Vol.2019-HPC-170 No.20
2019/7/25

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

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を高速に解く解法としてSA-AMG法がある．SA-AMG法は与えられた問題行列から，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，それらを用いることで高い収束性を実現している．ここで，粗い問題を生成する際に問題に応じたニアカーネルベクトルを用いることで，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．本研究では，ニアカーネルベクトルのより効率的な抽出手法の検討を行った結果を報告する．最上階の問題行列のような大規模問題に対し，直接固有値解析手法を用いると膨大な時間が必要となる．そこで本発表では効率的に抽出するため，粗いレベルにおいて0固有値に近い固有ベクトルを求め，補間演算子を用いて細かいレベルへの補間を行うことで，複数階層のニアカーネルベクトルを抽出する手法を提案し，評価を行った．この手法により，従来手法と比べ抽出時間を抑えつつ，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，工学における設計現場，気象予報，渋滞予測などの身近な場面で幅広く活用されている．このような問題の多くは最終的には連立一次方程式 $\boldsymbol{Ax}=\boldsymbol{b}$ 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，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法が提案されている．Multigrid法は係数行列Aから階層的に粗いレベルの行列を生成し，各波長成分を効率よく減衰させることによって，高い収束性を実現している．Multigrid法には様々な派生形が存在しており，本研究ではその中のAlgebraic Multigrid（AMG）法を対象としている[1]．AMG法はさらに，階層行列の生成方法により様々な手法が存在しており，本研究ではSmoothed Aggregationに基づくAMG（SA-AMG）法を用いている[2][3]．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，問題生成部（以下，構築部）と，反復解法部（以下，解法部）に分かれている．構築部では，問題行列に基づくグラフ構造を作成し，それを基にアグリゲートと呼ばれる節点集合を作成する．これを再帰的に行うことで，次元数のより小さい複数の行列を作成する．解法部では，構築部で階層的に生成された行列を用い，緩和法を用いて問題行列を解く．元の問題行列などの大規模な問題が設置される階層（レベル）を細かいレベル，問題行列から生成された小規模な行列が設置される階層（レベル）を粗いレベルと呼ぶ．

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法ではこのような誤差成分を効率よく減衰させるため，構築部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．これにより，粗いレベルの行列に誤差成分

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

が射影され，誤差成分を効率よく減衰させることが可能となる．その結果，残差を効率よく収束させることができる．

本研究では 3 次元弾性体の問題を用いて実験を行っており，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．著者等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，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 [4][5].

本研究では別のアプローチとして，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の提案を行った．そして，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．

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，Gauss-Seidel 法のような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．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．本章ではまず，Multigrid 法の中の解法のひとつである AMG 法についての説明を行い，次に AMG 法の派生解法である SA-AMG 法についての説明を行う．

2.2 AMG 法

AMG 法は上記でも述べた通り，Multigrid 法の中の解法のひとつである．Multigrid 法では一般に大きく分けて，与えられた問題行列を複数段階に分けて小規模な行列を生成し（構築部），これらを用いて問題行列を解く（解法部），

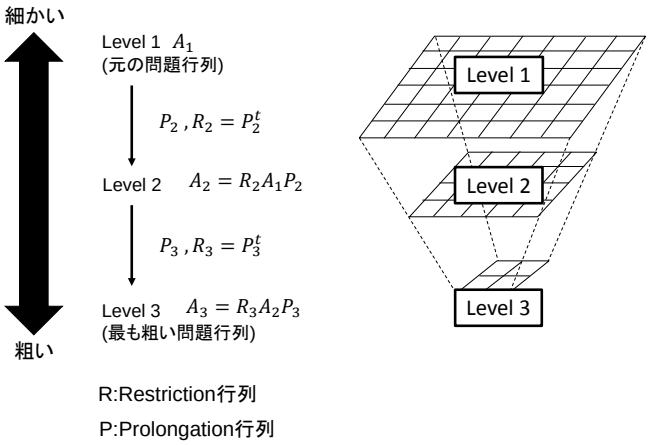

図 1 構築部の概要

2 つの処理からなる．以下の 2.2.1 節と 2.2.2 節において，AMG 法における構築部と解法部の流れを説明する．

2.2.1 構築部

AMG 法の構築部の概要を**図 1** に示す．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．図 1 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．AMG 法では一般に，粗いレベルの問題行列は，横長の Restriction 行列と縦長の Prolongation 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．また一般に，$R = P^t$ と設定する．

2.2.2 解法部

次に，解法部の構造を**図 2** に示す．解法部は，主に行列ベクトル積と緩和法から成り立つ．この図のように，最上層に与えられた問題行列（A_1）が設置され，階層が下がるにつれて，問題行列より小規模な行列（$A_2, A_3, ...$）が設置される．階層数は問題サイズによって可変となる．解法部の計算を Algorithm 1 に示す．

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．複数の階層を行き来する様子が V 字を連想させるため，このような解法部を V-cycle と呼ぶ．

補間演算子から行列の階層構造が生成されるため，Multi-

Algorithm 1 解法部（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: $\boldsymbol{x}_{l+1} \leftarrow S_l(\boldsymbol{x}_l, \boldsymbol{b}_l)$

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

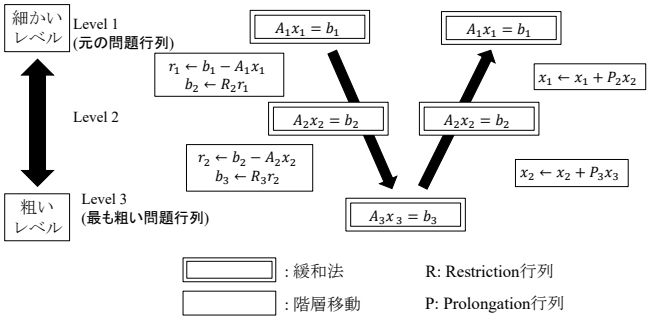

図 2 解法部の概要（V-cycle）

grid 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 [2]．本研究ではその中でも SA-AMG 法を対象とする．この手法では，問題行列のみから未知数間の依存関係を定義する．そして，依存関係のある未知数同士で集合を作り，その集合内で重みづけをして補間演算子を生成する．その後，生成された補間演算子を基に，集約を行う．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．

2.3 SA-AMG 法

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．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．またその上で，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する必要がある．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．この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を計算し，行列の階層構造を作成する．補間演算子である Prolongation 行列と Restriction 行列は，行列の階層構造を作成する際に用いられる．著者らの研究から，これらの行列の生成にニアカーネルベクトルを用いることで，収束性をさらに高められることがわかっている．このことについては，次章の 3.1 節および 3.2 節にて詳しく述べる．

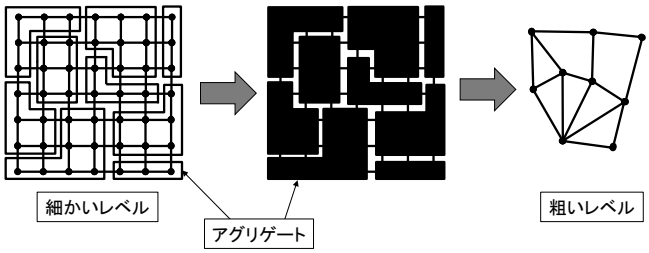

図 3 アグリゲート生成の概要

3. ニアカーネルベクトル

3.1 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 [2][3].

SA-AMG 法では，補間演算子である Prolongation 行列と Restriction 行列にニアカーネルベクトルを用いることで，粗いレベルに収束しにくい成分を移動させることができる．これにより，粗いレベルで収束しにくい成分を効率よく減衰させることができ，残差を効率よく収束させることが可能となる [3]．つまり，SA-AMG 法においてはニアカーネルベクトルの設定により収束性が大きく左右されるため，Multigrid 法の特徴である高い収束性能の実現のためには，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 [6]．例えば，本研究で用いている弾性体の問題では，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．これについては，次節の 3.2 節にて，数値実験による結果を示す．

ニアカーネルベクトルの補間演算子への設定方法の概要を，**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．

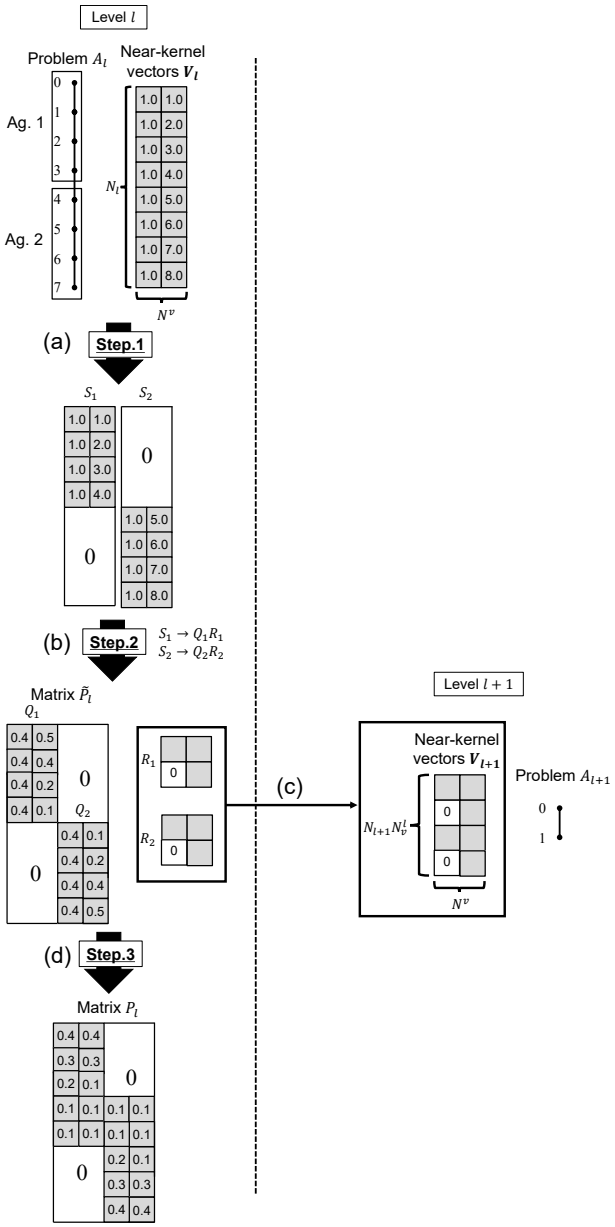

図 4　補間演算子（Prolongation 行列）の生成方法

3.2　事前実験

3.2.1　SA-AMG 法におけるニアカーネルベクトル設定による収束性検証

図 5 と**表 1**，および**図 6** に，SA-AMG 法においてニアカーネルベクトルを複数本設定することによる反復回数の変化を示すための，事前実験の問題設定と比較対象，およびその結果を示す．図 5 にこの実験で対象とした問題を示す．この問題は 3 次元弾性体問題である．弾性体の問題については，5.1 節で詳しく説明する．この弾性体の問題は，上半分が柔らかい物体に対して，ある一部分に力を加え，どのように変形するかを解く問題となっている．また，ヤング率を上半分が 0.8，下半分を 1 に設定している．反復の終了条件は相対残差が 1.0×10^{-7} となったときとした．また，問題サイズについては，1 プロセスあたり $6\times 15\times 60$ としたウィークスケーリングで計測を行った．さらに表 1 において，事前実験における比較対象を示す．表 1 に示すように，事前実験ではニアカーネルベクトルの設定本数により，3 種類の比較対象を用意し実験を行った．図 6 に，収束までに必要とした反復回数のグラフを示す．図 6 より，ニアカーネルベクトルを複数本設定することで，収束性の改善がみられることがわかる．これは，適切なニアカーネルベクトルが設定できているため，このような傾向がみられたと考えられる．本研究では数値実験において，ニアカーネルベクトルを問題行列から抽出することにより，これらのベクトルよりもさらに反復回数の改善が可能となる性質のよいニアカーネルベクトルが設定できるかの検証も行う．

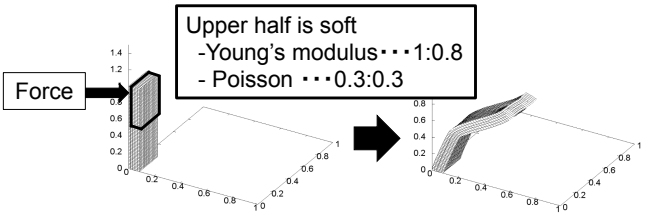

図 5　事前実験における問題設定

表 1　事前実験における比較対象

比較対象	ニアカーネルベクトル設定本数
Case 1	1（要素がすべて 1 の定数ベクトル）
Case 2	3（平行移動成分のみ）
Case 3	6（平行移動＋回転成分）

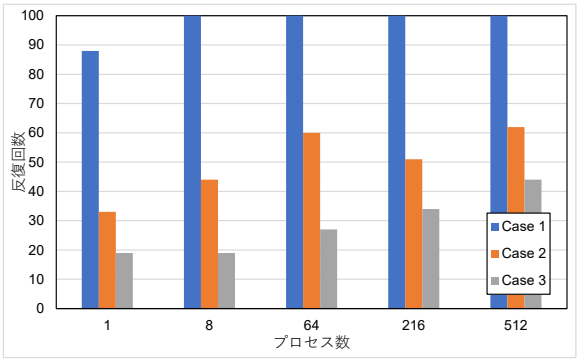

図 6　ニアカーネルベクトル設定による SA-AMG 法の反復回数の変化（事前実験結果）

3.3　ニアカーネルベクトル抽出手法に関する先行研究

3.3.1　関連研究

3.2 節から，ニアカーネルベクトルの設定が SA-AMG 法の収束性に大きくかかわることがわかる．そのため，どのようなニアカーネルベクトルを設定するかを決めることは，SA-AMG 法において重要なこととなる．通常，このニアカーネルベクトルの設定に関しては，問題の性質から予想されるベクトルを用いることで収束性の改善を図ることが多い [6]．[6] では，薄板弾性体問題において，SA-AMG 法の適用とその結果が報告されている．[6] における研究では，回転成分 $(0,-z,y),(z,0,-x),(-y,x,0)$（$(x,y,z)$ は節点要素の座標）がニアカーネルベクトルとして用いられている．しかし，このベクトルのみで十分なニアカーネルベクトルが設定できているとは限らない．また，これは問題設定に依存したものであり，すべての問題において，対象とする問題の物理的性質に基づき適切なニアカーネルベクトルを予測できるとは限らない．そこで，ニアカーネルベクトルを問題行列から抽出する手法が提案されてきた．この手法により，上記のようなニアカーネルベクトル設定の際に起こる問題を解消することができると考えられる．

ニアカーネルベクトルを抽出する手法に関しての関連研究はいくつか存在する．まず，Brezina M. らにより提案された αSA 法である [7]．αSA 法は，問題行列から V-cycle（Multigrid 法の解法部）を用いてニアカーネルベクトルを抽出する手法である．αSA 法は，1 本のベクトルに基づき全階層における行列を網羅するように，ニアカーネルベクトルを抽出する．

また，Brandt A. らにより提案された Bootstrap AMG 法と呼ばれる手法がある [8]．この手法ではまず，Multigrid 法の構築部の処理により生成された階層行列において，最下層の行列に対し固有値解法を適用し，固有値と固有ベクトルを算出する．その後，補間演算子である Prolongation 行列を用いて，算出された最下層での固有ベクトルを細かいレベルに移動し，ニアカーネルベクトルとして出力する．Bootstrap AMG 法の概要を Algorithm 2 に示す．Bootstrap AMG 法ではまず，最下層 L において，固有値解法を適用する（3 行目）．その後，補間演算子 P やスムーザを適用し（5〜9 行目），最上層であるレベル 1 のニアカーネルベクトルとして出力する．

Algorithm 2 Bootstrap AMG

```
for l = L to 1 do
  if l == L then
    W_L = {w_l^κ | A_l w_l^κ = λ_l^κ T_l w_l^κ, κ = 1, ..., k_e}
  else
    w_l^κ = P_{l+1}^l w_{l+1}^κ, λ_l^κ = λ_{l+1}^κ, κ = 1, ..., k_e
    for κ = 1 to k_e do
      Relax on (A_l − λ_l^κ T_l) w_l^κ = 0
      λ_l^κ = ⟨A_l w_l^κ, w_l^κ⟩_2 / ⟨T_l w_l^κ, w_l^κ⟩_2
    end for
  end if
end for
Output W_1 matrix as near-kernel vectors
```

L：最大レベル数
A_l：レベル l における問題行列 A
P_{l+1}^l：レベル $l+1$ から l へ補間を行う Prolongation 行列

Algorithm 3 著者らの先行研究 [4] におけるニアカーネルベクトル抽出手法（従来手法 1）

```
Given : B
Select : x
for n = 1 to extract_number do
  x ← Random()
  x̃ ← V_cycle^μ(Ax = 0)
  B ← [B, x]
  Multilevel_creation(B)
end for
Output B matrix as near-kernel vectors
```

$extract\_number$：抽出したい本数
A：与えられた問題行列 A
$V\_cycle^{\mu}(Ax=0)$：$Ax=0$ 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
$[B,\tilde{\boldsymbol{x}}]$：行列 B の最終列へのベクトル x 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

3.3.2　著者らによる先行研究

著者らは先行研究として，αSA 法を基に，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 [4]．Algorithm 3 に [4] における抽出手法の概要を示す．詳細は [4] にゆだねる．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．

SA-AMG 法では通常，細かいレベルのニアカーネルベクトルを基に，粗いレベルのニアカーネルベクトルを生成する．そのため，[7] や [4] 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ため，全階層でニアカーネルベクトルの本数が同じとなる．しかし著者らは，粗いレベルのニアカーネルベクトルは，細

Algorithm 4 [4] を改良したニアカーネルベクトル抽出提案手法 [5](従来手法 2)

Given : B_1
Select : x_1
for $level = 1$ to $max\_level - 1$ **do**
  for $n = 1$ to $extract\_number$ **do**
    $\tilde{\boldsymbol{x}}_{\boldsymbol{level}} \leftarrow Random()$
    $x_{level} \leftarrow V\_cycle^{\mu}(A_{level}\tilde{\boldsymbol{x}}_{\boldsymbol{level}} = 0)$
    $B_{level} \leftarrow [B_{level}, \boldsymbol{x}_{\boldsymbol{level}}]$
    $Multilevel\_creation(B_{level})$
  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
$extract\_number$: 抽出したい本数
A_{level} : レベル $level$ における問題行列 A
$V\_cycle^{\mu}(Ax = 0)$: $Ax = 0$ を対象に V-cycle を μ 回適用
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
$[B, \boldsymbol{x}]$: 行列 B の最終列へのベクトル x の追加
$Multilevel\_creation(B_l)$: 行列 B を基に, 補間演算子 $P_l, P_{l+1}, ...$, および階層行列 $A_l, A_{l+1}, ...$ を再生成

Algorithm 5 本研究で用いたニアカーネルベクトル抽出手法（レベル 1 のみ）

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

$\hat{L}$: 固有値解法適用の対象とする最大レベル以下の任意のレベル
A_l : レベル l における問題行列 A
P_{l+1}^{l} : レベル $l+1$ から l へ補間を行う Prolongation 行列

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抽出や追加に設定を行えるようにするべきではないかと考えた．そこで著者等はさらに，Algorithm 3 の手法を改良し，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を行った [5]．Algorithm 4 に概要を示す．Algorithm 4 の赤字箇所は，Algorithm 3 から追加された箇所である．Algorithm 4 から分かるように，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．

上記のようにニアカーネルベクトルを設定する手法はいくつか存在する．数値実験では比較対象として，これらの手法との比較を行う．次章より，本研究で提案するニアカーネルベクトル抽出手法について述べる．

4. ニアカーネルベクトル抽出手法の提案

4.1 本研究で提案するニアカーネルベクトル抽出手法

この節では，本研究で提案するニアカーネルベクトルを問題行列から抽出する手法について説明する．

3.3.2 節にて述べた従来手法では，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てしまう．これにより，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問題を解くコストと比べ，大きくなってしまう問題がある [5]．そこで，Bootstrap AMG 法 [8] や Algorithm 4 を基に，任意の粗いレベルにおいて固有値解法を適用し補間を行い，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．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 行目）．そして，そのレベルにおいて，Bootstrap AMG 法と同様に固有値解法を適用する（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 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 Algorithm 4 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 $W_1, ..., W_L$ をニアカーネルベクトルとして出力する．

このように，粗いレベルにおいて固有値解法を適用することで，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．また，粗いレベルにおいてもニアカーネルベクトルの抽出を行うことで，さらに高い収束性能を発揮できると考えられる．

5. 数値実験

本節では，提案手法を用いることによる数値実験の結果を示す．本節ではまず，本実験で用いた環境について述べ，その後，数値実験の内容とその結果を示す．

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

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

$\hat{L}$: 固有値解法適用の対象とする最大レベル以下の任意のレベル
L : 最大レベル数
A_l : レベル l における問題行列 A
R_l^{l+1} : レベル l から $l+1$ へ補間を行う Restriction 行列
$Multilevel\_creation(W)$: 行列 W を基に，補間演算子 $P_l, P_{l+1}, ...,$ および階層行列 $A_l, A_{l+1}, ...$ を再生成

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，16GB の MCDRAM と 96GB の DDR4 メモリを搭載している．

数値実験では最大 8 ノード使用し，計測を行った．また並列化手法については，1 コアに 1 プロセス起動するフラット MPI を使用した．本研究では、AMGS ライブラリ [9] を使用して実験を行っている．AMGS ライブラリは，著書らが研究開発を行っている，大規模な疎行列係数の線型方程式を AMG 法で解くライブラリである．解法部では CG 法 [10] を使用し，前処理として SA-AMG 法を適用している．SA-AMG 法における解法部の各レベルの緩和法として，対称ガウス・ザイデル法を 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．また，コンパイラは Intel®コンパイラ（mpiifort），コンパイラオプションは高速化のための最適化オプションである”-O3”と“-xMIC-AVX512”，OpenMP を用いるためのオプションである”-qopenmp”を使用した．

5.2 数値実験内容

本実験では，2 つの問題を用意し，それぞれにおいて実験を行った．まず，実験 1 として，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 7** に示す．この弾性体の問題は図 7 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．また，弾性体の問題において硬さを表すヤング率を，すべての部分で 1 に設定している．ヤング率は値が大きければ物体が硬いことを表している．また，ポアソン比（物体のひずみに関する係数）を 0.3 に設定した．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．ウィークスケーリングは，1 プロセスが担当する問題サイズが一定になるように問題サイズを設定するようにしたものである．そのため，プロセス数が増加するほど全体の問題サイズが増加する．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．実験 1 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法を用いる必要があるが，今回の実験では，1 プロセス時においては Lapack ライブラリの固有値解法の関数（dsyev）を用いた．また，64 プロセス時では，Scalapack ライブラリの固有値解法の関数（pdsyev）を適用した場合と，正定値対称な疎行列向けの固有値解法である Lanczos 法を 300 反復適用した場合の 3 つにおいて，それぞれ実験結果を示す．

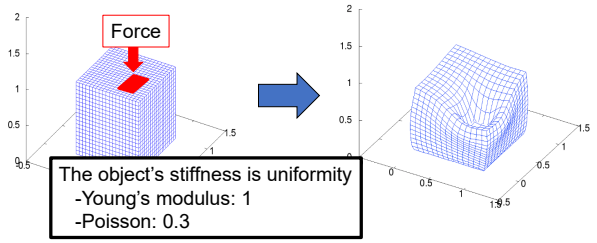

図 7 実験 1 で使用した 3 次元弾性体の問題設定

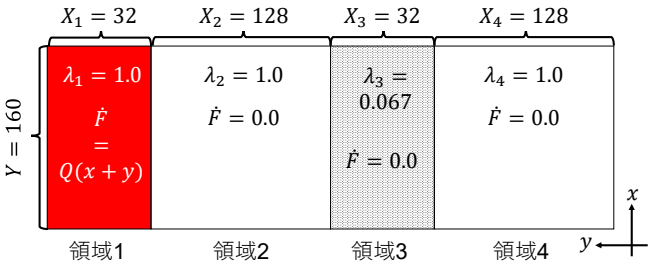

図 8 実験 2 で用いた 2 次元定常熱伝導問題の問題設定

次に実験 2 として，2 次元定常熱伝導問題（ただし，本実験では 4 面における領域境界 Γ 上で Dirichlet 境界条件を付している）に対し，ニアカーネルベクトルを複数本設定した際の有用性を示す．本実験で使用した問題の概要を**図 8** に示す．また，本問題は 1 プロセスのみ用いて計測を行い，Algorithm 5 の固有値解法は Lapack を用いた．

5.3 実験結果

5.3.1 実験 1

本実験では 5.2 節でも述べたように，3 次元弾性体問題

における実験結果を示す．本実験における比較対象を**表 2** に示す．本実験では比較のため，著者らの研究で従来用いていた Algorithm 3 と，Algorithm 4 の手法を適用した際の結果も示す．また，本実験では第 1 レベルのニアカーネルベクトルに関しては，**表 3** 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 3 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，...）．Algorithm 4 と Algorithm 6 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．

まず，1 並列時の結果を**図 9** から**図 11** 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，構築部と解法部の実行時間を合計した全体実行時間と解法部のみの実行時間をそれぞれ示している．まず，図 9 より，従来手法である Algorithm 3 や Algorithm 4 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，今回の手法である Algorithm 5 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，Algorithm 3 において実際に固有値解法を適用しているレベル 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，抽出時間を抑えられていることがわかる．また，**図 10** と図 11 より，Algorithm 5 は従来手法と比べ，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，今回の手法である Algorithm 5 や Algorithm 6 を用いることで，従来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．5.2 節でも述べたように，本実験では Scalapack の関数（pdsyev）と Lanczos 法の 2 種類を適用したときの結果を示す．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 12** と**図 13** に示す．図 12 と図 13 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 12 より，Scalapack を用いた手法は，従来手法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，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計算自体に大きなオーバーヘッドがかかってしまったものと考えられる．一方，Lanczos 法による手法では，従来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 法は疎行列向け固有値解法であり，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 13 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，

表 2　実験 1 における比較対象

比較対象	詳細
レベル 1 のみ	粗いレベルで抽出なし [4]
粗いレベルで抽出：+1，+5	Level2 以降の階層毎に 1（+1）または 5（+5）本抽出，追加設定

表 3　第 1 レベルのニアカーネルベクトル抽出本数

表記	詳細
3p	平行移動成分 X，Y，Z（3 本）
6p	3p（3 本）+回転成分 X，Y，Z（3 本）
3p+1,3p+2,...	3p（3 本）+第 1 レベルの抽出本数（最大 7 本）

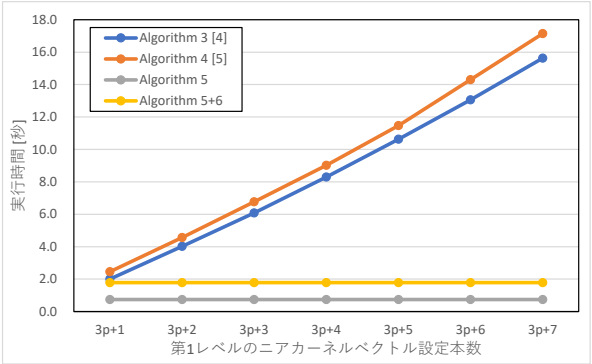

図 9　実験 1：1 並列時の抽出時間

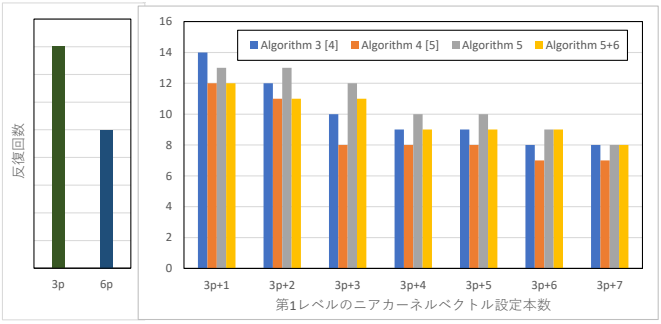

図 10　実験 1：1 並列時の反復回数

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．

以上より，本研究の提案手法である Algorithm 5 や Algorithm 6 を用いることで，抽出手法を抑えつつ，従来手法と同等の高い収束性能を発揮できることがわかった．また，今回は Algorithm 5 の固有値解法と適用する箇所において，Lapack の固有値解法関数と Lanczos 法のみを用いたが，その他の手法を適用することによる結果の分析も必要であると考えられる．次の実験では，他問題への有用性の検証として，2 次元定常熱伝導問題における実験結果を示す．

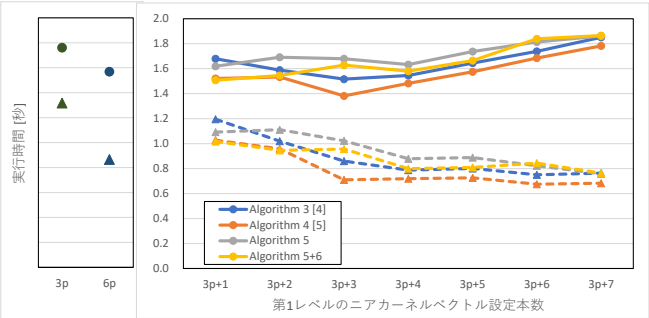

図 11 実験 1 : 1 並列時の全体実行時間（下図における実線（丸マーカ）は全体（構築部+ 解法部），破線（三角マーカ）は解法部のみ）

表 4 実験 1 : 1 並列時の各レベルの未知数個数

レベル	未知数個数
1	3375
2	125
3	6

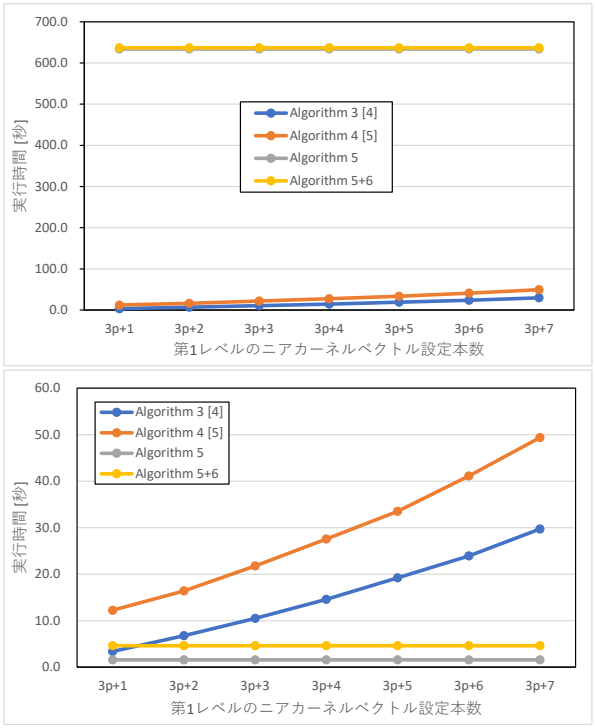

図 12 実験 1 : 64 並列時の抽出時間（上 : Scalapack の関数を使用，下 : Lanczos 法を使用）

5.3.2 実験 2

次に，実験 2 の結果を示す．本実験は 2 次元定常熱伝導問題における実験結果を示す．本実験では，1 コアのみを用いて実験を行った．また本実験では比較対象として，ポアソン方程式や拡散方程式のニアカーネルベクトルとして知られている定数ベクトル $(1, 1, ...)^T$ を，ニアカーネルベクトルとして設定した際の結果を 1p として記載している．結果を**表 5** と**表 6** に示す．表 5 はそれぞれの手法を用いた際の反復回数を示している．本実験では，Algorithm 5 の手法における固有値解法を適用する対象のレベルに関

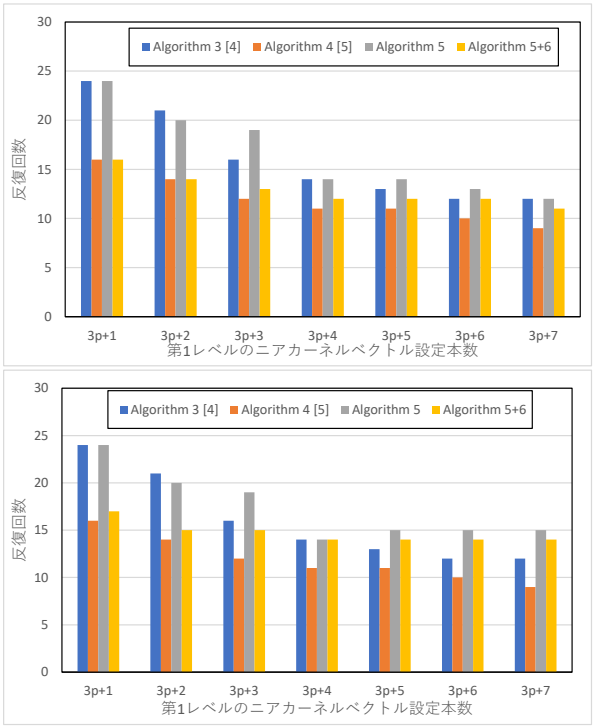

図 13 実験 1 : 64 並列時の反復回数（上 : Scalapack の関数を使用，下 : Lanczos 法を使用）

して，レベル 3 において適用した結果も示している．不均質性が強い問題では条件数が大きくなり，それに伴い収束性も悪化する．実際，表 5 の 1p では収束性の悪化が見受けられる．しかし，いずれの手法においても，ニアカーネルベクトルを適切に設定することで，高い収束性を発揮していることがわかる．以上より，ニアカーネルベクトルを適切に設定することで収束性が改善し，特に不均質性が強い，条件数が大きい問題に対して高い効果を発揮することがわかる．次に表 6 に着目する．表 6 は，それぞれのニアカーネルベクトル抽出手法において，ニアカーネルベクトル抽出にかかった時間を示している．表 6 より，本実験では Algorithm 5 の手法による抽出時間が，従来手法と比べ遅くなってしまっていることがわかる．ここで，**表 7** に着目する．表 7 は，各レベルにおける未知数個数を示している．表 7 と表 4 のレベル 2 を比較するとわかるように，本実験の問題設定では，レベル 2 においても未知数個数が多い．そのため，固有値解法の計算時間を多く必要としたため，Algorithm 5 の抽出時間が大きくなってしまっている．しかし，Algorithm 5 における固有値解法を適用するレベルを 1 つ下げ，レベル 3 において固有値解法を適用することで，抽出時間を削減することができることがわかる．一方，表 5 より，反復回数が若干悪化していることもわかる．このように，Algorithm 5 において固有値解法を適用するレベルによる，抽出時間と反復回数はトレードオフの関係があり，さらに分析する必要があると考えられる．

表 5　2 次元定常熱伝導問題：反復回数

	ニアカーネルベクトル設定本数				
	1p	1p+1	1p+2	1p+3	1p+4
Algorithm 3	81	4	4	3	3
Algorithm 4	—	4	4	3	3
Algorithm 5	—	6	4	4	3
Algorithm 5 + 6	—	6	4	4	3
Algorithm 5（レベル 3 で固有値解法適用）	—	6	5	4	3

表 6　2 次元定常熱伝導問題：抽出時間 [秒]

	ニアカーネルベクトル設定本数			
	1p+1	1p+2	1p+3	1p+4
Algorithm 3	5.20	11.14	18.17	26.61
Algorithm 4	19.72	30.10	43.24	57.58
Algorithm 5	379.68	379.68	379.68	379.68
Algorithm 5 + 6	385.36	385.36	385.36	385.36
Algorithm 5（レベル 3 で固有値解法適用）	2.33	2.33	2.33	2.33

表 7　実験 2：1 並列時の各レベルの未知数個数

レベル	未知数個数
1	51200
2	8550
3	884
4	131
5	21

6. 結論

本研究ではニアカーネルベクトル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の提案を行った．そして，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．本研究の手法を用いることで，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．本研究の手法では，抽出時に固有値解法を適用する必要がある．本研究では Lapack ライブラリの固有値解法関数（dsyev），並列時に Scalapack ライブラリの固有値解法関数（pdsyev）と正定値対称疎行列向け固有値解法である Lanczos 法を用いて実験を行ったが，各手法により抽出時間や抽出されたニアカーネルベクトルの性質に影響があることもわかった．そのため，他の固有値解法を用いることによる影響の分析も必要であると考えられる．

今後の課題として，上記でも述べたように，まず本研究の手法に用いる固有値解法に関する分析を行うことが考えられる．例として，今回は Lanczos 法を用いたが，派生解法である Thick-restart Lanczos 法 [11] などの解法を用いることが考えられる．そのうえで，他の問題に対して適用し，有用性の検証を行うことが考えられる．本研究の実験では，3 次元弾性体問題と 2 次元定常熱伝導問題のみを対象としたが，異方性のあるような，より悪条件な問題などに対して適用し，有用であるかを分析する必要があると考えている．

参考文献

[1] Pereira, F. H., Verardi, S. L. L., and Nabeta, S. I.: “A fast algebraic multigrid preconditioned conjugate gradient solver”, Applied Mathematics and Computation 179, pp.344-351, 2006.
[2] Chan, T. F. and Vanek, P.: “Multilevel algebraic Elliptic Solvers”, UCLA Math, Dept. CAM Report, 1999.
[3] Vanek, P., Mandel, J. and Brezina, M.: “Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems”, Computing, Vol.56, pp.179-196, 1998.
[4] Nomura, N., Fujii, A., Tanaka T., Nakajima, K. and Marques, O.: “Performance Analysis of SA-AMG Method by Setting Extracted Near-kernel Vectors”, VECPAR2016 (2016).
[5] 野村直也, 藤井昭宏, 中島研吾：“高並列環境下に向けた粗いレベルでニアカーネルベクトルを抽出および追加的に設定する手法の性能評価”，SWoPP2017，秋田市，秋田，2017.7.26-28.
[6] R. Tamstorf, T. Jones and S. F. McCormick: “Smoothed Aggregation Multigrid for Cloth Simulation”, ACM Trans. Graph., 34-6, pp. 245:1–245:13 (2015).
[7] Brezina, M., Falgout, R., Maclachlan, S., Manteuffel, T., Mccormick, S. and Ruge, J.: Adaptive Smoothed Aggregation (αSA), SIAM J. Sci. Comput, Vol. 25, No.6, pp.1896-1920 (2004).
[8] Brandt A., Brannick J., Kahl K., and Livshits I.: “Bootstrap AMG”, SIAM Journal on Scientific Computing, 2011, 33(2), 612-632.
[9] AMGS ライブラリ，http://hpcl.info.kogakuin.ac.jp/lab/software/amgs.
[10] Hestenes, M. R. and Stiefel, E.: “Methods of Conjugate Gradients for Solving Linear Systems”, Journal of Research of the National Bureau of Standards, Vol. 49 , No.6, pp.409-436 (1952) .
[11] Kesheng W.: “Thick-restart Lanczos Method for Large Symmetric Eigenvalue Problems”, SIAM J. Matrix Anal. Appl., Vol.22, No.2, pp.602-616 (2000).