．ここで，収束しにくい成分とは，一般にニアカーネルベクトルと呼ばれ，問題行列 A との行列ベクトル積が 0 に近くなるような非ゼロベクトルをいう．Gauss-Seidel 法のような通常の定常反復解法で解いた際，この成分が収束の停滞を引き起こす要因となることが知られている．SA-AMG 法では，階層行列の生成においてニアカーネルベクトルを用いて粗いレベルの行列を生成することで，粗いレベルの行列に誤差成分が射影され，誤差成分の効率よい減衰を実現している．その結果，残差を効率よく収束させることが

[1] 東京大学
The University of Tokyo
[2] 工学院大学
Kogakuin University

できる．

本研究では科学技術計算において広く用いられている，3 次元弾性体の問題を用いて実験を行っている．この問題では平行移動成分と回転成分がニアカーネルベクトルとして知られている．著者らによる先行研究により，SA-AMG 法においてこれらのニアカーネルベクトルを設定することにより，反復回数と実行時間双方で改善がみられることがわかっている．しかし，これは問題設定に依存したものであり，すべてのユーザーが対象とする問題の物理的性質に基づいた，適切なニアカーネルベクトルの設定を行うことができるとは限らない．SA-AMG 法の高い収束性をより多くの問題に対して生かすために，係数行列 A から適切な数のニアカーネルベクトルを，代数的に効率よく抽出する手法の開発が急務である．

そこで著者等は，問題行列に応じた適切なニアカーネルベクトルの設定方法と，その効果についての研究を行ってきた．著者等の現在までの研究では，問題行列からニアカーネルベクトルを複数本抽出する手法の提案，および有用性の検証を行った．そして，その手法で抽出したニアカーネルベクトルを SA-AMG 法に適切な本数設定することで，平行移動成分と回転成分を設定した場合よりも，反復回数と実行時間双方で改善がみられることがわかった [6], [7]．

本研究では抽出のさらなる効率化のため，粗いレベルで固有値解法を用い補間を行うことでニアカーネルベクトルを抽出する手法を用い，抽出されたニアカーネルベクトルを用いることによる反復回数や実行時間，および抽出時間への影響の分析を行った．その後，本手法に対し Texas A&M 大学の SuiteSparse Matrix Collection より取得した問題に対して適用し，様々な問題に対する有用性の検証を行った．その後本稿では付録として，抽出したベクトルが適切にニアカーネルベクトルを表現できているかの，検証実験を行った結果を示す．

2. Multigrid 法

2.1 Multigrid 法の概要

有限要素法のような格子を用いて離散化した問題に対し定常反復解法を適用すると，メッシュサイズと同じサイズの誤差が早く減衰し（空間的高周波成分），長い波長の誤差は減衰しにくい（空間的低周波成分）ことが知られている [8]．そこで Multigrid 法ではこの性質に着目し，粗い格子を階層的に生成しそれらを用いることで，低周波成分を効率よく減衰させ，高速で安定な収束やスケーラビリティを実現している．Multigrid 法の大きな特徴として、収束性が問題サイズによらないスケーラブル性があり，大規模問題に対して非常に有効な解法となっている．Multigrid 法は粗い格子の生成方法により，さらに GMG 法と AMG 法が存在する．GMG 法は粗い格子の生成に空間離散化の情報が必要となる．一方，AMG 法は係数行列 A のみでよく，不規則メッシュ状の問題やメッシュそのものが存在しない問題まで適用できるため，一般に AMG 法のほうが対象とする問題の適用可能範囲が広く，よく用いられている．

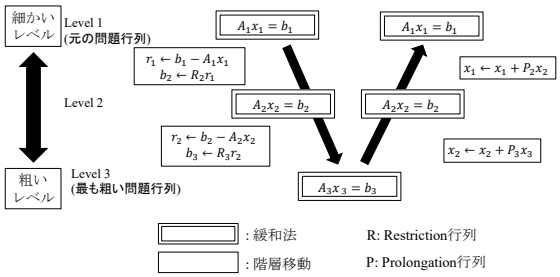

図 1 求解部の概要（V-cycle）

2.2 Multigrid 法のアルゴリズム

本節では，Multigrid 法のアルゴリズムを示す．Multigrid 法は階層行列を生成する構築部（Setup part）と，それらを用いて問題を解く求解部（Solution part）に分かれている．以下より，それぞれについての説明を行う．

2.2.1 構築部

構築部では細かいレベルの問題行列を基に，未知数間のグラフ構造を作り，次のレベルに残す未知数を選択する．そして，粗いレベルの問題行列や，階層移動の際に必要となる補間演算子 Prolongation（P）行列と Restriction（R）行列を生成する．これを再帰的に行うことで，階層的に行列を生成する．1 レベル目は与えられた問題行列が置かれ，階層が下がるにつれて問題行列より小規模な行列を生成する．階層数は問題サイズによって可変となる．

2.2.2 求解部

Algorithm 1 に Multigrid 法の求解部の計算方法のひとつである V-cycle についての概要を示す．$\boldsymbol{x}_l$ のように添え字に l がついているものは，レベル l におけるベクトルや行列を意味している．階層移動（Coarse grid correction の箇所）では，構築部で作成された補間演算子の Prolongation 行列と Restriction 行列を使う．階層を下りる際には，現階層の残差を計算し，横長の Restriction 行列と行列ベクトル積を行うことで短いベクトルを生成し，ひとつ下の階層で利用する．階層を上る際は，現階層の解と縦長の Prolongation 行列との行列ベクトル積を行うことで長いベクトルを生成し，ひとつ上の階層の補正解として利用する．Multigrid 法を解法として用いる際には，Algorithm 1 の計算手順をマルチグリッド法の 1 反復とし，これを期待する精度の近似解が得られるまで繰り返す．図 1 に，以上の Multigrid 法における V-cycle の流れをまとめたものを図示する．この図のように，複数の階層を行き来する様子が V 字を連想させるため，V-cycle と呼ばれている．

2.3 代数的多重格子（Algebraic Multigrid，AMG）法

AMG 法は上記でも述べた通り，係数行列 A の情報のみ

Algorithm 1 Multigrid 法（V-cycle）

Pre-smoothing: $\tilde{\boldsymbol{x}}_l \leftarrow S_l(\boldsymbol{x}_l, \boldsymbol{b}_l)$
Coarse grid correction:
$\boldsymbol{r}_l \leftarrow \boldsymbol{b}_l - A_l\ \tilde{\boldsymbol{x}}_l$
$\boldsymbol{b}_{l+1} \leftarrow R_l\ \boldsymbol{r}_l$
if $l+1 = L$ **then**
　$A_L\ \boldsymbol{x}_L = \boldsymbol{b}_L$ を直接法（例；LU 分解）で解く．
else
　$l+1$ において $\boldsymbol{x}_{l+1} = 0$ とし Pre-smoothing から計算．
end if
$\boldsymbol{x}_l \leftarrow \boldsymbol{x}_l + P_l\ \boldsymbol{x}_{l+1}$
Post-smoothing: $\boldsymbol{x}_l \leftarrow S_l(\boldsymbol{x}_l, \boldsymbol{b}_l)$

$S(\boldsymbol{x}, \boldsymbol{b})$: $A\boldsymbol{x} = \boldsymbol{b}$ に対するスムーザの適用
$l = 1, 2, ..., L$: レベル

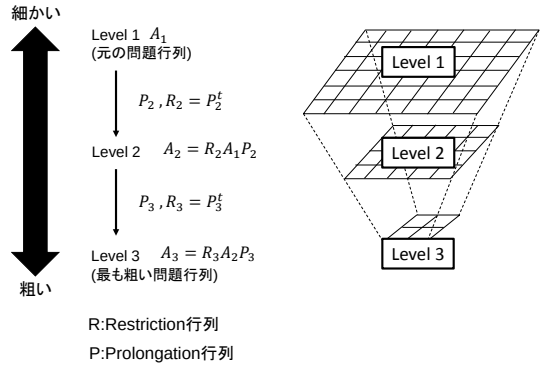

図 2 構築部の概要

に基づき粗い格子を生成する Multigrid 法である．AMG 法における構築部の概要を図 2 に示す．上記で述べたように，構築部では細かいレベルの問題行列を基に，未知数間のグラフ構造を作り，次のレベルに残す未知数を選択する．そして，粗いレベルの問題行列や，階層移動の際に必要となる P 行列と R 行列を生成する．これを再帰的に行うことで，階層的に行列を生成する．図 2 のように，1 レベル目は与えられた問題行列が置かれ，階層が下がるにつれて問題行列より小規模な行列を生成する．階層数は問題サイズによって可変となる．また，粗いレベルの問題行列は，R 行列と P 行列，さらに現階層の問題行列とで行列行列積 RAP を行うことで作成する．

AMG 法では粗いレベルの行列を $\bar{A}$ とすると，$\bar{A} = RAP, R = P^T$ とするのが普通である．これは，新しい近似解 $\tilde{u} = u + Pv$ の残差が P の像空間と直交するようにするためである．このとき A が正定値対称であれば，新しい近似解に含まれる誤差 $\tilde{e} = e + Pv$ を最小にする，つまり，

$$||\boldsymbol{e} + P\boldsymbol{v}||_A = \min_{w} ||\boldsymbol{e} + P\boldsymbol{w}||_A \qquad (1)$$

となることが変分原理により確かめることができる [8]（ただし，$||w||_A = (Aw, w)^{1/2}$）．これにより，収束しにくい低周波成分を効率よく減衰でき，高収束性を実現している．

補間演算子から行列の階層構造が生成されるため，Multigrid 法では補間演算子の生成方法は重要となる．この補間演算子の生成手法により様々な AMG 法が存在する [3]．一般に AMG 法のほうが対象とする問題の適用可能範囲が広く，そのため本研究では，AMG 法を対象とし研究を行っている．AMG 法の中で，本研究では SA-AMG 法と呼ばれる手法を対象としている．SA-AMG 法はさまざまな分野で利用されており，AMG 法の代表的な手法のひとつとなっている．

3. SA-AMG 法

2 章でも述べたように，SA-AMG 法は，Multigrid 法の中の解法のひとつである．SA-AMG 法では，問題行列に基づく節点と辺で構成されたグラフ構造を用いて粗い問題を作成していく．ここで，問題行列の各行が節点に対応し，非ゼロ要素が辺に対応している．

SA-AMG 法では，粗い問題を作成する際に，節点全体をアグリゲートと呼ばれる強連結同士の節点集合に分解する．アグリゲートは図 3 のように，次の粗いレベルで 1 つの節点に対応し，グラフ構造である節点を中心に近くの節点をまとめた節点集合と定義される．そのため，アグリゲート数と次の粗いレベルの節点数は同じとなる．実際にアグリゲートを生成する際は，以下の式を用いて生成を行う．

$$N_i(\epsilon) = \{j : |A_{ij}| \geq \epsilon\sqrt{|A_{ii}||A_{jj}|}\} \cup \{i\} \qquad (2)$$

ここで，N_i は i 番目のアグリゲート，A_{ij} は行列 A の i 行 j 番目の要素を示す．この式に示す通り，i 番目のアグリゲートは i 番目の要素に対応する節点かつ，絶対値が対角成分に対し比較的大きい非対角成分の要素に対応する節点で構成される．アグリゲート生成の全体の流れを Algorithm 2 に示す．まず Algorithm 2 の Step:1 では，式 (2) を用いて，アグリゲートの生成を行う処理となっている．ここで，アグリゲート生成では，補間の関係上すべての要素がどこかしらのアグリゲートに属していなければいけないという制約がある．そのため，任意の節点がどこか 1 つのアグリゲートに属するように，アグリゲートを生成する．Step:2 ではそのために，すでに生成されたアグリゲートと残った要素とで再度式 (2) を適用し，成立した場合アグリゲートに追加する．Step:3 では，Step:2 を経てもなお残る節点に対し，その節点をアグリゲートとするように処理を行う．このように，すべての節点がいずれかのアグリゲートに属するように，アグリゲートの生成を行う．その後，作成されたアグリゲート内の未知数に重み付けをすることで補間演算子である Prolongation 行列と Restriction 行列を計算し，行列の階層構造を作成する．

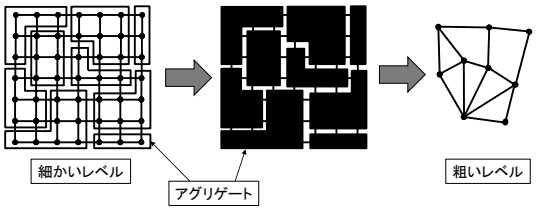

図 3　アグリゲート生成の概要

Algorithm 2 SA-AMG 法におけるアグリゲート生成の手順

Step 1:
全節点の集合 $V = 1, 2, ...$ から，$N_i(\epsilon) \in V$ を計算．
その後，$j \leftarrow j+1, C_j \leftarrow N_i(\epsilon), V \leftarrow V \setminus C_j$ とし，新たな C_j が作成できなくなるまで繰り返す．

Step 2:
C_k のコピー $\tilde{C}_k$ を作成．（$\tilde{C}_k = C_k, k = 1, ..., j$）
もし $N_i(\epsilon) \cap \tilde{C}_k \neq \phi$ となる $i \in V$ が存在するなら，$C_k \leftarrow C_k \cup \{i\}$ とし，条件を満たす i が存在しなくなるまで繰り返す．

Step 3:
もし $i \in V$ が存在するなら，$j \leftarrow j+1, C_j = V \cap N_i(\epsilon)$ とし，$V = \phi$ となるまで繰り返す．

4. ニアカーネルベクトル抽出手法

4.1 ニアカーネルベクトルの概要

ニアカーネルベクトルとは，$A\boldsymbol{e} \approx 0$ となる非ゼロベクトル $\boldsymbol{e}$ をいう（代数的に滑らかな成分とも呼ばれる）．2.1 節にて述べた通り，Gauss-Seidel 法のような定常反復解法を数回適用すると，高周波成分が早く減衰し，低周波成分だけが残るという現象がある．つまり，定常反復解法の反復行列を S と定義すると，$S\boldsymbol{e'} \approx \boldsymbol{e'}$ となる低周波成分 $\boldsymbol{e'}$ が存在する．これにより，定常反復解法において一定回数の反復の後は，収束性が悪化することが多く見受けられる．この成分 $\boldsymbol{e'}$ はニアカーネルベクトル $\boldsymbol{e}$ と同値である．また，ニアカーネルベクトルは 0 固有値に近い固有ベクトルに対応することが知られている [3].

AMG 法では，補間演算子である Prolongation 行列（Restriction 行列）を適切に選ぶことで，ニアカーネル成分を効率よく減衰できる．具体的には，2.3 節にて述べたように，適当な近似解 v を選んで新しい近似解 $\tilde{\boldsymbol{u}} = \boldsymbol{u} + P\boldsymbol{v}$ に含まれる誤差 $\tilde{\boldsymbol{e}} = \boldsymbol{e} + P\boldsymbol{v}$ を 0 にすることを考えた際，適当な条件下において，式 1 のような最小化が行われるのであった [8]．このように，適切なニアカーネルベクトルを用いることで，減衰しにくい成分（つまりニアカーネルベクトル成分）を効率よく減衰させることができ，高い収束が実現可能となる．ここで，適切な Prolongation 行列を設定するためには $\tilde{\boldsymbol{e}} = \boldsymbol{e} + P\boldsymbol{v}$ から，$\boldsymbol{e} \in \mathrm{Im}P$ となることが必要である [8]．著者らの研究から，SA-AMG 法ではニアカーネルベクトルを用いて，Prolongation 行列の作成を行うことが有効となることがわかっている．これにより，Prolongation 行列の各行ベクトルの線形結合によりニアカーネルベクトルを表現することが可能となり，$\boldsymbol{e} \in \mathrm{Im}P$ となることが期待できる．以上より Multigrid 法の特徴である高い収束性の実現のため，実際に SA-AMG 法を適用する際には，ニアカーネルベクトルの生成および設定方法の確立が必要不可欠となる．

問題の性質からニアカーネルベクトルが特定できる場合もある [9]．例えば，本研究で用いている 3 次元弾性体の問題では，平行移動成分と回転成分がニアカーネルベクトルとして知られている．弾性体問題は，物体に力が加えられたときの，物体の変形をシミュレートする問題である．弾性体の問題を対象としている場合，これらを SA-AMG 法に用いることで，収束性が高まる．

4.2 SA-AMG 法への補間演算子生成方法

補間演算子作成の流れを，図 4 に図示する．図 4 は 2 本のニアカーネルベクトルを用いて，問題 A から 2 つのアグリゲートが作成されたときの，Prolongation 行列の作成方法を図示している．図 4 のように，まずアグリゲートの節点番号に対応したニアカーネルベクトルの列要素を，一次行列 S に入力する（a）．その後，S に対し QR 分解を行い，算出された行列 Q を仮の補間演算子 $\tilde{P}_l$ に入力する（b）．同時に算出された行列 R は，アグリゲート情報を基に，次レベルのニアカーネルベクトル V_{l+1} の構築に用いられる（c）．最後に，仮の補間演算子 $\tilde{P}_l$ に緩和法を適用する（d）．以上の操作を行うことで，補間演算子 P_l や次レベルのニアカーネルベクトル V_{l+1} が生成される．図 4 の例では Level$l+1$ において，ニアカーネルベクトルを要素番号と対応させるために，1 要素が 2×2 のブロック行列として扱う必要がある．このように，次レベルにおけるニアカーネルベクトル V_{l+1} は，1 節点が $N_v \times N_v$ のブロック行列として扱うこととなる．図 4 は 2 本のニアカーネルベクトルを用いた例だが，本研究の SA-AMG 法では，より多くのニアカーネルベクトルを設定することができる．その場合，Step.1 における補間演算子の候補行列 $\hat{P}$ の行列サイズが大きくなり，結果として計算量が増加する．そのため，ニアカーネルベクトルを複数本設定することで反復回数が減少するが，1 反復あたりの計算時間が増加するといった，トレードオフが発生すると考えられる．

4.3 著者らによるニアカーネルベクトル抽出先行研究

4.3.1 ニアカーネルベクトル抽出先行研究手法 1[6]

ニアカーネルベクトル抽出手法の先行研究として αSA 法 [10] があるが，この手法では全階層の行列に対するニアカーネルベクトルを 1 本のベクトルで作成する．しかし著

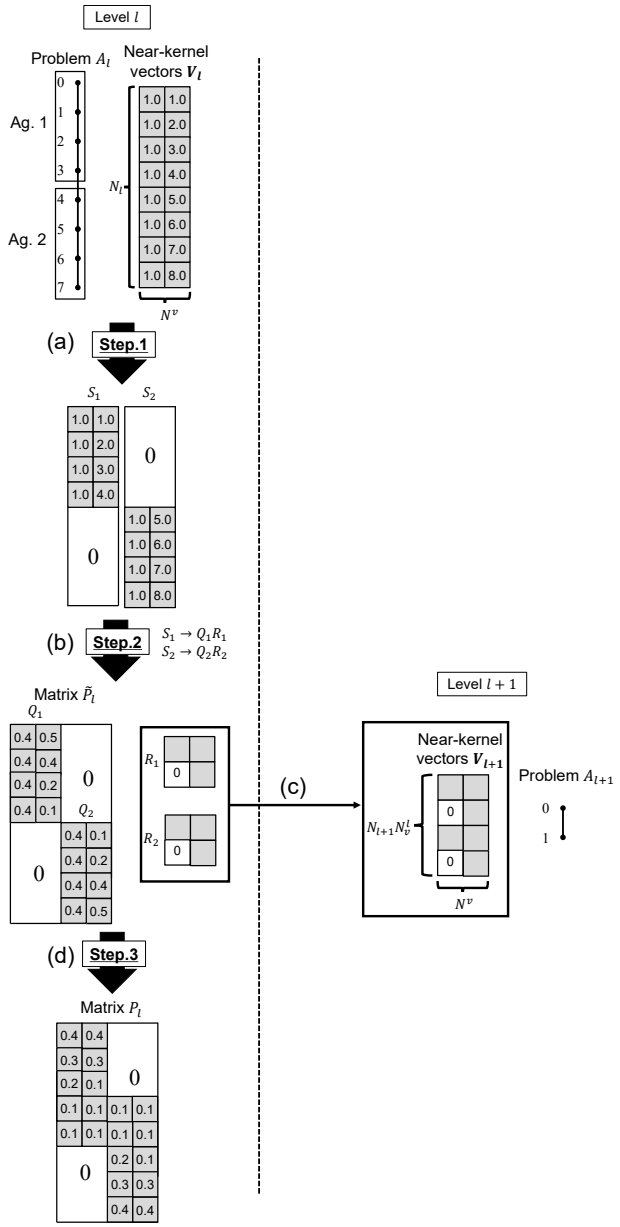

図 4 補間演算子（Prolongation 行列）の生成方法

者らは，全階層を 1 本のベクトルのみに基づいて表現することは難しく，さらにニアカーネルベクトルを用いることによる SA-AMG 法の収束性への影響について，最も細かいレベルの行列が，最も影響を与えると考えた．そこで，抽出時間の削減も含め，ニアカーネルベクトル抽出の効率化を図るため，著者らによる先行研究として，αSA 法を基に，V-cycle を用いレベル 1 のみにおいてニアカーネルベクトルの抽出を行う手法を提案し，収束性や実行時間に関する計測および評価を行った [6]．Algorithm 3 に [6] における抽出手法の概要を示す．この手法では Algorithm 3 のように，まず初期ベクトル $\boldsymbol{x}$ を乱数にて初期化を行い（4 行目），$A\boldsymbol{x}$=0 を対象に V-cycle を μ 回繰り返す（5 行目）．この μ はユーザー側が与えるパラメータである．その後，ニアカーネルベクトル候補群である行列 B に，V-cycle により出てきたベクトル $\boldsymbol{x}$ を入力する（6 行目）．その後 αSA 法と同様に，抽出されたニアカーネルベクトル群の行列 B を使用して，図 4 で示した補間演算子 P の生成，および粗いレベルの行列生成を再度全レベルで行い，階層行列を再度作り変える（7 行目）．これを抽出したい本数分繰り返す（3 行目から 8 行目）．最後に，ニアカーネルベクトル候補群である行列 B を，ニアカーネルベクトルとして出力する．このようにして，レベル 1 の与えられた問題行列 A に対して，ニアカーネルベクトルの複数本抽出を実現している．本手法では最初に行列 B を与えることとなっているが，これはニアカーネルベクトルとして考えられるベクトルを入力する（例えば，3 次元弾性体問題では平行移動や回転成分を行列 B に与えている）．

Algorithm 3 の 5 行目において $A\boldsymbol{x} = 0$ に対し V-cycle を適用しているが，$A\boldsymbol{x} = 0$ を V-cycle で近似的に解くことにより，V-cycle で減衰されない成分が残ることが期待できる．つまり，収束しにくいニアカーネルベクトル成分 $\boldsymbol{e}$ に対し，前処理行列を M とすると

$$M\boldsymbol{e} \approx \boldsymbol{e} \tag{3}$$

となる成分が残ると考えられる．これを SA-AMG 法にニアカーネルベクトルとして追加設定し，それらを基に階層行列を再生成し，さらに再度 V-cycle で近似的に解く．これにより，SA-AMG 法における設定したニアカーネルベクトルの成分が効率よく減衰される性質により，設定されたニアカーネルベクトルの成分が除かれた，新たなニアカーネルベクトルが抽出されると期待できる．つまり，ニアカーネルベクトル全体のなす空間を E，n 本の抽出されたニアカーネルベクトル成分でなす空間を $\tilde{E}_n$ とし，n 本目に抽出されたニアカーネルベクトルを e_n とすると，$n+1$ 本目に抽出されるニアカーネルベクトル $\boldsymbol{e}_{n+1}$ は

$$\boldsymbol{e}_{n+1} \in E \backslash \tilde{E}_n \tag{4}$$

となるいずれかのニアカーネルベクトル成分が抽出できることが期待できる．これを実現するために，5 行目にて抽出されたニアカーネルベクトルを用いて，階層行列の再構成を行っている．本研究ではニアカーネルベクトルが抽出されるたびに，余分な成分を除去するため QR 分解を施し，抽出された各ベクトルが一次独立となるような処理を行っている．これにより，本手法を用いることで，適切なニアカーネルベクトルを効率よく複数本抽出できると予想される．この手法では，最初にニアカーネルベクトルを設定する必要があり，このベクトルをもとに独立なニアカーネルベクトルを抽出していくこととなる．

4.3.2 ニアカーネルベクトル抽出先行研究手法 2[7]

[10] や [6] の手法では，最終的にレベル 1 としてのニアカーネルベクトルのみを出力している．しかし，粗いレベルのニアカーネルベクトルは，細かいレベルのニアカーネルベクトルだけでは不十分であり，粗いレベルにおいても

Algorithm 3 先行研究手法 1 におけるニアカーネルベクトル抽出手法

Given : B
for $n = 1$ to *extract_number* **do**
  $\tilde{\boldsymbol{e}} \leftarrow Random()$
  $e \leftarrow V\_cycle^{\mu}(A\tilde{\boldsymbol{e}} = 0)$
  $B \leftarrow [B, \boldsymbol{e}]$
  $Multilevel\_creation(B)$
end for
Output B matrix as near-kernel vectors

$Random()$：乱数生成
extract_number：抽出したい本数
A：与えられた問題行列 A
$V\_cycle^{\mu}(A\boldsymbol{e} = 0)$：$A\boldsymbol{e} = 0$ を対象に V-cycle を μ 回適用
B：ニアカーネルベクトル候補群の行列
$[B, \boldsymbol{e}]$：行列 B の最終列へのベクトル $\boldsymbol{e}$ の追加
$Multilevel\_creation(B)$：行列 B を基に，補間演算子 $P_1, P_2, ...$，および階層行列 $A_1, A_2, ...$ を再生成

抽出や追加に設定を行えるようにするべきと考えた．そこで本研究では [6] の手法を基に，粗いレベルにおいてもニアカーネルベクトルを抽出および SA-AMG 法にて追加的に設定する手法の提案，および収束性や実行時間の評価を行った [7]．Algorithm 4 に具体的な流れを示す．Algorithm 4 の赤字箇所は，Algorithm 3 から追加された箇所である．Algorithm 4 からわかるように，本手法は Algorithm 3 を基に，粗いレベルにおいてもニアカーネルベクトルを複数本抽出できるように改良を加えた手法となっている．本手法ではニアカーネルベクトル候補群である行列 B を各階層で用意しており，最終的に各階層ごとのニアカーネルベクトルがそれぞれ出力される．これらのニアカーネルベクトルを SA-AMG 法に用いる際には，まずは先行研究による手法と同様に，最も細かいレベルにおいて抽出されたニアカーネルベクトルを設定し，次の粗いレベルの行列とニアカーネルベクトルを生成する．そして，粗いレベルでも同様に，細かいレベルのニアカーネルベクトルを基に，さらに粗いレベルを生成する．本手法ではこの際，細かいレベルをもとに生成されたニアカーネルベクトルに追加して，抽出されたニアカーネルベクトルを用いる．この操作を最大レベル数-1 まで行う．図 5 に上記で述べた操作を図示する．以上のように本手法では，[6] で用いた手法に加え粗いレベルのニアカーネルベクトルも考慮しているため，粗いレベルの行列が全体の収束性に大きな影響を与えている場合，[6] よりもさらに収束性が改善することが見込める．

外部から入力するパラメータとして，*extract_number* と μ が存在する．*extract_number* は抽出したい本数であり，著者らによる研究 [6] から，適切な本数を設定することが

Algorithm 4 先行研究手法 2 のニアカーネルベクトル抽出提案手法

Given : B_1
for $level = 1$ to $max\_level - 1$ **do**
  for $n = 1$ to *extract_number* **do**
    $\tilde{\boldsymbol{e}}_{level} \leftarrow Random()$
    $\boldsymbol{e}_{level} \leftarrow V\_cycle^{\mu}(A_{level}\tilde{\boldsymbol{e}}_{level} = 0)$
    $B_{level} \leftarrow [B_{level}, \boldsymbol{e}_{level}]$
    $Multilevel\_creation(B_{level})$
  end for
end for
Output $B_1, B_2, ...$ matrices as near-kernel vectors

$Random()$：乱数生成
max_level：階層の最大レベル数
A_{level}：レベル $level$ における問題行列 A
B_{level}：レベル $level$ におけるニアカーネルベクトル候補群の行列

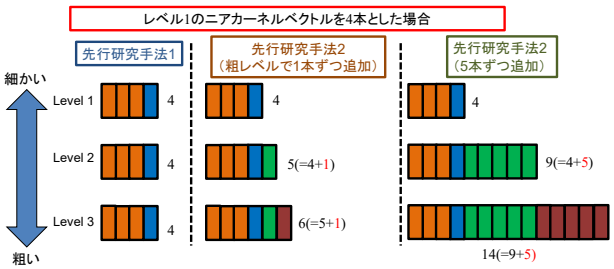

図 5　粗いレベルにおけるニアカーネルベクトル設定方法

必要であることが考えられる．数値実験において，抽出本数による反復回数や抽出時間への影響の分析を行う．また μ は，V-cycle を繰り返す回数であり，この値により収束性が変化すると考えられるが，本事項については今後の課題とし，本研究では μ を 20 に設定し，計測を行っている．

4.4　本研究で用いたニアカーネルベクトル抽出手法

4.3 節にて述べた 2 つの手法では，V-cycle を用いてニアカーネルベクトルを 1 本づつ抽出する．そのため，抽出本数により抽出時間が増加し，ニアカーネルベクトル抽出コストが，実際に解法を適用し連立一次方程式を解くコストと比べ，大きくなってしまう問題がある．

そこで，Bootstrap AMG 法 [11] と先行研究手法 2 を基に，任意の粗いレベルにおいて固有値解析を実施し補間を行い，ニアカーネルベクトルを複数階層において抽出，および設定する手法を提案し評価を行った [12]．概要を Algorithm 5 および Algorithm 6 に示す．まず，Algorithm 5 について説明する．Algorithm 5 は，レベル 1 のみを対象にニアカーネルベクトルを抽出する手法である．この手法ではまず，任意の最大レベル数以下の数 $\hat{L}$ を入力する（1

Algorithm 5 ニアカーネルベクトル抽出手法：提案手法（レベル 1 のみ）[12]

Given : $\hat{L}$
for $l = \hat{L}$ to 1 **do**
  if $l == \hat{L}$ **then**
    $W_L = \{w_l^\kappa | A_l w_l^\kappa = \lambda_l^\kappa w_l^\kappa, \kappa = 1, ..., k_e\}$
  else
    $w_l^\kappa = P_{l+1}^l w_{l+1}^\kappa, \lambda_l^\kappa = \lambda_{l+1}^\kappa, \kappa = 1, ..., k_e$
    for $\kappa = 1$ to k_e **do**
      Relax on $A_l w_l^\kappa = 0$
    end for
  end if
end for
Output W_1 matrix as near-kernel vectors

$\hat{L}$: 固有値解析対象となる最大レベル以下の任意のレベル
P_{l+1}^l : レベル $l+1$ から l へ補間を行う Prolongation 行列

Algorithm 6 提案手法における粗いレベルのニアカーネルベクトル設定方法 [12]

Given : $\hat{L}$
Given : $W_1, ..., W_{\hat{L}}$
for $l = 1$ to $\hat{L}$ **do**
  $Multilevel\_creation(W_l)$
  $\hat{W}_{l+1} = R_l^{l+1} W_l$
end for
Output $\hat{W}_1, ..., \hat{W}_{\hat{L}}$ matrices as near-kernel vectors

R_l^{l+1} : レベル l から $l+1$ へ縮約を行う Restriction 行列

行目）．そして，そのレベルにおいて，固有値解析を実施する（4 行目）．その後，補間演算子 P 行列とスムーザを用いて，細かいレベルに移動する．粗いレベルにおいての抽出を行いたい場合は，Algorithm 5 に加え，Algorithm 6 の処理を行う．Algorithm 6 は，Algorithm 4（先行研究手法 2）に基づいており，粗いレベルにおけるニアカーネルベクトルの抽出を行う際には，まず Algorithm 5 において抽出されたニアカーネルベクトルを基に階層行列の再作成を行い，階層行列の更新を行う．次に粗いレベルのニアカーネルベクトルを算出するのだが，本手法では提案手法 1 とは異なり，補間演算子 R 行列のみを用いて行列行列積を行い，最終的に算出された各レベルの行列群 $\hat{W}_1, ..., \hat{W}_{\hat{L}}$ をニアカーネルベクトルとして出力する．この操作を Algorithm 4 と同様に，レベル $\hat{L}$ まで行う．上記のように粗いレベルにおいて固有値解析を実施することで，低コストで複数本のニアカーネルベクトルを抽出できると期待できる．SA-AMG 法に用いる際には，Algorithm 4 で説明した操作と同様のことを行う．また，Algorithm 6 を行うことで，Algorithm 4 のような粗いレベルでニアカーネルベクトルを設定することによる改善効果が期待でき，さらに高い収束性能を発揮できると考えられる．このことについては，次章の数値実験において示す．

5. 数値実験と結果

5.1 実験環境

本研究では，東京大学情報基盤センターと筑波大学計算科学研究センターが共同運営する，最先端共同 HPC 基盤施設（JCAHPC : Joint Center for Advanced High Performance Computing）による，Oakforest-PACS スーパーコンピュータシステム [13] を使用し数値実験を行った．Oakforest-PACS は，1 ノードに 1 個の Intel(R) Xeon Phi(TM) プロセッサ（68cores, 1.4GHz）と，MCDRAM（16GB, 400GB/s）と DDR4（16GB×6，1 メモリ当り 19.2GB/s）メモリを搭載している（本研究では，MCDRAM のみ用いる設定を行い，数値実験を行った）．

求解部では CG 法 [14] を使用し，前処理として SA-AMG 法を適用している．求解部で用いる V-cycle の各レベルの緩和法として，対称 Gauss-Seidel 法を 2 回適用する．ただし，領域境界では，依存関係を無視している．また節点数が 100 以下になったとき，最も粗いレベルとし，LU 分解を行い，解を求めている．また，反復の終了条件は相対残差が 1.0×10^{-7} とし，反復回数の上限を 500 回とした．

5.2 数値実験内容

本研究では，以下のような 2 つの数値実験を行った（それぞれ実験 1，実験 2 とする）．

(1) ニアカーネルベクトル抽出本数による反復回数や実行時間への影響分析
(2) 様々な問題に対するニアカーネルベクトル抽出手法の有用性検証

5.2.1 実験 1

実験 1 では，ニアカーネルベクトルの抽出本数の変化による反復回数や実行時間への影響の分析を行った．本実験では，1 プロセスと 64 プロセスの，2 つの環境下においてそれぞれ計測を行った．Algorithm 5 において，抽出の際に固有値解析を実施する必要があるが，今回の実験では，1 プロセスにおいては Lapack ライブラリの固有値計算法の関数（dsyev）[15]，64 プロセスでは，Scalapack ライブラリの固有値計算法の関数（pdsyev）[16] と，正定値対称な疎行列向けの固有値計算法である Lanczos 法を 300 反復適用した場合の 3 つにおいて，それぞれ結果を示す．また，固有値解析を実施するレベル（Algorithm 5 の $\hat{L}$）を 2 に設定しており，実験 2 についても同様とする．

実験 1 における比較対象を表 1 に示す．また，本実験で

表 1　実験 1 における比較対象

表記	内容
先行研究手法 1	Algorithm 3 を使用
先行研究手法 2	Algorithm 4 を使用
提案手法（粗いレベルなし）	Algorithm 5
提案手法（粗いレベルあり）	Algorithm 5+ Algorithm 6

表 2　第 1 レベルのニアカーネルベクトル抽出本数

表記	内容
3p	平行移動成分 (X, Y, Z)
6p	3p + 回転成分 (X, Y, Z)
3p+1,...	3p + レベル 1 で抽出された ニアカーネルベクトル (最大本数: 7)

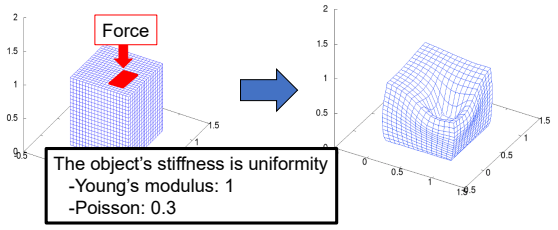

図 6　実験 1 で使用した 3 次元弾性体の問題設定

は第 1 レベルのニアカーネルベクトルに関しては，表 2 に示すような本数の抽出および設定をそれぞれ行った．表 2 の 3p と 6p は弾性体問題の問題設定から予想されるニアカーネルベクトルである．また，第 1 レベルにて抽出したニアカーネルベクトルを用いる際には，3p に追加する形で設定を行っている（3p+1，3p+2，...）．「先行研究手法 1」と「提案手法（粗いレベルあり）」に関しては，粗いレベルでのニアカーネルベクトル設定本数によりさらに性能が変化する．そのため，実験結果を示す際には，粗いレベルで最良の本数を設定したときの結果を示している．

本実験では 3 次元弾性体の問題を使用した．この問題は，ある物体に対して力を加えたときの，物体の変形量を解く問題となっている．数値実験で用いた弾性体の問題設定を**図 6** に示す．この弾性体の問題は図 6 のように，均質な立方体の物体に対して，ある一部分に力を加えたとき，どのように変形するかを解く問題となっている．問題サイズについては，1 プロセスあたり 15×15×15 のウィークスケーリング（Weak Scaling）方式による計測を行った．また，問題行列の各プロセスへの分割は，各軸方向へ問題を均等に分割し，それにより作成された小行列を各プロセスへ分配する単純な方法で分割を行った．

5.2.2　実験 2

実験 2 では他問題に対する有用性検証のため，Texas A&M 大学の SuiteSparse Matrix Collection[17] から取得した 6 個の問題で計測した．使用した問題を表 3 に示す．表 3 は，行数の小さい順に問題を並べている．実験環境は

表 3　使用した問題リスト

問題名	列数	要素数	条件数
ex10	2,410	54,840	9.10e+11
bcsstk28	4,410	219,024	9.45e+08
s1rmq4m1	5,489	262,411	1.81e+06
fv2	9,801	87,025	8.81e+00
s3dkq4m2	90,449	4,427,725	-
af_shell1	504,855	17,562,051	-

実験 1 と同様のものを用い，すべて逐次で実行した際の結果を示す．提案手法における固有値解析は，すべて Lapack ライブラリの固有値計算法の関数（dsyev）[15] を用いた．

5.3　実験結果

5.3.1　実験 1

まず，1 並列時の結果を図 7 から図 9 に示す．3 つの図はそれぞれ各手法を用いた時の抽出時間，反復回数，構築部と求解部の実行時間を合計した全体実行時間と求解部のみの実行時間をそれぞれ示している．まず，図 7 より，先行研究手法 1 や 2 では，抽出時間が本数とともに大きく増加してしまっていることがわかる．しかし，提案手法を用いることで，抽出時間を少なく抑えることができ，3p+7 において約 90%程度の改善効果がみられることが分かった．表 4 に各レベルにおける未知数個数を示す．表 4 からわかるように，提案手法において実際に固有値解析を実施しているレベル 2 では，本実験の問題設定においては未知数個数が 125 程度と小さくなる．そのため，抽出時間を低く抑えることが可能となる．また，図 8 と図 9 より，提案手法は，著者らによる先行研究での抽出手法と比べ，ほぼ同等の収束性能や実行時間削減効果を示していることがわかる．これより，1 並列においては，提案手法を用いることで，著者らによる先行研究での抽出手法と比べ，低コストで比較的性質のよいニアカーネルベクトルの抽出が行えることがわかった．

次に，64 並列における結果を示す．本実験では，抽出時間と収束性のみに着目する．実験結果を図 10 と図 11 に示す．図 10 と図 11 はそれぞれ抽出時間と反復回数を，各手法において実行した際の結果を示している．まず図 10 より，Scalapack を用いた手法は，著者らによる先行研究での抽出手法やと比べ大きく抽出時間がかかってしまっていることがわかる．これは，本実験はウィークスケーリングなため，2 レベル目においても未知数個数が十分な粗さを確保できていないことに加え，Scalapack では密行列を対象としており，疎行列から密行列への変換および固有値解析自体に大きなオーバーヘッドがかかってしまったためである．一方，Lanczos 法による手法では，著者らによる先行研究での抽出手法と比べても抽出時間が低く抑えられていることがわかる．これは，Lanczos 法は疎行列向け固

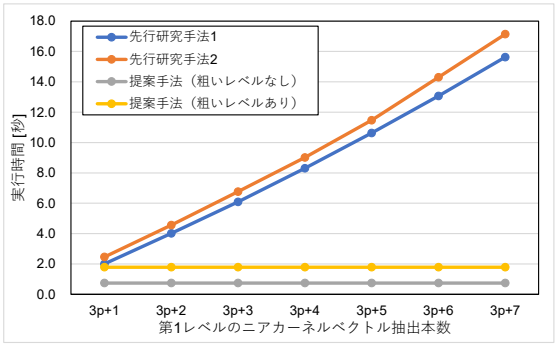

図 7　実験 1：1 並列時の抽出時間（凡例の詳細は表 1 に記載）

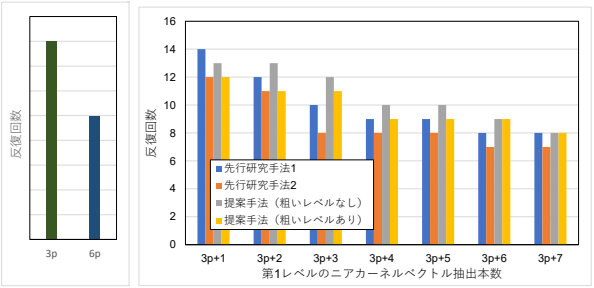

図 8　実験 1：1 並列時の反復回数（内容は図 7 と同様）

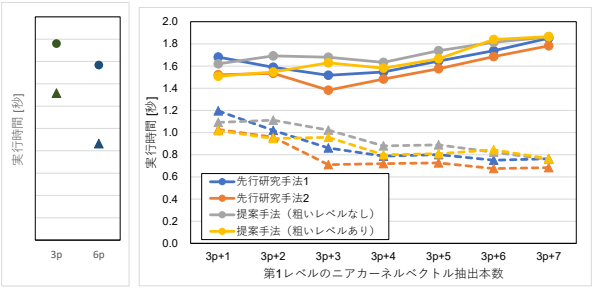

図 9　実験 1：1 並列時の全体実行時間（下図の実線（丸マーカ）は全体（構築部＋ 求解部），破線（三角マーカ）は求解部のみ）

表 4　実験 1：1 並列時の各レベルの未知数個数

Level	# of DOF
1	3375
2	125
3	6

有値計算法であり，Scalapack を用いた場合と比べ低コストで抽出できたためである．次に，図 11 に着目すると，Lanczos 法を使用した手法では若干悪化しているものの，どの手法もほぼ同様の収束性能を示すことがわかった．本研究で用いた Lanczos 法は 300 反復適用した際の近似的な結果であり，Scalapack は QR 法を用いた厳密な固有値と固有ベクトルの解である．そのため，Lanczos 法を用いた手法にみられた収束性の悪化に関しては，Restart 付きの Lanczos 法などを用い解の精度を高めることで，さらに改善すると考えられる．

5.3.2　実験 2

本実験により得られた反復回数，抽出時間と構築部と求

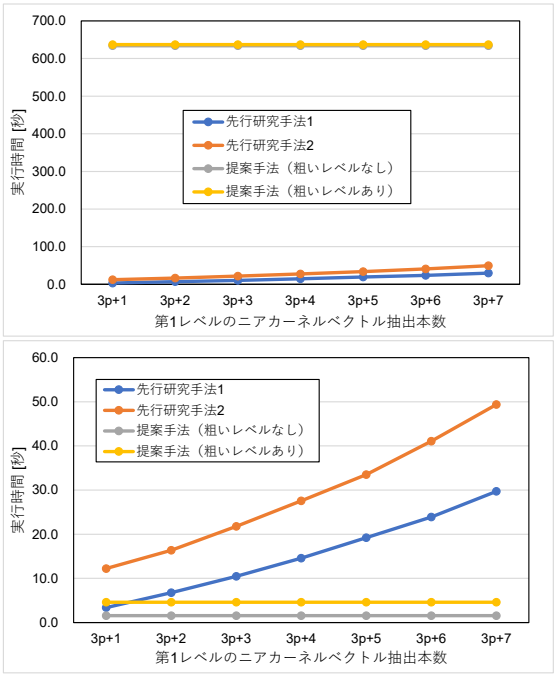

図 10　実験 1：64 並列時の抽出時間（上：Scalapack の関数を使用，下：Lanczos 法を使用）

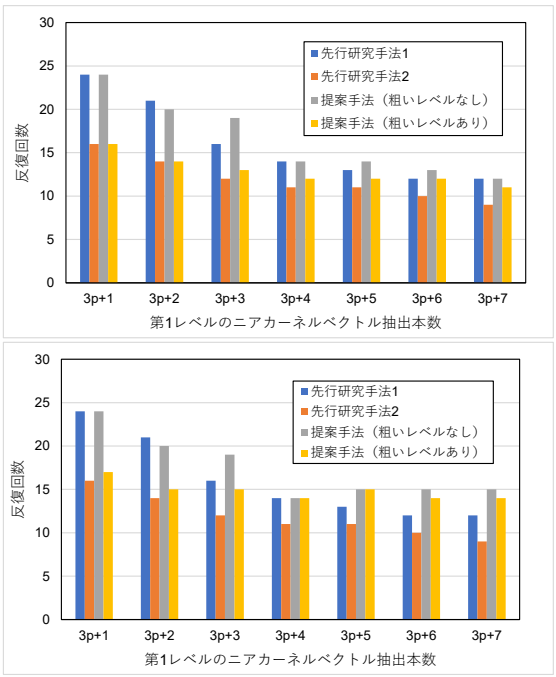

図 11　実験 1：64 並列時の反復回数（上：Scalapack の関数を使用，下：Lanczos 法を使用，凡例の詳細は表 1 に記載）

解部双方を足した全体の実行時間，求解部のみの実行時間，さらに抽出時間をそれぞれ表 5 から表 8 に示す．それぞれの表において，SGS は対称 Gauss-Seidel 法を前処理に用いた CG 法を示す．また，他は SA-AMG 前処理付 CG 法において，抽出なしはニアカーネルベクトルとしてすべての要素が 1 の定数ベクトルを使用，先行 2 は Algorithm 4，提案は Algorithm 5 と Algorithm 6 の手法を用いてニアカーネルベクトルの抽出を行った時の，最良の結果となってい

表 5　各問題における各手法による反復回数（SGS：対称 Gauss-Seidel 前処理付 CG 法，抽出なし：ニアカーネルベクトルとして全要素が 1 の定数ベクトルを使用，先行 2：Algorithm 4 の最良値，提案：Algorithm 5+Algorithm 6 の最良値）

問題名	SGS	抽出なし	先行 2	提案
ex10	369	63	36	48
bcsstk28	1,543	613	355	475
s1rmq4m1	282	120	29	72
fv2	16	5	3	3
s3dkq4m2	3574	660	275	429
af_shell1	収束せず	418	86	157

る．まず，表 5 において，提案手法を用いることで，他のニアカーネルベクトルを抽出しない手法と比べ改善されていることがわかる．これは，ニアカーネルベクトル抽出による収束性削減の効果によるものである．次に，表 6 に着目すると，提案手法が他の手法と比べ比較的良い結果を示すことがわかる．これは，表 6 や表 8 より，収束性改善に伴う解法適用時間削減も要因のひとつであるが，とりわけ提案手法の抽出時間の改善効果が大きく，これが表れたものであると考えられる．ここで，表 6 と表 7 に着目すると，問題サイズが比較的小さい問題では，提案手法の実行時間に優位性があまり見られないことがわかる．これは，1 反復の計算量が小さく，収束性改善効果より，ニアカーネルベクトル設定に伴う計算コストが大きくなったためであると考えられる．例外として，ex10 は提案手法が有用であるが，これは反復回数が 85%から 90%と大きな削減効果を示しており，結果として計算コスト増加よりも改善効果が大きいためであると考えられる．また，比較的大きい問題においては，提案手法が優れており，特に af_shell1 においては，SGS を前処理とした単純な CG 法では収束しなかったが，提案手法により，大きく反復回数を削減することに成功している．また，今回の 2 つの提案手法で比較を行うと，基本的に先行 2 のほうが提案よりも反復回数や実行時間の面で有用であることがわかる．しかし抽出時間に着目すると，表 8 からわかるように，提案手法のほうが低く抑えられることがわかる．例外として af_shell1 は提案手法のほうが抽出時間が遅くなっているが，これはレベル 2 においても十分に粗くならなかったため，固有値解析の実施に時間がかかってしまったためである．これに関しては，固有値解析を実施するレベル（Algorithm 5 の $\hat{L}$）を今回は 2 に設定したが，3 以降にする，または別の固有値計算法を用いるなどの対策をする必要がある．

6. 結論

本研究では抽出のさらなる効率化のため，粗いレベルで固有値解法を用い補間を行うことでニアカーネルベクトルを抽出する手法を用い，抽出されたニアカーネルベクトルを用いることによる反復回数や実行時間，および抽出時間への影響の分析を行った．その後，本手法に対し Texas A&M 大学の SuiteSparse Matrix Collection より取得した問題に対して適用し，様々な問題に対する有用性の検証を行った．本研究により，提案手法を用いることで，従来用いていた手法と比べ抽出コストを抑えつつ，従来手法と同等の高い収束性を得られることがわかった．また，様々な問題に対しても同様の有用性がみられることがわかった．しかし，用いる固有値解析手法やパラメタ設定により反復回数や実行時間に影響を与えることもわかった．

今後の課題として，まず提案手法に関するさらなる分析を行うことが考えられる．用いる固有値解析手法やパラメタ設定により反復回数や実行時間に影響を与えることから，適切な固有値解析手法やパラメタ設定が存在すると考えられる．そのため，分析および手法のさらなる改良が必要であると考えられる．また，本研究では合計 7 個の問題行列を対象として分析を行ったが，他問題に対しても同様の有用性がみられるかを検証していく必要がある．そのうえで，上記の分析結果を踏まえ，新たなニアカーネルベクトル抽出手法の提案を行うことが必要となると考えられる．

表 6　各問題での各手法による総実行時間（抽出+構築部+求解部）

問題名	SGS	抽出なし	先行 2	提案
ex10	5.53e-01	3.23e-01	1.12e+00	2.98e-01
bcsstk28	7.63e+00	9.77e+00	8.76e+00	7.93e+00
s1rmq4m1	1.76e+00	2.56e+00	5.30e+00	1.94e+00
fv2	4.41e-02	1.02e-01	4.56e+00	8.87e-01
s3dkq4m2	3.69e+02	2.41e+02	1.93e+02	1.82e+02
af_shell1	-	5.73e+02	4.36e+02	1.57e+03

表 7　各問題における各手法による求解部実行時間

問題名	SGS	抽出なし	先行 2	提案
ex10	5.53e-01	3.02e-01	1.78e-01	2.29e-01
bcsstk28	7.63e+00	9.71e+00	5.78e+00	7.75e+00
s1rmq4m1	1.76e+00	2.46e+00	6.79e-01	1.65e+00
fv2	4.41e-02	5.50e-02	4.61e-02	4.56e-02
s3dkq4m2	3.69e+02	2.392+02	1.10e+02	1.71e+02
af_shell1	-	5.68e+02	1.34e+02	2.46e+02

表 8　各問題における各手法による抽出時間

問題名	先行 2	提案
ex10	9.08e-01	4.44e-02
bcsstk28	2.85e+00	5.17e-02
s1rmq4m1	4.37e+00	6.67e-02
fv2	4.31e+00	6.36e-01
s3dkq4m2	7.80e+01	6.16e+00
af_shell1	2.87e+02	1.31e+03

付　　録

本章では，真のニアカーネルベクトルを適切に表現できているか，および収束性への影響について分析を行った結果を示す．式 (1) より，真のニアカーネルベクトル（$\boldsymbol{v}_1, \boldsymbol{v}_2, ...$ とする）と抽出されたニアカーネルベクトル（$\boldsymbol{e}_1, \boldsymbol{e}_2, ...$ とする）は，必ずしも $\boldsymbol{v}_1 \approx \boldsymbol{e}_1$,$\boldsymbol{v}_2 \approx \boldsymbol{e}_2$ のように一致することは必要ではなく，

$$\boldsymbol{e}_1 \in \mathrm{span}(\boldsymbol{v}_1, \boldsymbol{v}_2, ...) \tag{A.1}$$

のように，真のニアカーネルベクトルが張る空間に抽出されたニアカーネルベクトルが所属すればよいと予想できる．式 (A.1) の仮定を基にすると，抽出されたニアカーネルベクトルは，

$$\begin{aligned} \boldsymbol{e}_1 &= c_1^1 \boldsymbol{v}_1 + c_2^1 \boldsymbol{v}_2 + ... + c_k^1 \boldsymbol{v}_k + \boldsymbol{r}_1 \\ \boldsymbol{e}_2 &= c_1^2 \boldsymbol{v}_1 + c_2^2 \boldsymbol{v}_2 + ... + c_k^2 \boldsymbol{v}_k + \boldsymbol{r}_2 \end{aligned} \tag{A.2}$$

（ただし，$k, c_1^1, c_1^2, ...$ は適切な変数，$\boldsymbol{r}_1$ はベクトル $\boldsymbol{v}_k$ までの線形結合で表せなかった残りの成分）のように，線形結合で表すことができる．ここで，真のニアカーネルベクトル $\boldsymbol{v}$ はそれぞれ独立かつ正規化されていると仮定すると，変数 c は，

$$\begin{aligned} c_1^1 &= \boldsymbol{e}_1 \boldsymbol{v}_1 \\ c_2^1 &= \boldsymbol{e}_1 \boldsymbol{v}_2 \end{aligned} \tag{A.3}$$

のように算出可能である．

式 (A.2) 内のベクトル $\boldsymbol{r}_1$ は，$\boldsymbol{e}_1 \in \mathrm{span}(\boldsymbol{v}_1, \boldsymbol{v}_2, ...)$ とならなかった余分な成分であり，抽出されたニアカーネルベクトル内に存在する適切でない成分である．そこで，本実験ではベクトル $\boldsymbol{r}_1$ のノルムを算出することで，抽出されたベクトルが適切にニアカーネルベクトル成分を表現できているかの検証を行った．具体的には，式 (A.2) と式 (A.3) より，

$$\boldsymbol{r}_1 = \boldsymbol{e}_1 - (c_1^1 \boldsymbol{v}_1 + c_2^1 \boldsymbol{v}_2 + ... + c_k^1 \boldsymbol{v}_k) \tag{A.4}$$

であることがわかる．本実験では，式 A.4 の 2 ノルムをとり，抽出されたニアカーネルベクトル内に存在する適切でない成分 $\boldsymbol{r}_1$ の大きさを算出することで，妥当性の検証を行った（以下，$\boldsymbol{r}_1$ のノルムをエラーノルムとする）．ここで，変数 k は式 (A.1) における空間の次元数を示しており，k によって比較対象である真のニアカーネルベクトルのなす空間が決定する．そのため，本実験の結果は変数 k により変化することが考えられるが，本研究ではこのことについての分析は行わず，k を 20 と 30 で設定し実験を行った．

まず，先行研究手法 1 における実験結果を示す．表 A·1 に本実験の結果を示す．表 A·1 と図 A·1 に着目すると，エラーノルムが少ないほど，収束性の改善効果が大きく見えることがわかる．例えば 3p+3 から 3p+4 においては，エラーノルムが k が 20 と 30 の場合双方とも大きく増加しており，それに伴い収束性の改善も悪化していることがわかる．これより本実験においては，式 (A.4) により計算されたエラーノルムと収束性に，相関があることがわかった．

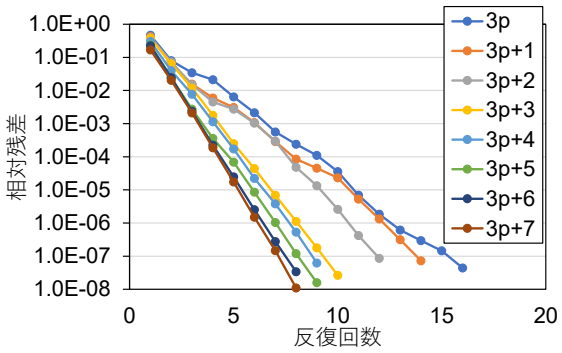

図 A·1　抽出したニアカーネルベクトルを用いた際の相対残差履歴：先行研究手法 1

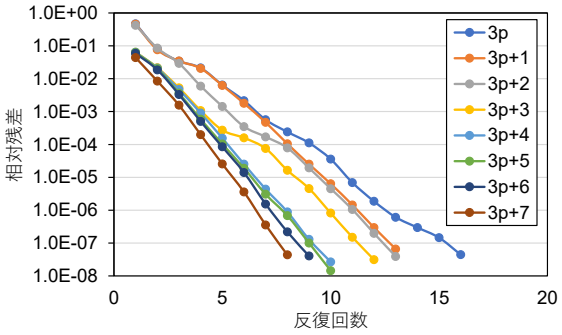

図 A·2　抽出したニアカーネルベクトルを用いた際の相対残差履歴：提案手法

次に，提案手法における実験結果を示す．表 A·2 に本実験の結果を示す．表 A·2 と図 A·2 から，先行研究手法 1 での検証実験にみられたような強い相関性はみられないという結果となった．エラーノルムは k の値により変化することから，そこで，様々な k により実験を行い，有意な特徴がみられるかの実験を行った．実験結果を表 A·3 に示す．表 A·3 より，本実験においては k を 10 に設定した場合に，比較的収束性とエラーノルムの相関性がみられることがわかった．しかしニアカーネルベクトルが張る空間と収束性への影響についての分析は不十分であり，今後の課題としてそれらの具体的な相関について，さらなる分析が必要であると考えられる．また，変数 k の設定，および本実験では 3 次元弾性体問題のみ扱ったが，他の問題に適用した際の分析なども必要であると考えられる．

参考文献

[1] Vaněk, P.: Fast multigrid solver, *Applications of Mathematics*, Vol. 40, No. 1, pp. 1–20 (1995).
[2] Pereira, F. H., Verardi, S. L. L. and Nabeta, S. I.: A fast algebraic multigrid preconditioned conjugate gradient solver, *Applied mathematics and computation*, Vol. 179, No. 1, pp. 344–351 (2006).

表 **A·1** ニアカーネルベクトル検証実験結果：先行研究手法 1

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
エラーノルム（$k = 20$）	2.77e-02	2.78e-02	2.21e-02	9.57e-02	1.27e-01	1.27e-01	2.59e-01
エラーノルム（$k = 30$）	1.76e-02	2.39e-02	1.50e-02	7.20e-02	7.50e-02	9.82e-02	1.45e-01
反復回数	14	12	10	9	9	8	8

表 **A·2** ニアカーネルベクトル検証実験結果：提案手法

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
エラーノルム（$k = 20$）	3.73e-02	3.83e-02	2.59e-02	6.80e-02	7.25e-02	9.19e-02	9.16e-02
エラーノルム（$k = 30$）	2.77e-02	2.92e-02	2.47e-02	6.59e-02	5.10e-02	6.88e-02	5.79e-02
反復回数	13	13	12	10	10	9	8

[3] Chan, T. F. and Vaněk, P.: Multilevel algebraic elliptic solvers, *International Conference on High-Performance Computing and Networking*, Springer, pp. 999–1014 (1999).
[4] Blaheta, R.: A multilevel method with correction by aggregation for solving discrete elliptic problems, *Aplikace matematiky*, Vol. 31, No. 5, pp. 365–378 (1986).
[5] Vaněk, P., Mandel, J. and Brezina, M.: Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, *Computing*, Vol. 56, No. 3, pp. 179–196 (1996).
[6] Nomura, N., Fujii, A., Tanaka, T., Nakajima, K. and Marques, O.: Performance Analysis of SA-AMG Method by Setting Extracted Near-Kernel Vectors, *International Conference on Vector and Parallel Processing*, Springer, pp. 52–63 (2016).
[7] Nomura, N., Fujii, A. and Nakajima, K.: The Study of the Automatic Extraction Method of Near-kernel Vector for High Scalable and Stable SA-AMG Method, *IPSJ Transactions on Computing Systems*, Vol. 12, No. 3, pp. 1–18 (2019).
[8] Briggs, W. L., Henson, V. E. and McCormick, S. F.: *A multigrid tutorial*, Vol. 72, Siam (2000).
[9] Tamstorf, R., Jones, T. and McCormick, S. F.: Smoothed aggregation multigrid for cloth simulation, *ACM Transactions on Graphics (TOG)*, Vol. 34, No. 6, p. 245 (2015).
[10] Brezina, M., Falgout, R., MacLachlan, S., Manteuffel, T., McCormick, S. and Ruge, J.: Adaptive smoothed aggregation (αSA), *SIAM Journal on Scientific Computing*, Vol. 25, No. 6, pp. 1896–1920 (2004).
[11] Brandt, A., Brannick, J., Kahl, K. and Livshits, I.: Bootstrap amg, *SIAM Journal on Scientific Computing*, Vol. 33, No. 2, pp. 612–632 (2011).
[12] 野村直也，中島研吾，藤井昭宏：SA-AMG 法における収束性安定化のための効率的なニアカーネルベクトル抽出手法に向けた研究，研究報告ハイパフォーマンスコンピューティング (SWoPP2019)， Vol. 2019, No. 20, pp. 1–10 (2019).
[13] for Advanced High Performance Computing (JCAHPC), J. C.: `https://ofp-www.jcahpc.jp/`, (accessed: Dec. 6, 2019).
[14] Hestenes, M. R. and Stiefel, E.: *Methods of conjugate gradients for solving linear systems*, Vol. 49, No. 1, NBS Washington, DC (1952).
[15] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D.: *LAPACK Users' guide*, Vol. 9, SIAM (1999).
[16] Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A. et al.: *ScaLAPACK users' guide*, Vol. 4, SIAM (1997).
[17] Davis, T. A. and Hu, Y.: The University of Florida sparse matrix collection, *ACM Transactions on Mathematical Software (TOMS)*, Vol. 38, No. 1, p. 1 (2011).

表 **A·3** ニアカーネルベクトル検証実験結果（k の設定による実験結果）：提案手法

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
$k = 10$	4.22e-02	4.26e-02	3.36e-02	9.65e-02	1.72e-01	3.27e-01	1.96e-01
$k = 20$	3.73e-02	3.83e-02	2.59e-02	6.80e-02	7.25e-02	9.19e-02	9.16e-02
$k = 30$	2.77e-02	2.92e-02	2.47e-02	6.59e-02	5.10e-02	6.88e-02	5.79e-02
$k = 100$	1.58e-02	1.66e-02	1.24e-02	3.95e-02	3.15e-02	3.17e-02	3.63e-02