Algebraic Multigrid for Eigenproblems in Industrial Applications of Big Data and Engineering

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Silvia Gries, geb. Ehrmann
aus
Düsseldorf

Bonn 2023

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter:	Prof. Dr. Marc Alexander Schweitzer
2. Gutachter:	Prof. Dr. Jochen Garcke

Tag der Promotion:	09.11.2023
Erscheinungsjahr:	2023

Abstract

More and more data are generated and gathered by computational simulations, network evaluations, and sensor usage in industrial applications and many other methods. To make the most out of the collected data, these should be examined in search of (hidden) patterns or any other predictive information. This analysis gives rise to sparse linear systems and the need to calculate the smallest eigensolutions of them.

To evaluate these data, matrices via Graph Laplacian or adjacency graphs are defined and the corresponding eigenproblems need to be solved. These matrices mainly have a sparse pattern. However, a few rows do not fit in this pattern. They are connected to more data points. This inhomogeneous pattern is difficult for most linear solvers. This also holds for classical Algebraic Multigrid (AMG), although it should generally be well-suited for these systems. It should be promising as it has excellent scaling properties and is only relying on algebraic information.

AMG, similar to other iterative linear solvers, is a two-phase algorithm. At first, a setup is constructed. This is then used for the solution. In this thesis, we are modifying both phases of the AMG solver in order to apply it to data science problems.

We are introducing a new approach for the setup of AMG that can especially handle the matrix inhomogeneities. Based on this, we will develop an approach to solve (modified) eigenproblems by exploiting the AMG hierarchy for problems as they arise in data analysis.

We demonstrate the applicability of these both methods with various examples. In the first instance, we take the Poisson problem to show the effect of algorithmic variations we implemented. The setup approach is additionally evaluated for petroleum reservoir simulations and different Graph Laplacians. We use the eigensolution calculation also for Graph Laplacians. However, we additionally improve the solution for very ill-conditioned problems from structural mechanics.

Zusammenfassung

Durch Computersimulationen, Auswertung von Netzwerken oder Sensordaten in industriellen Produktionen und vielen weiteren Optionen erzeugen wir immer mehr Daten. Nach Möglichkeit sollten diese Daten ausgewertet werden um Rückschlüsse auf (versteckte) Ähnlichkeiten oder andere weiterverwendbaren Informationen zu erhalten. Bei dieser Analyse ergeben sich große, dünn-besetzte lineare Gleichungssysteme und die Notwendigkeit die kleinsten Eigenlösungen zu bestimmen.

Um die Daten auszuwerten werden aus diesen Matrizen als Graph Laplaces oder Adjazenzgraphen berechnet. Diese werden dann benutzt um die zugehörigen linearen Gleichungssysteme oder Eigenwertprobleme zu lösen. Die entstanden Matrizen haben ein mehrheitlich dünnes Besetzungsmuster. Jedoch fallen einzelne Zeilen aus diesem Muster heraus, weil es einige Datenpunkte gibt, die mehr Verknüpfungen zu anderen Datenpunkten haben. Aus dieser inhomogenen Besetzungstruktur ergeben sich Schwierigkeiten für den linearen Löser. Dies gilt auch für Algebraische Mehrgitter (AMG), welches grundsätzlich als linearer Löser für diese Matrizen einsetzbar ist. AMG verspricht ein idealer linearer Löser zu sein, da dieser über gute Skalierungseigenschaften verfügt und rein algebraisch verwendbar ist.

Wie sehr viele iterative Löser ist auch AMG zweiphasig aufgebaut. Zunächst wird ein Setup konstruiert. Basierend auf diesem wird die Lösung dann iteriert. In der vorliegenden Arbeit werden wir beide Phasen modifizieren um eine bessere Anwendbarkeit für die Datenprobleme zu erreichen.

Zunächst führen wir eine neue Setup-Strategie für AMG ein um besser mit den Matrixinhomogenitäten umgehen zu können. Basierend darauf etablieren wir eine Lösungsphase, die es ermöglicht (generalisierte) Eigenprobleme zu lösen. Bei der Lösung werden wir die im Setup erzeugte AMG Hierarchie mehrfach wieder verwenden.

An Hand verschiedener Beispiele zeigen wir die Anwendbarkeit beider Methoden. Zunächst evaluieren wir diverse algorithmische Variationen am bekannten Poisson Problem. Die Setup-Strategie ist zusätzlich für Ölreservoirsimulationen und verschiedene Graph Laplace ausgewertet. Die Eigenwertberechnung wenden wir ebenfalls auf Graph Laplace an. Außerdem verbessern wir mit Eigenvektoren die Lösbarkeit von schlecht konditionierten Problemen aus der Strukturmechanik.

Contents

CHAPTER 1

Introduction

The continuously emerging need for analyzing huge amounts of data in the fields of Big Data, Machine Learning, and Artificial Intelligence in both research and industrial applications creates the requirement for more efficient analysis methods. This thesis contributes to this research area by improving and extending the applicability of existing methods.

Simultaneously with the growth of computational power, more and more data is generated in less time. These data should be analyzed to discover patterns and draw trends. In a first step, it is very common to reduce the dimension of the data. That means focusing on the core information of the data. This dimension reduction is possible with various methods. Principal Component Analysis (PCA) [74] applies a linear transformation to a lower-dimensional space to extract the core information. In contrast, diffusion maps [42, 106, 44] calculate non-linear embeddings to a lower-dimensional Euclidean space. A vast overview of further non-linear dimension reduction approaches is given in [87]. Another option to analyze the data is to search for patterns inside the data that have "similar properties", often called feature recognition. Spectral clustering [156, 110] is one useful family of algorithms to fulfill this task.

All of these methods have in common to use a matrix representation of the relation inside the data. Typically, Graph Laplacians are employed here [40]. This representation is then used to calculate a few smallest or largest eigensolutions, in some cases only the eigenvalues or the eigenvectors, or solutions of stationary linear systems. With these results, the machine learning algorithms are then proceeding and calculating a lower dimensional representation or feature recognition. As the background of the data originates from different application fields, we need generic methods to handle all the occurring eigenproblems. Hence, algorithms that work purely algebraically, i.e. matrix-based, are preferred.

A powerful algorithm to handle difficult and huge sparse linear systems is Algebraic

Multigrid (AMG) [23, 63, 137]. AMG has been developed as linear solver for linear systems that arise from the discretization of elliptic partial differential equations (PDEs) and works only with the matrix itself and exploits certain properties of it. Nevertheless, AMG is applicable to linear systems from many applications, e.g., for flow simulations [139] or petroleum reservoir simulations [59, 60]. Graph Laplacians from data analysis applications feature almost all those properties that AMG seeks to exploit. Graph Laplacians are only positive semi-definite [40], but they need to be positive definite to fulfill the requirements to apply AMG. However, by a rather small change, the Graph Laplacian can be turned positive definite and, thus, AMG becomes applicable without negative impacts on the approximated solution, as we can motivate. Thus, AMG would be a promising method, especially due to its efficiency.

Similar to other iterative linear solvers AMG is a two-phase linear solver - consisting of a setup phase and a solution phase. During the setup phase, a matrix hierarchy is constructed. In the solution phase this matrix hierarchy is applied to calculate the approximate solution. This approach has the advantage that different error components of a solution approximation are handled at various matrix hierarchy levels with appropriate resolutions [23, 148]. Additionally, it shows excellent scaling properties regarding parallelization [81, 162]. These properties make AMG a very considerable method to handle the increasing size of data in a reasonable amount of computational time. Furthermore, we can profit from further optimization options, such as the re-usage potential of a single setup of AMG.

However, classical AMG is hardly applicable in a straight-forward manner. First of all, we need to exploit the matrix hierarchy for the solution of an eigenproblem rather than classical linear systems. And, as Graph Laplacians only nearly fulfill AMG-suited properties, they pose certain challenges for creating an AMG setup. This originates from the growing mass of data along with an increasing data connectivity, besides the positive semi-definiteness of Graph Laplacians. By the first point, the representation matrices themselves are growing in size and decrease in sparsity. As the connectivity of the data grows the heterogeneity structure of the sparsity pattern changes. This pattern is no longer sparse as it initially was assumed by "classical" theory.

To enable the applicability of AMG for matrices that occur from various data applications, we have two starting points:

On the one hand, we need to improve the setup phase of AMG due to the presence of varying sparsity patterns. A few rows can extend the "classical" understanding of sparse.

On the other hand, we need to change the solution phase of AMG to directly combine the application of the matrix hierarchy with eigensolution calculations. Our contributions to both aspects, of course, can also be exploited by themselves in other application fields.

To summarize the objectives of this thesis, we have two question sets that we will answer:

How can we adapt the setup phase of AMG to handle growing heterogeneities in the sparsity pattern? This heterogeneity in the sparsity pattern occurs, for instance, with social media graphs depending on user activity, as they include strongly frequented nodes as well as isolated nodes [90]. But besides data analysis, it also is applicable for geomechanical petroleum reservoir simulations as stone and rock is a highly heterogeneous material [7].

We present a setup method for AMG that can especially handle such difficulties. For that purpose, we algebraize [48] a previously only geometrically applicable technique [163]. This setup of AMG includes various parameters to fit different connectivity of nodes on the constructed matrix hierarchy. This is combined with a localized ideal interpolation for transferring between the matrix hierarchy levels.

This takes us to the second question part. How can we exploit the AMG hierarchy in the eigensolution approximation? How are eigensolutions - eigenvalues and eigenvectors - related between various matrix hierarchy levels? Which already known eigensolution calculation algorithms can we use to extend them with AMG?

We will develop an approach for an eigenproblem solver based on AMG. While being based on established one-level eigensolver methods such as inverse iteration [6, 161] or Krylov-Schur method [69, 121, 143, 144, 82], the incorporation of an AMG hierarchy and necessary modifications to benefit from the AMG hierarchy will increase the applicability to bigger eigenproblems. Inverse iteration is the simplest eigensolver method to approximate the smallest eigensolution. To do so, this iterative method uses an inverse matrix and, thus, integration of an AMG hierarchy is a promising idea. The Krylov-Schur method is a more elaborate eigensolver that can calculate more smallest eigensolutions and is more promising for challenging problems. We will demonstrate the strength of both approaches in combination with using the AMG hierarchy with several practical examples from data science and beyond. And we will point towards further research directions.

Especially with answering the second questions part, we gain another application of the eigensolutions. It is possible to use (approximate) eigenvectors to improve the matrix hierarchy of AMG itself [152, 153, 151, 97]. Then, we can extend the application of AMG to more ill-conditioned linear systems as possible before. Very roughly spoken, the eigenvectors are an indicator for those variables that are "more" relevant in the adjacency graph and should, thus, have a representation on coarser matrix hierarchy levels.

Outline of this thesis

In Chapter 2, we start with a short overview of Algebraic Multigrid methods. We thereby focus on those parts of the wide range in AMG approaches and theory that are relevant for this thesis. We describe the solution phase and setup phase of AMG. In both parts, we introduce well-known methods and usage modes that will serve as reference in later evaluations. We end this chapter by a theoretic part. There we summarize important results of the convergence theory and define necessary notations.

Chapter 3 focuses on the introduction of Algebraic Multiscale AMG (AM-AMG). AM-AMG is an extension of aggregative coarsening that includes a separation of three different function types for the variables in the aggregation process. This is used to construct overlapping aggregates, which was impossible before in a purely algebraic manner. Furthermore, we use these function types to construct a locally ideal interpolation per aggregate. Both aspects make AM-AMG a perfectly suited setup approach for data science problems with heterogeneous stencil sizes.

The introduction of the three different function types, namely vertices, edges and interiors, results in a few additional parameters to fine tune the aggregation process. We explain these effects in Section 3.5. Moreover, we prove that AM-AMG fulfills all requirements in the convergence theory of AMG. Furthermore, we use AM-AMG in different application fields to compare it to other setup approaches of AMG.

Chapter 4 is about the application of AMG for eigensolution approximation (EP-AMG). We present three algorithms for eigensolution approximation that utilize our AMG hierarchy. We are starting with an initial guess calculation that exploits the full effect of an existing matrix hierarchy. Afterwards, we explain two eigensolution algorithms, namely inverse iteration and Krylov-Schur method, and how AMG technology is exploit. At the end of this chapter, we evaluate our algorithms for various applications.

Our thesis is finalized by Chapter 5. Here we summarize our results and provide an outlook on further research questions that have arisen during this thesis.

Acknowledgment

I would like to thank all persons who supported me to complete this thesis. First, my gratitude goes to my supervisor Prof. Dr. Marc Alexander Schweitzer. He gave me the opportunity to write this thesis in his working group and supported me during the whole process. Additionally, I would like to thank the whole SAMG-team at the Fraunhofer Institute SCAI. They introduced me to AMG and it's application field. Furthermore, they gave me the chance to learn a lot - also next to my PhD research. Finally, I would like to thank Dr. Bram Metsch for proof-reading this thesis. A thank you also goes to Clelia Albrecht for supporting and discussing all relevant topics during working on this PhD project in our regular PhD tea time.

Additionally, I thank Dr. Daniel Oeltz for introduction and discussing about bitcoin Graphs and Dr. Rodrigo Iza-Teran and Christian Gscheidle for introduction and discussing about Big Data and Machine Learning.

I am grateful to Prof. Dr. Jochen Garcke for being my second advisor. I would also like to thank Prof. Dr. Sergio Conti and Junior Prof. Dr. Annika Thiel for participating in the committee of the defense of this thesis.

Last but not least, I express my gratitude to my family and friends for supporting me through out this journey. A special thank you goes to my parents Kirsten and Wolfgang who gave me endless love and encourage me to follow my dreams. Moreover, I thank my sister Svenja for always having an open ear. At last, I thank my financé Sebastian for his patience, ongoing support and his uplifting words.

CHAPTER 2

AMG - Overview of Algebraic Multigrid Methods

The developments in the scope of this thesis rely on the well-known Algebraic Multigrid (AMG) method. To embed the newly developed AM-AMG approach from Chapter 3 and the application of AMG as an eigenvalue solver in Chapter 4 in the AMG theory, we give a short overview of the necessary aspects in this chapter.

Very shortly described, the AMG solution process is separated into a setup phase and a solution phase. The simplest version for the solution phase of AMG is a two-level scheme. By successive application of this two-level scheme, a multi-level scheme is established. In the following, we give a short introduction to the solution phase in Section 2.2 and the setup phase in Section 2.3. Furthermore, we provide some essential aspects such as requirements for convergence of an AMG method in Section 2.4.3 or the smoothing property in Section 2.4.2.

Throughout this chapter, we consider a linear system

$$Au = b \tag{2.1}$$

with a sparse, symmetric, positive definite M-matrix [118] $A = \left(a_{ij}\right)_{i,j=1,\dots,n} \in \mathbb{R}^{n\times n}$, a right-hand side vector $b = (b_i)_{i=1,\dots,n} \in \mathbb{R}^n$ and the solution vector $u = (u_i)_{i=1,\dots,n} \in \mathbb{R}^n$. Such linear systems typically arise from the discretization of elliptic partial differential equations (PDEs).

2.1 Motivation of AMG

Algebraic Multigrid (AMG) is an iterative solution approach for sparse linear systems as in Equation (2.1), which originally result from the discretization of elliptic PDEs. The origin for this linear solver is based on Geometric Multigrid (GMG) that constructs a

hierarchy of (geometric) grids with successively smaller grid resolutions [148, 62]. As an extension and generalization, AMG has been developed with a construction of matrix hierarchy levels in a purely algebraic manner. For a detailed description about initial theory and developments of AMG, see [23, 22, 26, 124, 50, 63]. Information about the applicability in an industrial context is given by [138, 139]. Other methods than multigrid for solving sparse linear systems are explained and described in [125].

Along with the extension of application fields for Algebraic Multigrid methods, there also evolved various extensions of specialized techniques for subproblem classes, e.g., Lean AMG [93], aggregation AMG [111, 153], or smoothed aggregation [153, 152]. A variety of AMG methods are slightly differently implemented in various software libraries, e.g., BoomerAMG [68, 162], BootstrapAMG [24, 25], Trilinos [70], SAMG [136, 139, 119], AMG-approaches in PETSc [9, 10, 11] . The wide range of various software libraries is based on different preferred AMG strategies or differing programming languages.

In this work, all described new AMG-technologies are implemented in the software package SAMG that has been developed by Fraunhofer SCAI [136] and is mainly written in Fortran.

The mathematical motivation for using a level hierarchy is similar for GMG and AMG with all its varying application fields. The error of the solution of a linear system as in Equation (2.1) has a wide error frequency spectrum. It is well-known that one-level methods, like Jacobi or Gauss-Seidel iteration, reduce high-frequency error components. Thus, they smooth the error and are called smoothing operators or smoothers $\mathscr{S}$. After applying such a smoother, the low-frequency error components remain. The aim of introducing a coarser problem is to create a framework in which the low-frequency error components can be solved efficiently. We will not go into further details here and refer to the literature [23, 148].

In the following, a brief summary of AMG is given with a focus on those theoretical aspects that are relevant during this thesis. This summary, including the notation, is mainly based on Stüben [137, 138].

Due to the original idea of constructing coarser problems induced by the discretization grids, some literature uses h and H as the subindicies of the matrices to refer to the fine and coarse grid refinement. As AMG works fully algebraically, we use the notation indicated by coarse c and fine f level in this thesis.

2.2 Solution Phase of AMG

2.2.1 Two-Level Scheme

The simplest solution phase of AMG is a two-level scheme. This employs a coarse-level matrix $A_c \in \mathbb{R}^{n_c \times n_c}$ based on the original matrix $A = A_f \in \mathbb{R}^{n_f \times n_f}$, called fine-level

matrix. The construction of the coarse-level matrix is performed in the setup phase of AMG that we will describe in Section 2.3. During the setup-phase of AMG the transfer operations between the fine and coarse-level matrices, namely the interpolation $I_c^f \in \mathbb{R}^{n_f \times n_c}$ and the restriction $I_f^c \in \mathbb{R}^{n_c \times n_f}$ operators, and the coarse-level operator A_c are calculated.

Exemplarily, a few coarsening and interpolation approaches are explained in more details in Section 2.3 about the setup phase of AMG.

To iteratively calculate an approximate solution w^f of the linear system (2.1), a defect correction e^c is calculated on the coarse level. For this purpose, the linear system

$$A_c e^c = I_f^c \text{res}_{\text{old}}^f = I_f^c \left(b^f - A_f w_{\text{old}}^f \right). \tag{2.2}$$

on the coarse level has to be solved. Afterwards the solution approximation w^f is updated by the interpolated solution e^c, i.e.

$$w_{\text{new}}^f = w_{\text{old}}^f + I_c^f e^c. \tag{2.3}$$

This coarse-level-based defect correction is combined with a smoothing process $\mathscr{S}$ on the fine level. This has the effect that high-frequency error components are handled on the fine level and low-frequency components on the coarse level, as outlined in the previous section.

All previously mentioned components of the basic algorithm for AMG as a two-level scheme are summarized in Algorithm 2.1. As for other iterative methods, a stopping or convergence criterion $\mathscr{C}$ is a requested residual reduction or a maximum number of iteration steps.

By a recursive strategy for solving the coarse-level equation in Line 5 in Algorithm 2.1, the two-level scheme is extended to a multi-level approach, see Section 2.2.2. Furthermore, it is possible to use AMG as a preconditioner in different Krylov methods. This will be described in Section 2.2.3.

2.2.2 Multilevel Scheme

Algorithm 2.1 describes a two-level scheme for AMG. For the solution of the coarse-level equation in Line 5, the two-level idea can be applied recursively. By this, we iterate over all coarser constructed matrix hierarchy levels. See Section 2.3 for more details regarding the matrix hierarchy construction. The multi-level solution phase is described in Algorithm 2.2.

The constructed matrix hierarchy levels are notated as $A = A_1 = A_f, \ldots, A_{lev}$ where lev denotes the number of constructed matrix hierarchy levels. The number of constructed matrix hierarchy levels depends on various user-defined AMG parameters, see Section

Algorithm 2.1: Solution phase of Algebraic Multigrid as a Two-Level Scheme

Input: A_f: fine-level matrix
b: right-hand side vector
u_0: start vector
$\mathscr{C}(u_i)$: stopping/convergence criterion
depending on current approximation u_i

Data: $\mathscr{S}$: smoothing operator
I_c^f: interpolation operator
I_f^c: restriction operator
A_c: coarse-level matrix
(I_c^f, I_f^c, A_c calculated in setup phase of AMG, see Section 2.3)

1 **while** *not* $\mathscr{C}(u_i)$ **do**
2 apply smoothing: $\widetilde{u}_i = \mathscr{S} u_{i-1}$
3 calculate residual: $\text{res}_i^f = b^f - A_f \widetilde{u}_i$
4 restrict residual: $\text{res}_i^c = I_f^c \text{res}_i^f$
5 solve coarse-level equation: $A_c e_i^c = \text{res}_i^c$ with solution e_i^c
6 interpolate coarse-level correction: $e_i^f = I_c^f e_i^c$
7 adding the correction: $\widehat{u}_i = \widetilde{u}_i + e_i^f$
8 apply smoothing: $u_i = \mathscr{S} \widehat{u}_i$
9 increase loop variable i

2.3. Furthermore, after the setup phase we have interpolation I_{k+1}^k and restriction I_k^{k+1} operators between two constructed matrix hierarchy levels k and $k+1$ for $k = 1, \cdots, (lev-1)$. Additionally, the smoothing process $\mathscr{S}_k$ is extended to be applied on every level to smooth the high frequency error components on this level. Similar to the two-level scheme in Section 2.2.1.

Figure 2.1 illustrates the typically used structure of the solution process, named V-cycle. We will not consider other forms of cycles as W- or F-cycle [148] here. These cycle-forms differ from the standard V-cycle by applying more coarse-level corrections per created level. For the applications we present in this work, the V-cycle is sufficient enough. The other cycles would work analogously with AM-AMG and EP-AMG.

2.2.3 AMG as a Preconditioner

Over the last decades, it has been established to use AMG not as a standalone solver, but rather as a preconditioner in combination with accelerators such as Conjugate Gradient Method (CG) [5, 125], Generalized Minimal Residual Method (GMRES) [125, 127] or

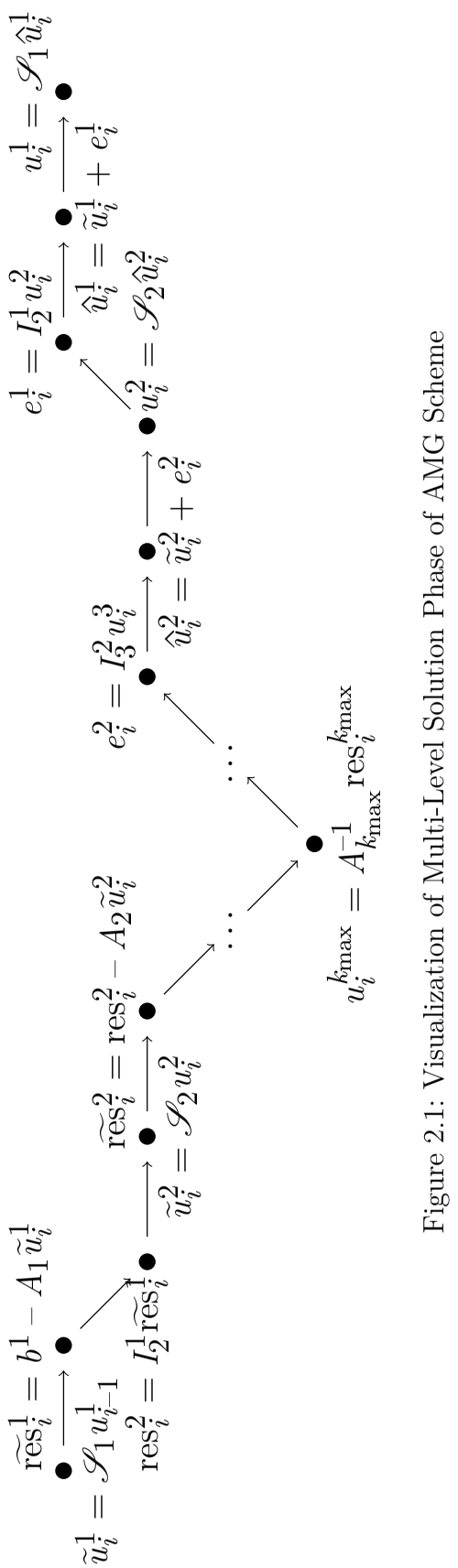


Figure 2.1: Visualization of Multi-Level Solution Phase of AMG Scheme

Biconjugate Gradient Stabilized Method (Bi-CGSTAB) [150]. The straight-forward way is to use the full AMG solution cycle, in combination with a previous setup phase, as a preconditioner [115].

Another possibility is to use an accelerator on every constructed matrix hierarchy level [116, 113], also called K-cycle. This provides additional robustness in certain ill-conditioned applications. These are not relevant for this thesis and the test cases that we will be concerned with.

Due to the fact that, in contrast to typically one-level preconditioners, e.g. ILU, AMG efficiently handles all error components, it provides an efficient preconditioner for elliptic problems.

Algorithm 2.2: Solution phase of Algebraic Multigrid as a Multi-Level Scheme AMG (A,b,u_0)

Input: $A = A_1$: fine-level matrix
b: right-hand side vector
u_0: start vector
$k_{\max}$: number of created levels
$\mathscr{C}\left(u_i^1\right)$: stopping/convergence criterion
depending on current approximation u_i^1

Data: each for $k = 1, \ldots, k_{\max} - 1$
$\mathscr{S}_k$: smoothing operator on level k
I_{k+1}^k: interpolation operator from level $k+1$ to k
I_k^{k+1}: restriction operator from level k to $k+1$
A_{k+1}: Galerkin operator on level $k+1$
(I_{k+1}^k, I_k^{k+1}, A_{k+1} calculated in setup phase of AMG, see Section 2.3)

1 **while** *not* $\mathscr{C}\left(u_i^1\right)$ **do**
2 apply smoothing: $\widetilde{u}_i^k = \mathscr{S}_k u_{i-1}^k$
3 calculate residual: $\mathrm{res}_i^k = b^k - A_k \widetilde{u}_i^k$
4 restrict residual: $\mathrm{res}_i^{k+1} = I_k^{k+1} \mathrm{res}_i^k$
5 solve coarse-level equation:
6 **if** $k + 1 = k_{max}$ **then**
7 direct solver: $e_i^{k+1} = A_{k+1}^{-1} \mathrm{res}_i^{k+1}$
8 **else**
9 init e_i^{k+1}
10 AMG(A_{k+1},res_i^{k+1}, e_i^{k+1})
11 interpolate coarse-level correction: $e_i^k = I_{k+1}^k e_i^{k+1}$
12 adding the correction: $\widehat{u}_i^k = \widetilde{u}_i^k + e_i^k$
13 apply smoothing: $u_i^k = \mathscr{S}_k \widehat{u}_i^k$
14 increase loop variable i

2.3 Setup Phase of AMG

During the setup phase of AMG, the matrix hierarchy for a given matrix A is constructed. There exists a wide range of applications for AMG technology. Hence, the linear systems to be solved provide various requirements. To handle these, different approaches for construction a matrix hierarchy during the setup phase exists. The setup phase of AMG itself is divided into three parts:

- Coarsening (or C/F-splitting)
  During the coarsening process, a set of variables is defined that induces the coarse-level variables C.

- Construction of the interpolation I_c^f
  During the interpolation setup, the interpolation operator I_c^f for transferring between coarse- and fine-level variables is constructed. Additionally, the restriction operator I_f^c as reverse operation, i.e. transferring between fine- and coarse-level variables, is constructed.

- Calculation of the coarse-level operator A_c
  Based on the interpolation operator I_c^f and the corresponding restriction operator I_f^c, the coarse-level operator A_c is calculated as Galerkin product of interpolation and restriction.

In the wide range of possible setup strategies we focus on Ruge-Stüben coarsening with direct or standard interpolation [137], aggressive coarsening (as a well-established variant of Ruge-Stüben) with indirect/multi-pass interpolation [137], and aggregation coarsening with piecewise-constant interpolation [111, 153, 21]. With them, we will compare our newly developed coarsening method AM-AMG with the locally ideal interpolation in Chapter 3. Furthermore, there exists many other coarsening and interpolation strategies especially suited for specific request of the application field as, e.g., PMIS/HMIS for high core numbers [141] or energy-based coarsening for further improving interpolation/restriction operators [29, 32].

The coarse-level matrix A_c is a coarse-level approximation of the fine-level matrix A_f via its construction as Galerkin product $A_c := I_f^c A_f I_c^f$, with the interpolation I_c^f and the restriction I_f^c. By recursively applying a coarsening and a calculation of interpolation/restriction operators, a full matrix hierarchy of Galerkin operators $A_2, \ldots, A_{lev}$ can be constructed based on the fine-level matrix $A_f = A_1$.

Normally, the construction of the matrix hierarchy ends when the coarsest matrix A_{lev} has a size that is suited for a (sparse) direct solver. The construction process can, for instance, be stopped after a fixed number of constructed levels is reached, the coarsest matrix is small enough for a direct solver, or a given density threshold is reached.

Throughout this thesis, we consider the restriction to be the transpose of the interpolation, i.e., $I_f^c = \left(I_c^f\right)^T$. This is a common procedure, as this ensures a symmetric coarse level operator A_c and, hence, a symmetric cycling in the solution phase, see Section 2.2. However, there are other options available, as e.g., a modified restriction [96, 95] that locally modifies the restriction to ensure an optimized restriction for a small surrounding of coarse-level variables.

The so-called accuracy of interpolation gives a formal relation between convergence of AMG and the constructed setup (coarsening and interpolation) of AMG. When this condition is fulfilled the full AMG cycle converges. We shortly embed this in the mathematical framework in Section 2.4.3 as Theorem 3.1.

Before describing how some coarsening approaches work in the Sections 2.3.2 to 2.3.4, we introduce some necessary notations.

2.3.1 Relevant Notations for Coarsening Description

The set C denotes all variables to be transferred to the coarse level. The set F is the complementary set of variables that remain on the fine level. By this setting, the full set of variables Ω^f on the fine level is the disjunct union of C and F. Thus, the coarsening is also called C/F-splitting.

For the sake of simplicity, we formulate the following notations and definitions without explicitly mentioning the level index with the matrix A and all relevant sets.

Two variables i and j are called coupled if $a_{ij} \neq 0$. We define the neighborhood N_i of a variable i as all variables j that have a coupling to the variable i, i.e.,

$$N_i := \left\{ j \in \Omega | i \neq j, a_{ij} \neq 0 \right\}, \tag{2.4}$$

and the complementary set N_i^0 of variables j that are not included in the neighborhood of i, i.e.,

$$N_i^0 := \left\{ j \in \Omega | i \neq j, a_{ij} = 0 \right\} = \Omega \setminus \left(N_i \cup \{i\} \right). \tag{2.5}$$

As a criterion for the coarsening, the so-called coupling strength between two variables is used. This measure indicates how strong or weak the coupling a_{ij} between two variables i and j is. The greater the coupling strength between two variables is, the better the smoothing of the error in the direction of these variables is [137]. Thus, for a good convergence behavior, strongly connected variables should ideally span the coarse-level matrix. A coupling between i and j is considered strong when the condition

$$-a_{ij} \geq \epsilon_{\text{str}} \max_{k \neq i} |a_{ik}| \tag{2.6}$$

with fixed $0 < \epsilon_{\text{str}} < 1$ is fulfilled. In practical applications, $\epsilon_{\text{str}} = 0.25$ has been established as a reasonable value to use. The set of strong couplings S_i is then defined as

$$S_i := \{j \in N_i | i \text{ strongly coupled to } j\}\,. \tag{2.7}$$

The couplings that are not strong are called weak couplings. The set of weak couplings is notated as $W_i = N_i \setminus S_i$. The set of coarse- or fine-level variables of i are all coarse- or fine-level variables that are strongly connected, i.e., $C_i = C \cap S_i$ and $F_i = F \cap S_i$, respectively.

The concept of strong couplings can be extended to longer paths of couplings. Two variables i and j have a strong coupling along a path of length l, if there exists a sequence of strongly coupled variables $i = i_0, i_1, \cdots, j = i_l$, i.e., $i_{k+1} \in S_{i_k}$ for $k = 0, \cdots, l-1$. Additionally, this can be extended to the variable i strongly connected to variable j w.r.t (p, l): there exists p sequences such that i has a strong coupling to j along a path of the length $\leq l$. The general set of strong couplings $S_i^{(p,l)}$ for a variable i is then defined as

$$S_i^{(p,l)} := \{j \in \Omega | i \text{ strongly coupled to } j \text{ w.r.t. } (p, l)\}\,. \tag{2.8}$$

For the set of strong couplings, it yields $S_i = S_i^{(1,1)}$.

The entries of the interpolation operator $I_c^f = (w_{ik})_{\substack{i=1,\ldots,n_f \\ k=1,\ldots,n_c}} \in \mathbb{R}^{n_f \times n_c}$ are named interpolation weights w_{ik}.

2.3.2 Ruge-Stüben Coarsening

The Ruge-Stüben coarsening [137] sets up a splitting into coarse-level variables C and fine-level variables F with the following two properties:

(RS1) The coarse-level variables C form a maximal independent set of variables in terms of strong connectivity, i.e., two coarse-level variables have no strong coupling.

(RS2) For every fine-level variable $i \in F$ and each strongly connected variable $j \in S_i$, the variable j is either a coarse-level variable, i.e., $j \in C$, or strongly coupled to another coarse-level variable, i.e., $S_j \cap C \neq \emptyset$.

In the simplest version, the Ruge-Stüben coarsening is combined with a direct interpolation to calculate the coarse-level correction e_i^f on the fine level. With the direct interpolation, the fine-level variables are only interpolated along the directly strongly connected coarse grid points. That is, the interpolation weights w_{ij} are given as

$$w_{ij} = -\alpha_i \frac{a_{ij}}{a_{ii}} \tag{2.9}$$

for each fine-level variable $i \in F$ and each strongly coupled coarse-level variable $j \in C_i$ to the variable i. The factor $\alpha_i = \frac{\sum_{k \in W_i} a_{ik}}{\sum_{k \in C_i} a_{ik}}$ that takes all weak couplings into account is applied to ensure that a constant vector on the coarse level is exactly interpolated to a constant vector on the fine level. This is a pre-requisite for robust convergence.

Commonly, Ruge-Stüben coarsening is used in combination with standard interpolation [137]. Standard interpolation is an extension of the direct interpolation that includes the interpolation of fine-level variables by all strongly coupled coarse-level variables.

In [137] it is proven that Ruge-Stüben coarsening with direct or standard interpolation fulfills the accuracy of interpolation (Theorem 2.3 in Section 2.4.3). This means that the overall constructed multigrid cycle by Ruge-Stüben coarsening and direct or standard interpolation approach converges.

Asides the direct or standard interpolation formulation, there are further ones, such as (modified) classical interpolation [68] or extended interpolation [140]. These further improve the accuracy of interpolation, but follow the same principle as direct or standard interpolation. But they are not relevant in the ongoing work of this thesis.

2.3.3 Aggressive Coarsening

In some cases it might be necessary that the coarse level is much smaller than the fine level, e.g., due to a higher density of the fine-level matrix A_f or memory restrictions or performance reasons.

The splitting described in the previous Section 2.3.2 can also be applied to a more general set $S_i^{(p,l)}$ of strong couplings, see Equation (2.8).

In [137], two types of aggressive coarsening, mainly $S_i^{(1,2)}$ and $S_i^{(2,2)}$, are referenced. By using more general sets of strong couplings, it may happen that a fine-level variable has no direct connection to a coarse-level variable. Such variables are then interpolated from their strongly coupled fine-level variables, called indirect interpolation or multi-pass interpolation, cf. Section 4.3 and 7.2.2 in [137].

In an analogous way as for Ruge-Stüben coarsening, the fulfillment of the accuracy of interpolation (Theorem 2.3 in Section 2.4.3) is motivated in [137]. This ensures convergence of the complete multigrid cycle under certain assumptions that are given in our cases.

The computation of the extended strong couplings sets $S_i^{(q,l)}$ is quite expensive. Thus, for practical reasons, roughly spoken, the Ruge-Stüben coarsening is simply applied twice.

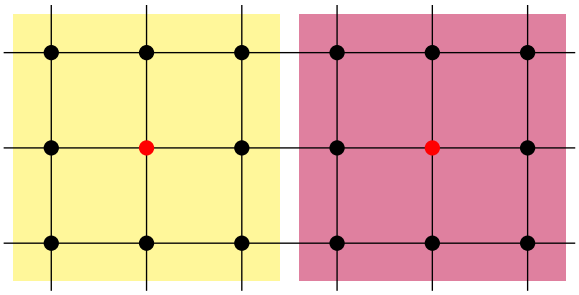


Figure 2.2: Schematic construction of aggregates (yellow and purple) with aggregative coarsening of size nine. This considers a Poisson problem discretized by a five-point stencil on a uniformly structured grid. The red points mark the coarse-level variables. Ideally, these variables are centrally located inside the aggregate.

2.3.4 Aggregative Coarsening

In contrast to both previously described coarsening strategies that firmly distinguish between coarse- and fine-level variables, in aggregation-type coarsening strategies disjoint aggregates of the fine-level variables are constructed [18, 35]. For every aggregate one variable i is set as coarse-level variable. Thus, we notate these aggregates as G_i with $i \in C$ and $G_i \cap G_j = \emptyset$ for $i, j \in C$ and $i \neq j$.

These aggregates only contain variables $j \in G_i$ that are strongly connected to at least one other variable, i.e., $S_j \cap G_i \neq \emptyset$. Each of these aggregates is much smaller than the fine level set of variables, i.e., $|G_i| \ll |\Omega^f|$. This provides a rather lean and well-parallelizable way of coarsening at the expense of less robustness. In Figure 2.2, we show two disjoint aggregates on an uniformly structured grid. For each aggregate, one variable is set as coarse-level variable and marked by a red point. Ideally, the coarse level variable is a central variable of the aggregate.

The aggregate construction is also based on the concept of strongly coupled variables. By this, the smoothing of the low-frequency errors on the coarse level is ensured. In [152, 153], a slightly different definition for the set of strongly coupled variables to a variable i is given, such that the variable i itself is included and the strong connectivity definition is symmetric. The symmetry is relevant, as for this coarsening approach one fine-level variable only relates to one coarse-level variable. Thus, a strong coupling should be strong in both directions.

As each fine-level variable only relates to one coarse-level variable, only piecewise-constant interpolation inside one aggregate is possible. For piecewise-constant interpolation, each fine-level variable of one aggregate takes the same value as the coarse-level variable of this aggregate. The interpolation weights are given as

$$w_{ij} = 1 \tag{2.10}$$

for each fine-level variable $i \in F$ and $j \in C$ with $i \in G_j$, all other are set to 0. This is quite unstable and the convergence would depend on the aggregate sizes. Due to this, the convergence rate per default is much worse than for Ruge-Stüben or aggressive coarsening.

To achieve a robust convergence for aggregation-type coarsening strategies, some adaptions are possible. In [34, 152, 153, 151] the problems of the piecewise-constant interpolation are solved by smoothing of the prolongation operator. This naturally includes multiple coarse-level information in the interpolation of one fine-level variable. However, at the expense of loosing the simplicity of the aggregation approach.

Another correction of the piecewise-constant interpolation by smoothing the interpolated error is feasible, which is often named as smoothed correction or V*-cycle [21, 138]. This targets in the same direction as the smoothed aggregation does. However, it is limited to the solution vector instead of considering the entire interpolation operator.

A further possibility is to use a K-cycle that adapts the coarse-level error correction on each matrix hierarchy level by using an additional Krylov solver [116, 111, 113, 112]. This addresses the limitations of the aggregation process by combining the AMG-cycle with further solver methods.

2.3.5 System-AMG Approaches

In various simulations, AMG is not only applied to linear systems based on a scalar PDE. In such systems, not only one (physical) unknown is considered, but n_u unknowns with a coupled system of PDEs. For an easier description, we assume the linear system to be ordered by the n_u different (physical) unknowns [1], i.e. the linear system (2.1) has the form

$$\begin{pmatrix} A_{[1,1]} & A_{[1,2]} & \cdots & A_{[1,n_u]} \\ A_{[2,1]} & A_{[2,2]} & & \vdots \\ \vdots & & \ddots & \\ A_{[n_u,1]} & \cdots & & A_{[n_u,n_u]} \end{pmatrix} \begin{pmatrix} u_{[1]} \\ u_{[2]} \\ \vdots \\ u_{[n_u]} \end{pmatrix} = \begin{pmatrix} f_{[1]} \\ f_{[2]} \\ \vdots \\ f_{[n_u]} \end{pmatrix}.$$

For the unknowns $i = 1, \ldots, n_u$ $u_{[i]}$ and $f_{[i]}$ are the corresponding slices of the solution vector and the right-hand side vector. The submatrices $A_{[i,j]}$ for $i, j = 1, \ldots, n_u$ include the couplings between the variables that are affiliated to the (physical) unknowns i and j, e.g., pressure or velocity in x-, y- or z-direction.

A straight-forward way to handle this linear system is by the unknown-based approach [41]. The coarsening and interpolation are applied independently to each (physical) unknown. That is only the submatrices $A_{[i,i]}$ are considered during the coarsening and

[1] Every system can be sorted trivially by unknowns. This is only done to enhance readability.

the construction of the interpolation operators of AMG. This coincides with using

$$\widetilde{A} = \begin{pmatrix} A_{[1,1]} & 0 & \cdots \\ 0 & \ddots & \\ 0 & & A_{[n_u,n_u]} \end{pmatrix}$$

as matrix for the setup phase of AMG, i.e., coarsening and interpolation/restriction are constructed on the basis of $\widetilde{A}$. For the calculation of the coarse-level matrix as Galerkin operator then there exist two variations: the block Galerkin $A_c = I_f^c \widetilde{A} I_c^f$ or the full Galerkin $A_c = I_f^c A I_c^f$ [41]. The latter one is typically used, as this includes the couplings between different unknowns in the coarse-level matrix.

Another approach to handle system problems is to use the so-called point-based approach. In this approach, a coarsening is constructed on a point-based adjacency graph. For the applications that we use in this work, this AMG approach is not relevant. For the system examples that we are concerned with in this work, the unknown-based approach is sufficient. For further details, however, see the description in [41] and references therein.

2.4 Theoretical Basics and Notation of AMG

Under certain assumptions, it has been proven that AMG converges to a solution of the linear system (2.1) [23, 137]. We choose the mathematical integration in the theoretical framework given by Ruge and Stüben, inter alia in [124]. Before we give a brief overview of the accuracy of interpolation in Section 2.4.3, which is a criterion for the quality of the interpolation to ensure convergence of AMG, we shortly introduce some further necessary notations in the following.

2.4.1 Notations for AMG

The interpolation weights $I_c^f = (w_{ik})_{\substack{i=1,\dots,n_f \\ k=1,\dots,n_c}} \in \mathbb{R}^{n_f \times n_c}$ specify the interpolation calculation from a given coarse-level vector $v^c = (v_i^c)_{i=1,\dots,n_c} \in \mathbb{R}^{n_c}$ as follows,

$$v_i^f = \left(I_c^f v^c\right)_i = \begin{cases} v_i^c & \text{if } i \in C \\ \sum\limits_k w_{ik} v_k^c & \text{if } i \in F \end{cases} = \begin{pmatrix} I_{CC} \\ I_{FC} \end{pmatrix} = \begin{pmatrix} \mathbb{1}_C \\ I_{FC} \end{pmatrix} \tag{2.11}$$

with $v^f = \left(v_i^f\right)_{i=1,\dots,n_f} \in \mathbb{R}^{n_f}$.

For a given splitting into coarse-level C and fine-level F variables, we can implicitly reorder [2] the original fine-level matrix A_f and formulate it in terms of submatrices of

[2] This is only necessary for readability.

the following form

$$A = A_f = \begin{pmatrix} A_{[F,F]} & A_{[F,C]} \\ A_{[C,F]} & A_{[C,C]} \end{pmatrix}, \tag{2.12}$$

where $A_{[\cdot,\cdot]}$ are the submatrices that include the couplings between fine-level variables F and/or coarse-level variables C. Furthermore, we define $D_{[\cdot,\cdot]} := \operatorname{diag}\left(A_{[\cdot,\cdot]}\right)$ as only the diagonal of the submatrices $A_{[\cdot,\cdot]}$ and $D_f := \operatorname{diag}\left(A_f\right)$ as the diagonal of the fine-level matrix A_f.

For the explanations of the smoothing property, in Section 2.4.2, and the accuracy of interpolation, in Section 2.4.3, we define the three scalar products

$$\begin{aligned} \langle u, v\rangle_1 &:= \langle A_f u, v\rangle, \\ \langle u, v\rangle_2 &:= \langle D_f^{-1} A_f u, A_f v\rangle, \\ \langle u_F, v_F\rangle_{0,F} &:= \langle D_{[F,F]} u_F, v_F\rangle \end{aligned}$$

and the induced norms $\|.\|_1$, $\|.\|_2$, $\|.\|_{0,F}$ by this. Here, $\langle\cdot,\cdot\rangle$ is the Euclidean scalar product, $u, v \in \mathbb{R}^{n_f}$, and $u_F, v_F \in \mathbb{R}^{n_F}$ with $u_F = u|_F$ and $v_F = v|_F$.

After finishing the AMG setup, the *matrix complexity* can be computed. It relates the number of non-zero entries in the whole matrix hierarchy to the number of non-zero entries of the original matrix, i.e., written as a formula

$$\textit{matrix complexity} := \frac{\sum_{i=1}^{lev} |A_i|}{|A_1|}$$

where $|B|$ is the number of non-zero entries of a matrix B. With the matrix complexity, an indicator for the memory consumption of the matrix hierarchy is provided.

2.4.2 Smoothing Property

2.1 Definition ([137], Section 3.2)
An operator $\mathscr{S}$ satisfies the smoothing property with respect to a symmetric and positive definite matrix $A \in \mathbb{R}^{n\times n}$, if

$$\|\mathscr{S} e\|_1^2 \leq \|e\|_1^2 - \sigma\|e\|_2^2 \tag{2.13}$$

holds with σ being independent of $e \in \mathbb{R}^n$.

The smoothing property essentially ensures that high-frequent error components are efficiently reduced by the smoother. The error $e_i^k = u_*^k - u_i^k$ of the current solution approximation $u_i^k$to the exact solution u_*^k on matrix hierarchy level k is smoothed by

applying the smoothing operator $\mathscr{S}$ before and after the coarse-level correction, see Lines 2 and 13 in Algorithm 2.2.

In Section A.3.2 in [137], it is shown that Gauss-Seidel and ω-Jacobi relaxation fulfill the smoothing property. We mainly use Gauss-Seidel smoother in our applications later on.

2.4.3 Accuracy of Interpolation

To ensure convergence for a two-level multigrid cycle, the interpolation should be reliable in some sense. The following theorems give properties for the interpolation and the underlying C/F-splitting. Essentially, the error correction of the coarse-level correction operator $\mathcal{T} := \mathbb{1}_f - I_c^f A_c^{-1} I_f^c$ should give a well-reducible error for post-smoothing afterwards. Especially, low-frequent errors should be reduced by the error correction.

The overview on convergence estimates of the two-level AMG method that is given in this section is based on the results presented in [124, 137].

2.2 Theorem ([137], Theorem A.4.1)
Let A be a symmetric positive definite matrix and $\mathscr{S}$ be the smoothing operator, which fulfills the smoothing property (2.13)*. Furthermore, we assume the C/F-splitting and the interpolation to fulfill*

$$\|\mathcal{T}e\|_1^2 \leq \tau \|\mathcal{T}e\|_2^2 \tag{2.14}$$

with $\tau > 0$ independent of $e \in \mathbb{R}^f$. Then yields $\tau \geq \sigma$ and $\|\mathscr{S}\mathcal{T}\|_1 \leq \sqrt{1 - \frac{\sigma}{\tau}}$.

This theorem can directly be proven by combing the smoothing property (2.13) and (2.14).

The following theorem gives an easier condition. It only requires a condition for the constructed C/F-splitting and the calculated interpolation such that Equation (2.14) is fulfilled.

2.3 Theorem ([137], Theorem A.4.2)
If the C/F-splitting and interpolation I_{FC} are such that for all $e \in \mathbb{R}^f$,

$$\| e_F - I_{FC} e_C \|_{0,F}^2 \leq \tau \| e \|_1^2 \tag{2.15}$$

with τ being independent of e, then Equation (2.14) *is satisfied.*

The interpolation formula in (2.11) fulfills Equation (2.15) and, thus, Theorem 2.2.

2.4 Remark
To be exact, there is a difference between using pre- or post-smoothing in combination with coarse-grid correction, cf. Section A.5 in [137]. But we aim at asymptotic convergence

behavior of a two-level multigrid and in almost all cases pre- and post-smoothing is used. Thus, the conclusive results of convergence theory for pre- or post-smoothing are transferable respectively.

CHAPTER 3

AM-AMG - New AMG Setup Suitable for Varying Matrix Patterns

In the following chapter, we present a newly developed setup method for AMG: Aggregative Multiscale AMG (AM-AMG). With that, we present an improved aggregation method, see Section 2.3.4, that is inspired by a geometric solver for reservoir modeling (cf. [163] and the outline in Section 3.1).

The definition of coarse- and fine-level variables in the setup phase of AMG relies on the ideas behind aggregation coarsening, i.e., the coarsening aims at constructing compactly shaped aggregates. In contrast to aggregation coarsening, we introduce specific function types for the variables inside the aggregate. This allows for defining an interpolation that is ideal per aggregate. Furthermore, our aggregate construction process generates aggregates that have an overlapping border to further increase interpolation quality. We explain the complete setup phase for AM-AMG in Section 3.2.

The idea of working with some form of overlapping aggregates as well as distinguishing between different variable types for AMG is already followed by the AMGe method [86, 155, 80, 33]. By additionally using the element stiffness matrix, two (local) measures for classifying the variable connections, determination of the "smooth" error direction and defining the interpolation are introduced. This means additional information beside the linear systems itself, in form of the element stiffness matrix, are necessary. Moreover, the interpolation calculation drastically differs.

Remembering the geometric idea of the function types for the variables inside the aggregates, we easily see a wide range of possibilities to vary the aggregate form. This gives us a very fine-grained control of the coarsening process. Due to this, we can sensitively react to various linear system structures. These features make AM-AMG more robust than aggregative AMG. Especially the overlapping aggregate border is advantageous for solving linear systems arising from Graph Laplacians. Very roughly

spoken, a Graph Laplacian describes the connectivity of a graph in matrix form [40]. On this account AM-AMG is more robust than aggregative AMG. A more detailed analysis for various examples is given in Section 3.3. The effects of these fine-grained splitting control or variation of the interpolation operator are explained in Section 3.5.

The linear systems to which AMG is applied to has continuously been and is increasing in terms of problem size, as the computational power is increasing. Due to this, the parallelization of the linear solver is important to be applicable in simulations on modern compute architectures. The parallelization options of AM-AMG are explained in Section 3.6. As an outlook, we describe why this approach is promising to transfer to a massively parallel shared memory system. The interpolation calculation is highly parallelizable and the splitting can easily be reused in full simulation runs.

3.1 Overview of Algebraic Multiscale

Inspired by the idea of upscaling in reservoir modeling, a new solver approach, Algebraic Multiscale (AMS), has been introduced [163]. AMS uses the cells from the given reservoir model, and groups them together to have smaller coarse grids. These groups are often called wirebaskets, see Figure 3.1. As this process significantly relies on the selected geometric discretization of the reservoir, we use those names that are related to the geometric approaches. When we generalize the AMS approach in the next Section 3.2 in the context of AMG, we change the naming to an AMG-related one.

The constructed wirebaskets for AMS are further distinguished by additional classification of the grid points as vertices V, edges E and interiors I. These labelings reflect the "role" of the points in the coarse grid construction: Vertices describe the coarse grid. Edges then have couplings to the vertices and/or to other wirebaskets and, by this, are directly interpolated from the vertices. At least ideally, interiors have only couplings inside their wirebaskets. Because for the setup of the interpolation only the local wirebaskets are necessary, it is natively parallelizable.

As our generalization of AM-AMG [48] is mainly inspired by the description in [163], we present the key-facts for AMS along this description. However, we note that there exist different ways for constructing AMS wirebaskets and prolongation operators. A broad overview on AMS-related work can be found in [92, 104, 105, 158, 146] and the references therein. But all of them rely on the same principle that is non-algebraic but exploits a geometrical discretization.

Later on, especially for the generalization and fully algebraic formulation in Section 3.2, we show the complete general case. For the moment, as in [163], we describe the AMS algorithm exemplarily for uniformly structured problems based on a 2D Poisson-like problem with a five-point stencil discretization. In these cases, the definition of the wirebaskets is rather intuitive, and, thus, the linear system matrix A in Equation (2.1)

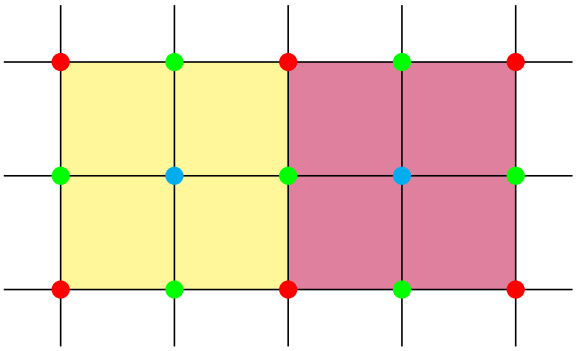


Figure 3.1: Schematic construction of wirebaskets (yellow and purple) of size nine with four vertices, four edges and one interior in a rectangular, uniformly structured grid. The color of the points indicates their point class: vertices are in red, edges in green and interior in cyan.

is easily reordered as

$$A = \begin{pmatrix} A_{[I,I]} & A_{[I,E]} & A_{[I,V]} \\ A_{[E,I]} & A_{[E,E]} & A_{[E,V]} \\ A_{[V,I]} & A_{[V,E]} & A_{[V,V]} \end{pmatrix} \tag{3.1}$$

with submatrices $A_{[\cdot,\cdot]}$. These submatrices include the couplings between the variables that are classified as vertices, edges or interiors. In the algorithmic implementation itself, the reordering is used implicitly and not explicitly calculated. We show the matrix reordering only for reasons of readability and clarity.

In the applications under consideration for the initial AMS developments [1], the linear system (2.1) in combination with the submatrices structure from Equation (3.1) reduces to the following form

$$\begin{pmatrix} A_{[I,I]} & A_{[I,E]} & 0 \\ 0 & \widetilde{A}_{[E,E]} & A_{[E,V]} \\ 0 & 0 & \mathbb{1}_{[V,V]} \end{pmatrix} \begin{pmatrix} u_{[I]} \\ u_{[E]} \\ u_{[V]} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ b_{[V]} \end{pmatrix} \tag{3.2}$$

where $\widetilde{A}_{[E,E]}$ considers the tangential flow that previously was described in the submatrix $A_{[E,I]}$. The vertices form the coarse level and, with an ideal transfer operation between coarse and fine level, we have $b_{[V]} = b_c$. Due to this, the submatrix $A_{[V,V]}$ reduces to the identity $\mathbb{1}_{[V,V]}$ for the vertices.

As the reduced linear system in this specific application has a simple structure, it can easily be solved via submatrix/block elimination, i.e., via a so-called Schur-complement

[1] assuming a five-point stencil of a flow problem discretization, elimination of the $A_{[E,I]}$-submatrix via tangential flow physically based on boundary conditions, and others, for more information see [163]

approach. This defines the interpolation operator

$$\mathcal{P}_{\text{AMS}} = \mathcal{G} \cdot \begin{pmatrix} A_{[I,I]}^{-1} A_{[I,E]} \widetilde{A}_{[E,E]}^{-1} A_{[E,V]} \\ -\widetilde{A}_{[E,E]}^{-1} A_{[E,V]} \\ \mathbb{1}_{[V,V]} \end{pmatrix}, \tag{3.3}$$

with $\mathcal{G}$ describing the implicit reordering of the matrix into separate wirebaskets and point classes.

3.2 Introduction to Aggregative Multiscale AMG

In the previous section, we have seen that the AMS approach uses a Schur-complement-based construction of the interpolation operator $\mathcal{P}_{\text{AMS}}$, considering a uniformly structured grid and other assumptions based on physical information from the application field. Thus, it is not working purely algebraically. In the following, we want to overcome the necessity of geometric discretization information and obtain a purely algebraic setup.

In a first step for a purely algebraic setup of AMS, we redesign the construction process for aggregates, see Section 2.3.4, to reflect the three function types vertices V, edges E and interiors I. Second, we can algebraically calculate the interpolation operator by analyzing the aggregate-local linear system. Finally, we show in Section 3.4 that the described AMG strategy with the newly defined interpolation operator converges under certain mild assumptions.

As starting point for the coarsening process, we use classical aggregative AMG. Due to this, we name the resulting AMG approach, with a new coarsening approach and new interpolation calculation, as Aggregative Multiscale AMG (AM-AMG).

The aggregation process is based on the matrix adjacency graph. For the graph interpretation of the matrix, a distance measure is necessary. For this purpose, we interpret $\frac{1}{a_{ij}}$ as the distance between the two variables i and j, if they are coupled. If two variables i and j are not coupled, the distance is defined as the shortest way over another variable k that is coupled to the variables i and j, i.e. the distance is defined as $\min_k \left(\frac{1}{a_{ik}} + \frac{1}{a_{kj}} \right)$. This definition can be extended if two variables are only reached over a path of more than one variable.

Each built aggregate should have minimal diameter in the matrix adjacency graph, as this minimizes the influence of heterogeneities on the interpolation in each single aggregate. The diameter of an aggregate is defined as the shortest path between those variables that are most distanced in the matrix adjacency graph.

For the calculation of the path length $p(i,j)$ between two variables i and j, we use the Floyd-Warshall algorithm [54], see Algorithm 3.1. We modified this algorithm for our

specific purpose to reduce the number of operations by saving intermediate calculation steps and save overall runtime. Thus, the calculation loops in Lines 11 and 12 in Algorithm 3.1 are restricted to the current aggregate sizes instead of the matrix size n as we are only interested in paths inside an aggregate. Additionally, we introduce intermediate savings for the calculation in Lines 13 and 14 in Algorithm 3.1, since the same couplings need to be considered multiple times. We are using this adapted Floyd-Warshall for the diameter calculations in Line 8 and the path length calculation in Line 24 in Algorithm 3.2.

Algorithm 3.1: General Floyd-Warshall Algorithm

Input: A_f: fine-level matrix $\in \mathbb{R}^{n \times n}$
Data: p : matrix of path lengths $\in \mathbb{R}^{n \times n}$

```
1  initialization of path matrix:
2  for i = 1 to n do
3      for j = 1 to n do
4          if i = j then
5              p(i, i) = 0
6          else if j ∈ N_i then
7              p(i, j) = 1 / a_ij
8          else
9              p(i, j) = ∞
10 calculation of path length between all variables:
11 for i = 1 to n do
12     for j = 1 to n do
13         for k = 1 to n do
14             p(i, j) = min(p(i, j), p(i, k) + p(k, j))
```

The aggregation process starts with an initial aggregate variable that is not included in any aggregate. For creating the current aggregate, all variables k that couple into this aggregate are considered. The diameter d_k, if this variable would be part of the aggregate, is calculated, see Line 8 in Algorithm 3.2. Afterwards, the variable with the smallest diameter d_k is added to the aggregate until the pre-defined maximum aggregate size is reached. Adding the variable k to the current aggregate has some constraints, as vertices and edges can be part of more than one aggregate. The following gives an overview of all conditions for adding the variable k to the aggregate, where on of them needs to be fulfilled:

- k is not included in another aggregate

or

- k is already a vertex in another aggregate, and
  the maximum number of aggregates in which k is a vertex is not reached, and
  the maximum number of vertices for this aggregate is not reached, i.e., $\text{agg}_v < n_v$

  or

- k is already an edge in another aggregate, and
  the maximum number of aggregates in which k is an edge is not reached, and
  the maximum number of edges for this aggregate is not reached, i.e., $\text{agg}_e < n_e$

with $\text{agg}_{v/e}$ the current number of vertices or edges in the constructed aggregate and $n_{v/e}$ the maximum number of vertices or edges per aggregate. These constraints are checked in the Function ADDING_NEXT_POINT(k) starting in Line 10 in Algorithm 3.2. It is possible that an aggregate is finished before the maximum number of aggregate variables is reached, when there are no more coupled variables that fulfill the necessary conditions can be added to the aggregate are available.

Directly after completing one aggregate, the function types vertices, edges and interiors are assigned. To do so, the shortest variable-to-variable path $p(i,j)$ between two variables i and j inside an aggregate is calculated. The per-variable distance $d(i)$ then is the maximum over all variable-to-variable paths inside this aggregate, i.e. $d(i) := \max_{i \neq j} p(i,j)$. All variables of an aggregate with the longest per-variable distance become a vertex, until a user-defined maximal number of vertices n_v is reached. Analogously, all variables of an aggregate with the smallest per-variable distance become an interior, until a user-defined maximal number of interiors n_i is reached. All remaining variables are defined as edges. In Figure 3.2, we show the process of function type assignments in a full aggregate and refer to Lines 26 to 32 in Algorithm 3.2 to complete the pseudo code description of the setup phase of AM-AMG.

The vertices are the coarse-level variables and define the next coarser level.

This manner of awarding the function types to the variables may appear arbitrary at first glance. But when the definition of vertices, edges and interiors for a uniformly structured grid with AMS is analyzed, both perfectly coincide. Moreover, the setting of function types is only necessary for the setup of the local interpolation per aggregate, which is a nearly ideal interpolation approximation. This setup of the interpolation is a localized version of the limit case of direct solver, explained in Section A.2.3. in [138]. In an ideal world, the interpolation is equivalent to a Schur-complement between coarse- and fine-level variables. When additionally the smoother is ideal for all fine-level variables, the overall algorithm yields a direct solver.

Unlike to classical aggregative AMG approaches, it is possible and even necessary for an adequate coarsening rate to use variables in multiple aggregates. This variable re-usage

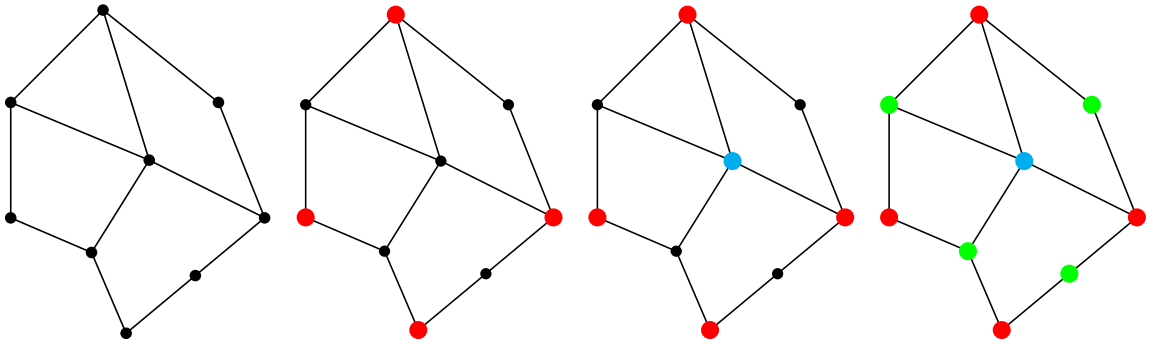


Figure 3.2: Visualization of function types definition for one aggregate in AM-AMG.

Starting point is a finally constructed aggregate. At a first, step the vertices (red colored points) are defined as these variables with maximal variable-to-variable path length inside the aggregate. Afterwards, the interior (cyan colored point) as these variables with minimal variable-to-variable path length is determined. Finally, all remaining variables in the aggregate are assigned as edges (green colored points).

in aggregates in combination with the non-constant interpolation inside the aggregates is the strength of AM-AMG compared to aggregative AMG.

The initialization of the user parameters for the aggregate form (number of vertices, edges and interiors per aggregate) and overlapping of aggregates (number of aggregates in which a vertex or edge variable is included) is based on rectangular aggregates on a uniformly structured grid, see the ideal schematic construction in Figure 3.1. Thus, we set as default that an aggregate consists of four vertices, eight edges and four interiors. Additionally, one vertex can be included in up to four aggregates and edges in up to two per default. Interiors should never be included in more than one aggregate to ensure an optimized interpolation. We discuss possible generalizations of the coarsening strategy based on geometric-like interpretations in Section 3.5. We will see in Section 3.3 that the chosen settings are working well for a huge bandwidth of applications.

For the construction of the interpolation for AM-AMG, we reformulate the linear system (2.1) localized for each aggregate and sorted for the different function types, as

$$\begin{pmatrix} A_{[I,I]} & A_{[I,E]} & A_{[I,V]} \\ A_{[E,I]} & A_{[E,E]} & A_{[E,V]} \\ A_{[V,I]} & A_{[V,E]} & A_{[V,V]} \end{pmatrix} \begin{pmatrix} u_{[I]} \\ u_{[E]} \\ u_{[V]} \end{pmatrix} = \begin{pmatrix} b_{[I]} \\ b_{[E]} \\ b_{[V]}. \end{pmatrix}. \tag{3.4}$$

In contrast to the linear system in Equation (3.2) of the initial AMS, which applies several information based on the application field, we do not apply any such simplifications.

The linear system (3.4), we want to solve with the two-level scheme that we described in Section 2.2, based on the aggregative splitting explained above. Thus, we solve a

defect correct equation.

For the construction process of the interpolation, we assume an ideal-per-aggregate pre-smoothing of the solution of Equation (3.4). Later on, we will see that this assumption is not necessary when we prove the accuracy of interpolation in Section 3.4. However, this coincides with the intuition behind AMS. The assumption of ideal pre-smoothing leads to a zero residual for the interior and edge variables. The solution of the coarse-level defect correction e^c is the solution of the vertex variables $e_{[V]}$. By this, we have the following linear system to be solved for the defect correction on the fine level:

$$\begin{pmatrix} A_{[I,I]} & A_{[I,E]} & A_{[I,V]} \\ A_{[E,I]} & A_{[E,E]} & A_{[E,V]} \\ 0 & 0 & \mathbb{1}_{[V,V]} \end{pmatrix} \begin{pmatrix} e_{[I]} \\ e_{[E]} \\ e_{[V]} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ e^c. \end{pmatrix}. \tag{3.5}$$

The equations for the interior defect correction $e_{[I]}$ and edge defect correction $e_{[E]}$ are solved separately. We start with a reformulation of the first row in (3.5), i.e., the interior defect correction $e_{[I]}$ depending on the edge defect correction $e_{[E]}$,

$$\begin{aligned} 0 &= A_{[I,I]} e_{[I]} + A_{[I,E]} e_{[E]} + A_{[I,V]} e^c \\ \Leftrightarrow e_{[I]} &= -A_{[I,I]}^{-1} A_{[I,V]} e^c - A_{[I,I]}^{-1} A_{[I,E]} e_{[E]}. \end{aligned}$$

We insert this formulation in the second row in (3.5) to gain the solution for the edge defect correction $e_{[E]}$

$$\begin{aligned} 0 &= -A_{[E,I]} A_{[I,I]}^{-1} A_{[I,V]} e^c - A_{[E,I]} A_{[I,I]}^{-1} A_{[I,E]} e_{[E]} + A_{[E,E]} e_{[E]} + A_{[E,V]} e^c \\ \Leftrightarrow 0 &= \left(A_{[E,E]} - A_{[E,I]} A_{[I,I]}^{-1} A_{[I,E]} \right) e_{[E]} + \left(A_{[E,V]} - A_{[E,I]} A_{[I,I]}^{-1} A_{[I,V]} \right) e^c \\ \Leftrightarrow e_{[E]} &= -\hat{A}_{[E,E]}^{-1} \left(A_{[E,V]} - A_{[E,I]} A_{[I,I]}^{-1} A_{[I,V]} \right) e^c, \end{aligned}$$

with $\hat{A}_{[E,E]} = A_{[E,E]} - A_{[E,I]} A_{[I,I]}^{-1} A_{[I,E]}$. By this, we have a completely algebraic formulation for the interpolation of the edges. It is only based on the coarse-level solution e^c and couplings inside this aggregate.

With the same procedure, we calculate the interpolation for the interiors. As a first step, the second row of (3.5) is interpreted as the defect-correction solution of the edges $e_{[E]}$, depending on the defect correction solution of the interior $e_{[I]}$. Afterwards, this is inserted into the first row and resolved for the interior defect correction $e_{[I]}$ with the following interpolation formula

$$e_{[I]} = -\hat{A}_{[I,I]}^{-1} \left(A_{[I,V]} - A_{[I,E]} A_{[E,E]}^{-1} A_{[E,V]} \right) e^c$$

with $\hat{A}_{[I,I]} = A_{[I,I]} - A_{[I,E]} A_{[E,E]}^{-1} A_{[E,I]}$.

In summary, the interpolation operator $\mathcal{P}_{\text{AM-AMG}}$ for AM-AMG is defined as

$$\mathcal{P}_{\text{AM-AMG}} := \mathcal{G}\mathcal{N} \cdot \begin{pmatrix} -\widehat{A}^{-1}_{[I,I]} \left(A_{[I,V]} - A_{[I,E]} A^{-1}_{[E,E]} A_{[E,V]} \right) \\ -\widehat{A}^{-1}_{[E,E]} \left(A_{[E,V]} - A_{[E,I]} A^{-1}_{[I,I]} A_{[I,V]} \right) \\ \mathbb{1}_{VV} \end{pmatrix}, \tag{3.6}$$

with $\mathcal{G}$ describing the reordering of the original linear system (2.1) in the function types and sorted for the single aggregates as in Equation (3.4). Actually, never reordered in the implementation - just written for easier instruction. $\mathcal{N}$ is the normalization of the interpolation formulas such that a constant vector would be interpolated as constant vector.

As the interpolation is constructed independently for each aggregate, we have some conflicts for the interpolation at edges: these can be part of more than one aggregate. In such cases, the edge is referred to one primary aggregate for interpolation calculation. In all other aggregates, no interpolation formula is set up for this edge. When we would use the interpolation of the edges from all aggregates, the Galerkin operator for this variable would rely on much more variables and, thus, the matrix complexity would drastically increase. Additionally, the parallelization of the interpolation for these variables would be more difficult, as we have to be careful when writing the interpolation operator in a distributed manner.

Vertices can also be included in several aggregates, but their interpolation formula is simply the identity, as they span the coarse level and, thus, this causes no problems.

During the entire process, construction of the aggregates and interpolation calculation, no distinction between strong and weak couplings is necessary. This property carries over from the initial idea of AMS. Indirectly, the coupling strength is encapsulated in the aggregate construction. The aggregate construction is based on the interpretation of $\frac{1}{a_{ij}}$ as distance measure between two coupled variables i and j in the matrix adjacency graph. When a coupling is strong, the distance for this coupling is seen as small and, by this, preferably chosen as new addable aggregate point in Line 10 in Algorithm 3.2. This indirect usage of coupling strength is especially advantageous, when the coupling strength is not symmetric: all couplings, irrelevant whether they are strong or weak, are considered for the setup of the interpolation operator.

Furthermore, we can apply the AM-AMG approach also to solve a system with different kinds of unknowns, the unknown-wise approach from Section 2.3.5. This is, for instance, useful for linear systems from petroleum reservoir simulations, as they include different physical unknowns. The inspiration for AM-AMG has been developed for this kind of simulations. For linear systems with different unknowns, the construction process for aggregates is divided to work for each unknown separately. As a consequence, each constructed aggregate only contains variables of one unknown. No further changes

are necessary afterwards, because the interpolation is constructed separately for each aggregate. In the Section 3.3.2, we give a short example for the application of AM-AMG for a simple petroleum reservoir simulation. Further examples can be found in [48].

Before we show two-level convergence for AM-AMG in Section 3.4, we give some results of AM-AMG compared to other well-known setup approaches of AMG in the following section.

Algorithm 3.2: Coarsening Process of the Setup Phase of AM-AMG

Input: A_f: fine-level matrix $\in \mathbb{R}^{n \times n}$
$n_{\text{var}/v/e/i}$: number of variables/vertices/edge/interiors per aggregate

Data: $\text{agg}_{\text{var}/v/e/i}$: current number of variables/vertices/edges/interiors in constructed aggregate

```
1  for i = 1 to n do
2      if i is part of an aggregate then
3          return
4      add i to aggregate, agg_var = 1
5      agg_v = 0, agg_e = 0, agg_i = 0
6      while agg_var < n_agg do
7          for each coupled variable k to this aggregate do
8              calculate diameter d_k if k would be part of this aggregate
9          select k with minimal d_k
10         Function Adding_Next_Point(k):
11             if k is not part of another aggregate then
12                 add k to this aggregate
13                 agg_var = agg_var + 1
14             else if k is part of another aggregate then
15                 if k is available as vertex and agg_v < n_v then
16                     add k as vertex to this aggregate
17                     agg_var = agg_var + 1, agg_v = agg_v + 1
18                 else if k is available as edge and agg_e < n_e then
19                     add k as edge to this aggregate
20                     agg_var = agg_var + 1, agg_e = agg_e + 1
21             else if another variable as in Line 8 exists then
22                 set k to the variable with the next smallest d_k
23                 Adding_Next_Point(k)
24     calculate variable-to-variable paths p(i, j) inside this aggregate
25     calculate per-variable distance d(i)
26     while agg_v < n_v do
27         set variable with the longest variable-to-variable path as vertex
28         agg_v = agg_v + 1
29     while agg_i < n_i do
30         set variable with the smallest variable-to-variable path as interior
31         agg_i = agg_i + 1
32     set all remaining variables as edges
```

3.3 Numerical Examples and Results for AM-AMG

We present three examples: the standard Poisson problem in Section 3.3.1, a Black-Oil problem from petroleum reservoir simulation in Section 3.3.2, and some graph network problems from the huge field of data science problems in Section 3.3.3.

As setup approaches, we chose Ruge-Stüben coarsening with standard interpolation (see Section 2.3.2), aggregative coarsening with piecewise-constant interpolation (see Section 2.3.4) and the previously introduced AM-AMG coarsening with locally optimal interpolation. Ruge-Stüben coarsening is chosen as it is a very well-known standard method and works very good as default setting in many cases. The newly developed method AM-AMG is an improvement to aggregative coarsening and, thus, we want to compare these two methods for various applications.

All benchmarks have been performed on a node with Intel Xeon Gold 6130F dual 16-core CPU and 192 GB RAM.

3.3.1 Poisson Problem

Poisson's equation $-\Delta u = f$ is a very classical elliptic PDE. The discretization of this PDE on a two-dimensional unit square with different mesh resolutions is a well-known benchmark example for the evaluation of AMG approaches. The difficulty to solve Poisson's equation can be increased by using inhomogeneous coefficients, realized by a cell-wise random diagonal tensor K included in Poisson's equation, which then reads as $-\nabla K \nabla u = f$. This variation is chosen as the basic idea of AM-AMG, namely AMS, is constructed in an petroleum-reservoir-related context and, thus, problems with various inhomogeneities occur. By construction of AM-AMG, in the examples with growing inhomogeneities, the convergence of AM-AMG is much better than for aggregative AMG. The interpolation is optimal per aggregate and, by this, the inhomogeneities are represented in the interpolation. This is not possible for the piecewise-constant interpolation in the aggregation approach.

In Figure 3.3, the number of iterations until convergence (relative residual lower than 10^{-8}) are plotted for the three different coarsening strategies. As expected, Ruge-Stüben has the best convergence performance and the lowest sensitivity on the increasing discretization mesh size. The convergence rate of aggregation coarsening and AM-AMG is influenced by the discretization mesh size. They both have a more aggressive coarsening, i.e., less matrix hierarchy levels are constructed. Therefore, they have a higher necessity of approximation in the interpolation. Especially for higher mesh sizes, AM-AMG has a better convergence rate, as this approach profits from the locally optimal interpolation per aggregate, in contrast to the piecewise-constant interpolation for aggregation.

Figure 3.4 shows the convergence history for one specific mesh size. As the previous

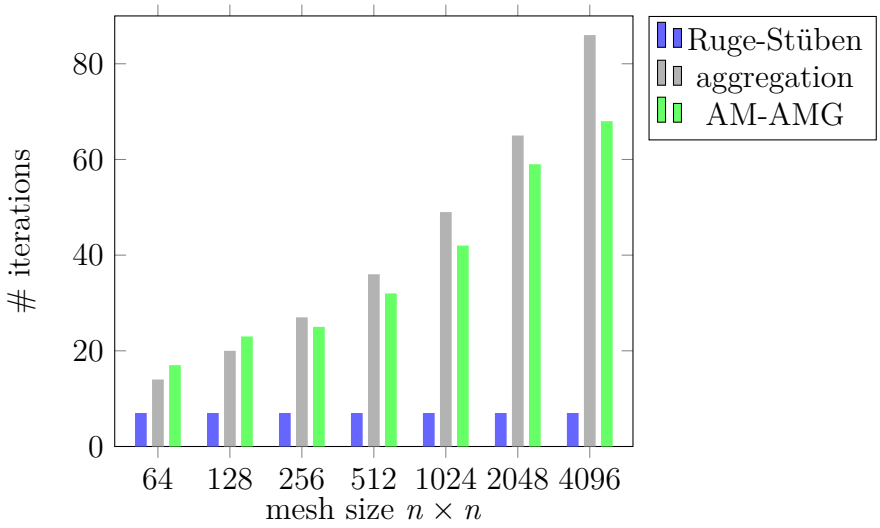


Figure 3.3: Numbers of iterations for different coarsening strategies and various mesh sizes for a 2D homogeneous Poisson problem discretization with a five-point stencil on the unit square.

figure already implies, the Ruge-Stüben coarsening has a continuous convergence history. In contrast, aggregation coarsening and AM-AMG converge much slower. But AM-AMG then has the better residual reduction for already small residuals as the locally optimal interpolation can better handle the eigenfrequencies.

The convergence results for a Poisson problem with a high inhomogeneity rate as in Figure 3.5 are quite similar to the normal Poisson problem. But particularly the convergence difference between aggregation coarsening and AM-AMG for growing discretization mesh sizes grows. AM-AMG shows less sensitivity on the growing mesh size in combination with the more aggressive coarsening, due the locally optimal interpolation setup. This reduced sensitivity regarding inhomogeneities is advantageous in real applications, as we will see for petroleum reservoir simulation in the following.

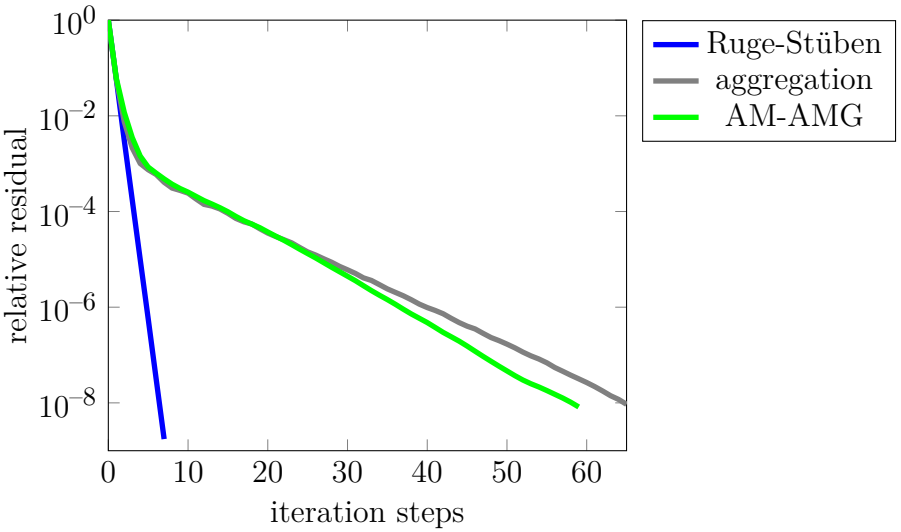


Figure 3.4: Convergence history for different coarsening strategies and a mesh size of 2048x2048 for a 2D homogeneous Poisson problem discretization with a five-point stencil on the unit square.

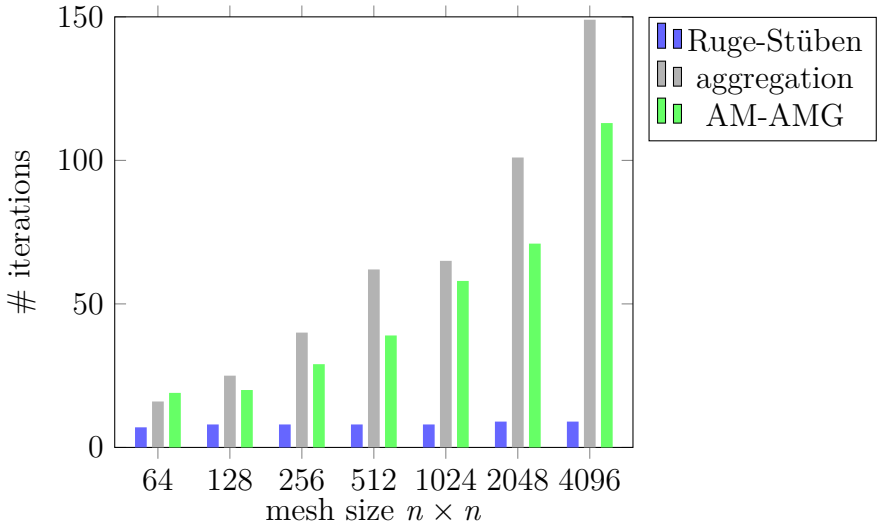


Figure 3.5: Numbers of iterations for different coarsening strategies and various mesh sizes for a 2D Poisson problem discretization and inhomogeneity between 10^{-8} and 10^{8} with a five-point stencil on the unit square.

3.3.2 Petroleum Reservoir Simulation

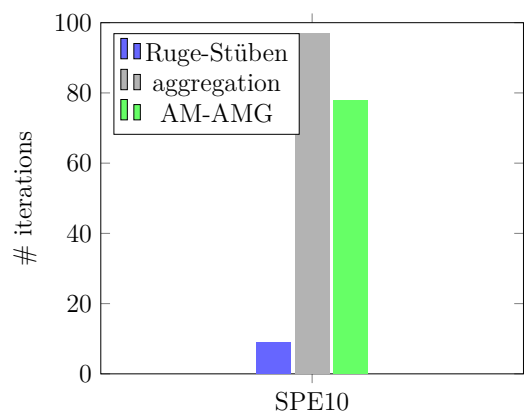
The original, but geometric, idea behind AM-AMG arises from solving simulations of petroleum reservoirs. To demonstrate the application of AM-AMG also in this field, we consider a basic Black-Oil case. For this purpose, we chose a representative linear system from a SPE10 simulation [39].

For solving this problem and others arising from petroleum reservoir simulations, we use the System-AMG approach for coupled systems from fully implicit (FIM) simulations [59, 60]. In this linear solver framework, the construction of the matrix hierarchy is automatically restricted to those physical unknowns, mainly pressure and temperature, that actually need it. Additionally, an incomplete factorization smoother (ILU) on the finest level for the solution phase of AMG is activated to update those physical unknowns that are not considered for the matrix hierarchy construction in the setup phase of AMG. By all of this, matrix properties for the setup phase of AMG are secured to safely apply the coarsening approaches. We refer to [59, 60] for further details.

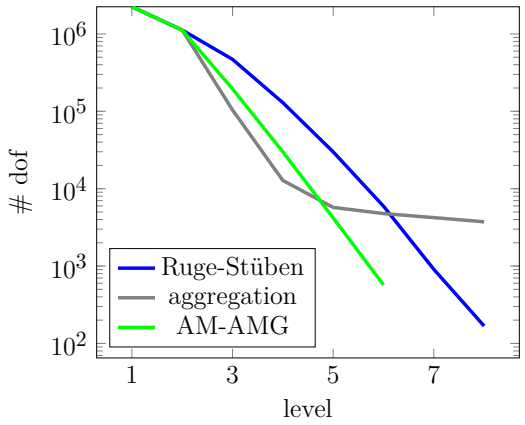
Similar to the Poisson problem, we see in Figure 3.6a the good quality of Ruge-Stüben coarsening. But additionally, we see the drastic difference between aggregation and AM-AMG. AM-AMG needs roughly 20% less iterations than aggregation. This fact is mainly depending on the improved quality of the interpolation as this is now working locally ideal.

Another improvement regarding aggregation, we can see in Figure 3.6b, where we plot the number of variables in the full constructed matrix hierarchy. At the beginning, aggregation has the best coarsening, i.e., the first matrix hierarchy levels are the smallest. But at some point saturation occurs and no further valid coarsening is achieved. AM-AMG has a similar coarsening rate during the entire coarsening process and constructs the lowest number of matrix hierarchy levels. AM-AMG here profits from the overlapping aggregate construction with shared vertices and edges. This construction stabilizes the coarsening rate.

For more examples arising from reservoir simulations and further explanations and analysis, see [48].



(a) Number of iterations for different coarsening strategies for the reference Black-Oil problem SPE10.



(b) Number of variables (#dof) per level of the matrix hierarchy for different coarsening strategies for the reference Black-Oil problem SPE10. Ruge-Stüben and aggregation created 8 matrix hierarchy levels and AM-AMG only 6.

Figure 3.6: Results for the reference Black-Oil problem SPE10

3.3.3 Graph Network Problems

In this section, we analyze our coarsening method for graph network problems. Exemplarily, we chose three example graphs, namely as-Caida, soc-Slashdot0811 and email-EuAll from the SNAP Dataset [90] with a slightly different data origin:

- as-Caida (26475 nodes, 106762 edges) [88, 90]
  This graph analyzes the relationship structure in the internet between customers and providers. Our graph represents these relationships on November, 5th 2007.

- soc-Slashdot0811 (77360 nodes, 905468 edges) [91, 90]
  This graph represents a social network from a technology-related news website generated in November 2008. This graph contains the connectivity between the users of this website.

- email-EuAll (265214 nodes, 420045 edges) [89, 90]
  This graph is an email communication graph from a large European research institution. The data includes some non-existing or spam mail addresses that are not connected to the main part of the graph. In our benchmarks, we only use the main component of the graph.

These graphs are analyzed in the Megaman software [98, 99]. Megaman is a python software that has various scalable manifold learning algorithms implemented. It gives the opportunity to handle data with million of points and hundreds of dimension. For

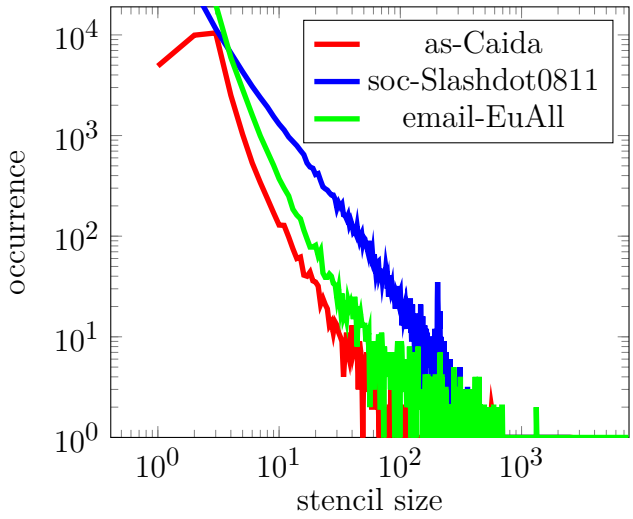


Figure 3.7: Occurrences of stencil sizes for the data science graphs as-Caida, soc-Slashdot0811 and email-EuAll (main component)

analyzing purposes, the graph data is transferred to a matrix as Graph Laplacian. As most algorithms work better for positive definite matrices, the diagonal is slightly increased already in the Megaman software. Graph Laplacians per definition are only positive semi-definite.

All of these graph matrices are more inhomogeneous than the previous Poisson-related problems. Additionally, the sparsity pattern of these matrix kinds is virtually random: there exist a few variables with less couplings and a few variables with much more couplings. In Figure 3.7, we show the frequency with which different stencil sizes in the Graph Laplacians occur for our three chosen graphs. This sparsity pattern inhomogeneity makes classical coarsening like Ruge-Stüben or aggregative difficult at some points, as the methods' idea is based on small discretization stencils.

Due to high inhomogeneities in the number of couplings between variables, AM-AMG is better suited for this kind of problems. The idea of vertices for AM-AMG is ideally suited for these problem cases, as they allow to handle more than one aggregate by one variable, see Figure 3.12. Initially inspired by the geometric idea, one vertex can only be included in up to four aggregates. But this number can be varied by a user parameter or by a heuristic based on the matrix connectivity itself, see Section 3.5.3. Due to this variation, the handling of variables with high numbers of couplings is much easier, as they become vertices and, thus, are part of many aggregates. Hence, a representation of the highly coupled variables on coarser matrix hierarchy levels is ensured.

In contrast to AM-AMG, Ruge-Stüben coarsening struggles in all of our chosen example cases to create a useful matrix hierarchy with a reasonable amount of memory, visualized via matrix complexity in Figure 3.8, and in an adequate setup time, see Figure 3.9a.

Aggregation coarsening partly fails completely to construct a matrix hierarchy, as

no reasonable coarse grid size to apply a direct solver is reached. For the examples email-EuAll and as-Caida no matrix hierarchy with a suitable coarse grid size for a direct solver is constructed. Thus, in both Figures 3.8 and 3.9, only the results for soc-Slashdot can be presented. While, for this example, aggregation shows the best results in setup time and matrix complexity, it is not generally applicable to other examples from the large network dataset collection [90] as it often fails to create a usable matrix hierarchy. The difficulty for aggregation coarsening is the varying stencil size and the non-adequate handling of variables with many couplings. These variables are, by construction, only part of one single aggregate. This fact completely neglects the importance of such variables for the graph structure.

AM-AMG has no problem to construct a matrix hierarchy with a reasonable coarse grid size and an acceptable amount of memory. Opposed to aggregation coarsening, the inhomogeneity in the matrix stencils are now transferred to the coarser matrix hierarchy levels. Such variables with many couplings to other variables are preferred set as vertices. By this, they can be included in more aggregates. As the interpolation is locally ideal per aggregate, the interpolations for the coupled variables are more specific calculated than with the constant interpolation for aggregation coarsening.

To finalize our evaluation, we now look at the solution phase for all examples with the tested coarsening approaches. In all three examples it holds that when a matrix hierarchy is successfully constructed, the quality of this is similar for the different setup approaches in terms of the convergence rate. To compare the convergence behavior, we consider the main key facts (number of iterations and total residual reduction) about this in Table 3.1. For the two examples soc-Slashdot0811 and as-Caida, these are in the same range over all coarsening approaches. For email-EuAll, the convergence for Ruge-Stüben is much better. But this comes at the cost of a drastically higher setup time, see Figure 3.9a, and memory consumption. In Figure 3.9b, we see that AM-AMG for all three cases has the fastest overall time, which includes setup and solution time.

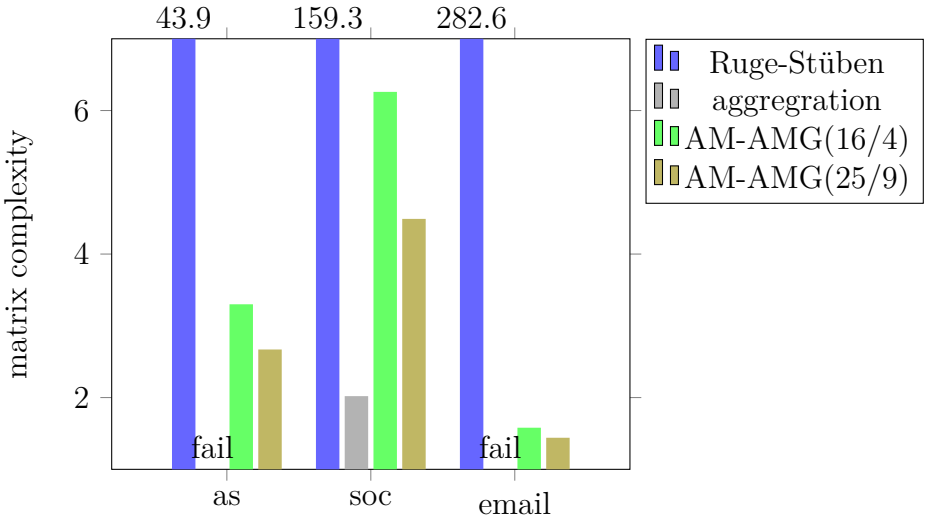
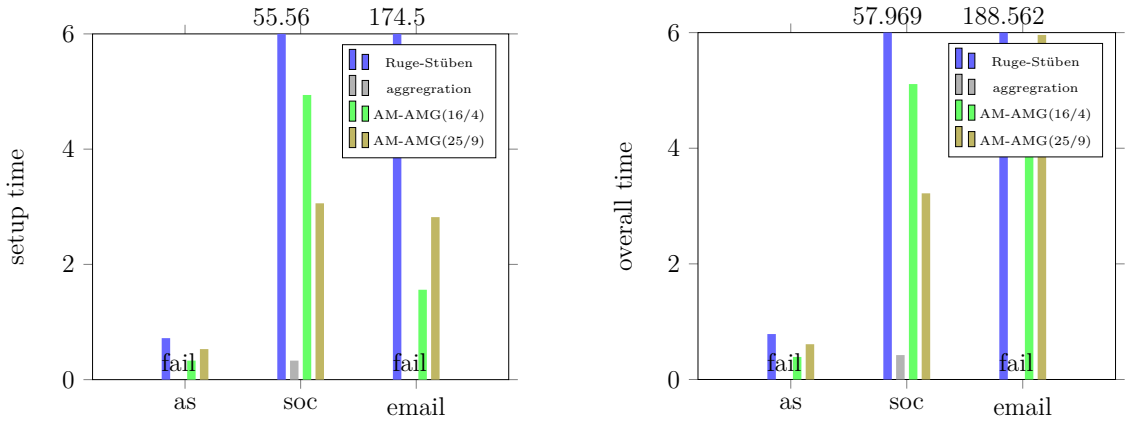


Figure 3.8: Results of the matrix complexity for the data science graphs as-Caida, soc-Slashdot0811, email-EuAll (main component).



(a) Setup time

(b) Overall time

Figure 3.9: Plot of the setup and overall time for the data science graphs as-Caida, soc-Slashdot0811 and email-EuAll (main component).

	setup approach	iterations	final residual reduction
as	Ruge-Stüben	9	$0.248 \cdot 10^{-8}$
	aggregation	fail	fail
	AM-AMG(16/4)	18	$0.757 \cdot 10^{-8}$
	AM-AMG(25/9)	18	$0.962 \cdot 10^{-8}$
soc	Ruge-Stüben	6	$0.159 \cdot 10^{-8}$
	aggregation	8	$0.105 \cdot 10^{-8}$
	AM-AMG(16/4)	8	$0.105 \cdot 10^{-8}$
	AM-AMG(25/9)	7	$0.432 \cdot 10^{-8}$
email	Ruge-Stüben	15	$0.581 \cdot 10^{-8}$
	aggregation	fail	fail
	AM-AMG(16/4)	132	$0.950 \cdot 10^{-8}$
	AM-AMG(25/9)	127	$0.858 \cdot 10^{-8}$

Table 3.1: Main information about convergence behavior for the data science graphs as-Caida, soc-Slashdot0811 and email-EuAll (main component).

3.4 Two-Level Convergence for AM-AMG

Now we show how AM-AMG can be embedded in the theoretical framework summarized in Chapter 2. We especially focus on the accuracy of interpolation, cf. Section 2.4.3, to prove the convergence of a two-level AMG method with post-smoothing that uses AM-AMG as coarsening and interpolation strategy.

We evaluate the convergence theory for a two-level AMG-hierarchy with a symmetric and positive defined M-matrix A. As shortly outlined in Section 2.4.3, to show convergence for an AMG method we have to prove the prerequisites of Theorem 2.3. For easy readability and traceability of the proof, we introduce some additional notations.

When the coarsening process is finished, we analyze the incurred aggregates and set f as the maximum number of fine-level variables (edges and interiors) per aggregate, c as the maximum number of coarse-level variables (vertices) per aggregate and p as the maximum number of aggregates that include the same variable. For each fine-level variable i, we define the set C_i of coarse-level variables, i.e., vertices that span the aggregate that include variable i, i.e.,

$$C_i := \{\text{coarse-level variables of the aggregate that include } i\} \text{ for } i \in F.$$

That is, the variable i is interpolated from C_i. The set C_i can be further divided into variables with direct couplings to the variable i

$$N_i := \left\{ j \in C_i | a_{ij} \neq 0 \right\}$$

and the variables without a direct coupling

$$N_i^0 := \left\{ j \in C_i | a_{ij} = 0 \right\}$$

and, thus, $C_i = N_i \cup N_i^0$. Contrary to the definition in Section 2.3.1, N_i and N_i^0 are restricted to the neighborhood in one aggregate here.

For the interpolation operator, in addition to the derived formula (3.6), we use the following two notations

$$\mathcal{P}_{\text{AM-AMG}} := \mathcal{GN} \cdot \begin{pmatrix} \begin{pmatrix} -\hat{A}_{II}^{-1} \left(A_{IV} - A_{IE} A_{EE}^{-1} A_{EV} \right) \\ -\hat{A}_{EE}^{-1} \left(A_{EV} - A_{EI} A_{II}^{-1} A_{IV} \right) \end{pmatrix} \\ Id_{VV} \end{pmatrix} = \begin{pmatrix} I_{FC} \\ I_{CC} \end{pmatrix} = \left(w_{ij} \right)_{i,j} \quad (3.7)$$

where $\hat{A}_{II} = A_{II} - A_{IE} A_{EE}^{-1} A_{EI}$ and $\hat{A}_{EE} = A_{EE} - A_{EI} A_{II}^{-1} A_{IE}$ are defined as in Section 3.2, $\mathcal{G}$ denotes the matrix that provides the implicit reordering, and $\mathcal{N}$ is the normalization.

It follows that $0 \leq w_{ij} \leq 1$ for all variables i and j and $\sum\limits_j w_{ij} = \sum\limits_{j \in C_i} w_{ij} = 1$ for all variables i, as A is an M-matrix and i is only interpolated from one specific aggregate. Even when the variable i belongs to more aggregates (as it is possible for edges).

With these notations, we are now able to derive an estimate for the accuracy of interpolation.

3.1 Theorem
For a C/F-splitting and interpolation as described in Section 3.2, Equation (2.15)*, i.e.,*

$$\| e_F - I_{FC} e_C \|_{0,F}^2 \leq \hat{\tau} \| e \|_1^2, \quad (3.8)$$

with $\hat{\tau}$ *depending on* $\tau := \max\limits_{\substack{i \in F \\ k \in N_i}} \frac{a_{ii}}{|a_{ik}|}$*, c, f and p, is fulfilled.*

Proof:
We start with the evaluation of the Inequality (2.15) for the AM-AMG interpolation and use the fact that the rowsum of the interpolation operator is normalized to 1. Then we have

$$\| e_F - I_{FC} e_C \|_{0,F}^2 = \sum_{i \in F} a_{ii} \left(\sum_{k \in C_i} w_{ik} \left(e_i - e_k \right) \right)^2 . \quad (3.9)$$

In the next step, we use the Cauchy-Schwarz-inequality $|\langle u, v \rangle|^2 \leq \|u\|^2 \|v\|^2$ for the standard Euclidean scalar product. In our use case, we choose $u = \left(\sqrt{w_{ik}} \right)_k$ and

$v = \left(\sqrt{w_{ik}}\,(e_i - e_k)\right)_k$. Thus, the Equality (3.9) can be estimated as

$$\|e_F - I_{FC}e_C\|_{0,F}^2 \leq \sum_{i\in F} a_{ii} \sum_{k\in C_i} w_{ik}\,(e_i - e_k)^2\,. \tag{3.10}$$

Due to normalization of the rowsum, all entries of the interpolation operator satisfy $w_{ij} \leq 1$ and, as A is a M-Matrix, $w_{ij} \geq 0$.

Furthermore, we now distinguish between the sum over non-zero and zero off-diagonal couplings. We note that, despite $a_{ij} = 0$, w_{ij} can be non-zero and recall that $C_i = N_i \cup N_i^0$. The sum over the non-zero off-diagonal couplings, can be (directly) estimated by the norm $\|.\|_1$, see Section 2.4.1. Formally, this reads as (Formula (A.4.12) in [137]):

$$\|e\|_1^2 \geq \sum_{i\in F}\left(\sum_{k\in N_i} (-a_{ik})\,(e_i - e_k)^2 + s_i e_i^2\right) \tag{3.11}$$

$$\geq \sum_{i\in F}\sum_{k\in N_i} (-a_{ik})\,(e_i - e_k)^2 \tag{3.12}$$

with the row sum $s_i = \sum_j a_{ij} \geq 0$. Thus, we have the inequalities

$$\|e_F - I_{FC}e_C\|_{0,F}^2 \leq \sum_{\substack{i\in F\\ k\in C_i}} a_{ii}\,(e_i - e_k)^2 = \sum_{\substack{i\in F\\ k\in N_i}} a_{ii}\,(e_i - e_k)^2 + \sum_{\substack{i\in F\\ k\in N_i^0}} a_{ii}\,(e_i - e_k)^2 \tag{3.13}$$

$$\leq \tau \sum_{\substack{i\in F\\ k\in N_i}} (-a_{ik})\,(e_i - e_k)^2 + \sum_{\substack{i\in F\\ k\in N_i^0}} a_{ii}\,(e_i - e_k)^2 \tag{3.14}$$

$$\leq \tau\|e\|_1^2 + \sum_{\substack{i\in F\\ k\in N_i^0}} a_{ii}\,(e_i - e_k)^2 \tag{3.15}$$

$$= \tau\|e\|_1^2 + \sum_{\substack{i\in F\\ k\in N_i^0}} a_{ii}\left(e_i - e_j + e_j - e_k\right)^2. \tag{3.16}$$

As the variables i and k belong to the same aggregate, there exists at least one coupling variable j between them inside this aggregate, i.e., $a_{ij} \neq 0 \neq a_{jk}$. Moreover, there holds $a_{ii} \leq \tau|a_{ij}| = \tau|a_{ji}| \leq \tau a_{jj} \leq \tau^2|a_{jk}|$, which allows to estimate

$$\|e_F - I_{FC}e_C\|_{0,F}^2 \leq \tau\|e\|_1^2 + 2\sum_{\substack{i\in F\\ k\in N_i^0}} \left(\tau\left(-a_{ij}\right)\left(e_i - e_j\right)^2 + \tau^2\left(-a_{jk}\right)\left(e_{j(k)} - e_k\right)^2\right). \tag{3.17}$$

Finally, we have to resolve the two remaining sums in terms of the energy norm and, to do so, consider the occurrence of such indirect couplings in one aggregate. For a "triangle coupling" $i-j-k$, the coupling $j-k$ can occur for each fine-level variable in one aggregate and for each aggregate that includes $j-k$. Thus, in total $f \cdot p$ times. The coupling $i-j$ can occur for each coarse-level variable in one aggregate and for each aggregate that includes $i-j$. Thus, in total $c \cdot p$ times. As with the sub-sum of Inequality (3.14) regarding N_i, the sums can be estimated with the norm $\|.\|_1$, compare to [124, 137]. Overall, we now have the inequality

$$\begin{aligned} \|e_F - I_{FC} e_C\|_{0,F}^2 &\leq \tau \|e\|_1^2 + 2\tau cp \|e\|_1^2 + 2\tau^2 fp \|e\|_1^2. && (3.18) \\ &= \left(\tau + 2\tau cp + 2\tau^2 fp\right) \|e\|_1^2 =: \hat{\tau} \|e\|_1^2. && (3.19) \end{aligned}$$

It is possible to extent the estimation for indirect couplings up to the length of the aggregate size. However, involving a larger constant $\hat{\tau}$, which then also only depends on τ, c, f, p and some binomial coefficients to take all possible paths in the adjacency graph into account. ■

With this theorem, Theorem 2.3 is fulfilled stating that the convergence condition in Theorem 2.2 for a two-level method with post-smoothing is fulfilled. Overall, we have shown that AM-AMG yields convergence as a two-level post-smoothed AMG method for a symmetric positive definite M-matrix A. The previously presented results carry over to a two-level AMG-method that uses pre- and post-smoothing as already explained in Remark 2.4.

The presented proof shows that the convergence of AM-AMG depends on various matrix and coarsening related parameters.

The depending parameter τ indicates two properties. On the one hand, the matrix condition number has an influence on the expected convergence rate. On the other hand, indirectly, the coupling strength of the adjacency graph is taken into account. Hence, no explicit distinction between strong and weak couplings, as for Ruge-Stüben coarsening, see Section 2.3.2, is necessary.

The aggregation process itself is represented by the two parameters f and c in the convergence estimation. Finally, the parameter p takes the density of the constructed interpolation into account.

For an M-matrix, we know that τ is limited. Additionally, we have f, c and p fixed for a chosen AM-AMG setup. Therefore, the expected convergence rate as $\sqrt{1 - \frac{\sigma}{\hat{\tau}}}$, see Theorem 2.2 and the detailed explanations in [124, 137], is bounded away from 1 and, hence, we expect good convergence.

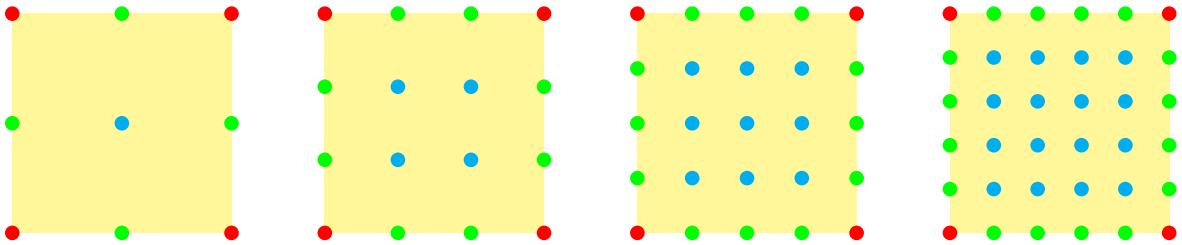


Figure 3.10: Visualization for varying aggregate sizes by increasing number of edges (green points) between two vertices (red points). The number of vertices per aggregate is fixed to four and the number of included interiors (cyan points) grows dynamically with growing number of edges. For easier visibility we chose a rectangular aggregate to demonstrate the effect.

3.5 Control Options of AM-AMG

For the coarsening process described in Section 3.2, various variations are possible. In most cases of our benchmarks, we use the setting with aggregates of size 16 along with four vertices, eight edges and four interiors. They have shown to be ideal in the sense of providing a good balance between coarsening rate and convergence behavior. These settings are interpreted as upper limits and do not need to be fulfilled for each aggregate in the coarsening process. In an overall view, these settings are a good working basis for a wide range of application examples with the best compromise between sufficiently fast coarsening and robust interpolation. With the following presented modifications, we have some fine-grained possible adjustments for AM-AMG to further support this compromise.

3.5.1 Size of Aggregates

The easiest parameter to vary is the aggregate size. The total number of aggregate elements is varied by a variation of the number of edges between two vertices and, therefore, a growing set of interior variables, see Figure 3.10 for a visualization of this process. This allows for a more aggressive coarsening rate. As a consequence, less levels are created and less memory is used. But this can affect the robustness of the interpolation, cf. the parameters f and c in the convergence proof in Theorem 3.1. With this variation, the aggregates are stretched or shrinked.

Table 3.2 shows the results of the variation of aggregate size evaluated by the matrix complexity, setup runtime and number of iterations. As expected, the matrix complexity is reduced, when the aggregate size grows: The rate between vertices, which define the coarse level variables of the new matrix hierarchy level, and edges and interiors is getting lower. For all aggregate sizes, the number of iterations till convergence is quite similar. In the setup runtime, we observe a different effect. For the soc-Slashdot0811 example, the setup runtime is becoming smaller with growing aggregate size, as there

	number of (allowed) edges	matrix complexity	setup runtime	iterations	overall runtime
soc	1	12.59	16.16	7	16.45
	2	6.26	4.95	7	5.14
	3	4.49	3.08	8	3.23
	4	3.42	2.73	8	2.88
email	1	2.00	1.52	126	4.68
	2	1.58	1.52	127	4.38
	3	1.44	2.68	132	5.57
	4	1.37	6.02	128	8.84
as	1	4.98	0.44	19	0.54
	2	3.26	0.34	18	0.39
	3	2.67	0.38	18	0.43
	4	2.20	0.72	18	0.78

Table 3.2: Results for the data science graphs as-Caida, soc-Slashdot0811 and email-EuAll (main component) for a variation of the size of the aggregates by changing the number of (allowed) edges. The number of vertices is set to 4.

are less new aggregates with a searching process for suitable variables necessary. In contrast, we observe that for the email-EuAll at some aggregate size the setup runtime is growing again: The underlying graph structure is not well suited for that size of aggregates then. Various experiments have shown that choosing two edges between two vertices is the best balance between matrix complexity and setup runtime. For a rectangular aggregate form in mind this means eight edges and four interiors.

3.5.2 Number of Vertices per Aggregate

As another parameter, the maximum number of vertices per aggregate can be changed. We have chosen the geometric 2D interpretation of AMS on a uniformly structured grid as starting point and, thus, have a square with four vertices in mind. But it is possible to change this to three (triangle in a geometric interpretation) or five (pentagons in a geometric interpretation) vertices per aggregate and also to higher numbers. Figure 3.11 gives a geometric visualization of the different shape forms for a better understanding. This parameter change gives a more fine-grained control of the coarsening rate, which is especially important for linear systems with bigger matrix stencils, e.g., in problems from data science.

Table 3.3 shows the result of varying the number of vertices per aggregate for three different data matrices. As evaluation criterion, the matrix complexity, as an indicator for memory consumption, the setup time and the total number of iterations, as convergence criterion, are used.

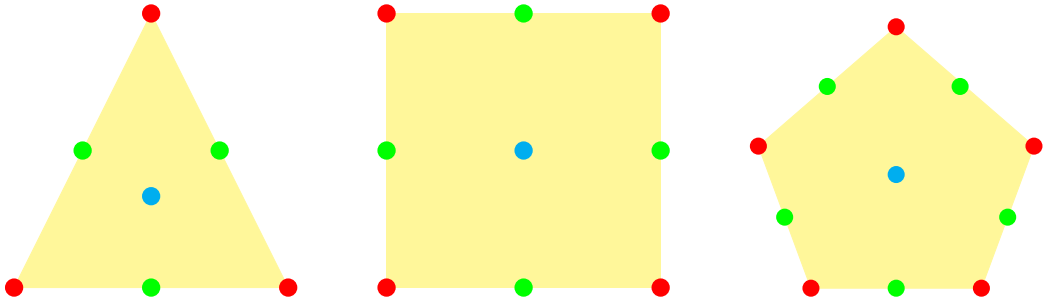


Figure 3.11: Visualization of different shapes of aggregates, which are defined via the number of vertices (red points) per aggregate. For easier visibility, we chose a uniform distribution of the points. The number of edges (green points) and interiors (cyan points) is adapted to the geometrically inspired shape.

The results are quite similar in all cases, but there are slight differences. As expected, the matrix complexity for triangles is the lowest: The rate of aggregates that contain the maximum requested number of variables is the highest. For squares and pentagons in the coarsening strategy, the advantage of lower matrix complexity is not clearly visible for these examples. When using pentagons, the rate of vertex variables is theoretically decreasing. But as the aggregate size is growing, there exist more aggregates that are not filled up to the maximum requested size. Thus, the matrix complexity requires a careful trade-off. The number of iterations and the setup runtime is in a similar range in all our example cases. On average, using the idea of rectangular aggregates, i.e., four vertices per aggregate, is the best parameter setting.

	number of vertices per aggregate	matrix complexity	setup runtime	iterations	overall runtime
as	3	2.70	0.44	19	0.54
	4	3.26	0.34	18	0.41
	5	3.14	0.66	18	0.73
soc	3	5.35	3.96	7	4.13
	4	6.26	4.95	7	5.12
	5	5.94	5.31	8	5.50
email	3	1.42	1.31	132	4.43
	4	1.58	1.52	127	4.54
	5	1.70	4.69	126	7.74

Table 3.3: Results for the data science graphs as-Caida, soc-Slashdot0811 and email-EuAll (main component) for different shapes of aggregates, which are defined via the number of vertices per aggregate (triangle (3), square (4) and pentagon (5)). The number of edges between two vertices is set to 2 for each shape.

3.5.3 Number of Aggregates per Vertex

To better handle inhomogeneties in the matrix stencils, the number of included aggregates per vertex is variable in addition. By this, it is possible to strengthen variables that are starting points of many couplings, e.g., as in network graphs, and to enhance the influence of these variables on coarser levels. Figure 3.12 visualizes one vertex variable to be included in more aggregates and having more couplings to edges and interiors.

Table 3.4 shows the results for varying the number of aggregates that can include the same point as vertex. In contrast to the variation of the number of vertices per aggregate, as in Section 3.5.2, for the usage of vertex in aggregates a clear tendency is observed. As all these data matrices have some points with huge matrix stencils, an increase of the number of aggregates per vertex reduces the matrix complexity and setup runtime. As we can conclude from the stable number of iterations per chosen example, the quality of the constructed matrix hierarchy is similar in all cases. The automatic detection of the "ideal" number of aggregates per vertex for the current example results in a good to medium value of the matrix complexity, but comes at some setup runtime overhead, as the number of couplings for the full matrix has to be reconsidered.

	number of aggregates per vertex	matrix complexity	setup runtime	iterations	overall runtime
as	automatic	2.77	0.34	18	0.41
	4	3.26	0.32	18	0.40
	10	2.66	0.34	18	0.40
	20	2.54	0.34	18	0.40
	50	2.30	0.36	18	0.43
	100	2.12	0.40	18	0.45
soc	automatic	3.67	4.35	8	4.49
	4	6.26	4.97	7	5.15
	10	4.41	3.64	8	3.79
	20	3.54	3.06	7	3.18
	50	2.90	2.82	8	3.00
	100	2.66	2.92	8	3.05
email	automatic	1.45	1.60	128	4.49
	4	1.58	1.52	127	4.44
	10	1.44	1.71	125	4.57
	20	1.37	1.75	127	4.61
	50	1.33	1.80	123	4.65
	100	1.30	2.09	121	4.86

Table 3.4: Results for the data science graphs as-Caida, soc-Slashdot0811 and email-EuAll (main component) for a variation of the number of aggregates that can include the same vertex. The standard number is 4. "automatic" means that the average number of couplings per matrix row is considered. The aggregates have our standard size of 16 with four vertices, eight edges and four interiors.

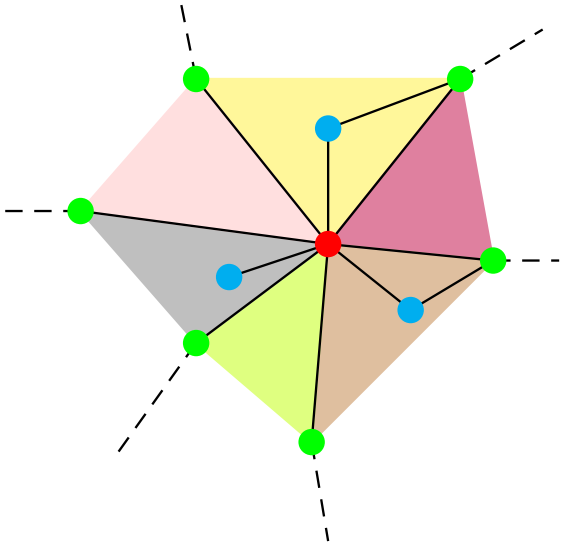


Figure 3.12: One vertex variable (red colored point) is origin to multiple aggregates (yellow, purple, brown, green, gray and pink) with shared edge variables (green colored points) and unique interior variables (cyan colored points).

3.5.4 Variation of Interpolation Weights

Supplementary to an adjustment of the coarsening, adaptations for the interpolation operator are possible. When we look at the interpolation formulas in Equation (3.6), in the formulas for edges and interiors we have two summands. One summand describes the coupling interaction between edges and interiors and the other one between edge/interior and vertex. Sometimes, it turns out to be useful to neglect these mixed interactions, or to give them less weights.

Another thinkable modification is the handling of the mixed couplings in the setup of $\hat{A}_{EE}$ and $\hat{A}_{II}$. The idea to (optionally) consider variations for these interpolation factors is inspired by the AMS idea in [163]: There are some couplings between edges and interiors that have been neglected during the setup process due to physical backgrounds knowledge. Adding some control factors in the interpolation formulas in Equation (3.6) gives the following interpolation operator

$$\mathcal{P}_{\text{AM-AMG}} = \mathcal{GN} \cdot \begin{pmatrix} -\hat{A}_{II}^{-1}\left(A_{IV} - \gamma_I \left(A_{IE}A_{EE}^{-1}A_{EV}\right)\right) \\ -\hat{A}_{EE}^{-1}\left(A_{EV} - \gamma_E \left(A_{EI}A_{II}^{-1}A_{IV}\right)\right) \\ Id_{VV} \end{pmatrix}, \tag{3.20}$$

with the individual parameters γ_I and γ_E to define the coefficients weights for the edge or interior interpolation, and $\hat{A}_{II} = (A_{II} + \delta_{EI}A_{EI} + \delta_{IE}A_{IE}) - A_{IE}A_{EE}^{-1}A_{EI}$ and $\hat{A}_{EE} = (A_{EE} + \delta_{EI}A_{EI} + \delta_{IE}A_{IE}) - A_{EI}A_{II}^{-1}A_{IE}$ with the individual parameters δ_{EI} and δ_{IE} to define the effect of the couplings between interiors and edges in both coupling directions.

We evaluated the effect of these four parameters by a machine learning analysis [61] to improve the performance for the email-EuAll example. By this, the number of iterations with one setting for all four parameters can be reduced to 129 for aggregates with three variables per edge (previously 132, see Table 3.2) and to 126 for aggregates with two variables per edge (previously 127, see Table 3.2).

3.6 Parallelization of AM-AMG

As the AM-AMG includes two parts, the coarsening process and the calculation of the interpolation operator, we consider both parts separately for parallelization. We focus on moderate shared-memory parallelization in the scope of this work because we are mainly interested in the numerical applicability of a parallelized AM-AMG. We use a row-wise separation of the linear system (2.1) in partitions. Hence, a parallelization with distributed memory is possible in the same manner.

As the coarsening process is based on an extension of aggregative AMG, the parallelization essentially boils down to parallelization of aggregative AMG [149] - with some adaptions due to using function types in combination with aggregates. To ensure (eligible) determinism during the coarsening process, we consider two parallelization strategies: one simple but slightly approximate and one that is mathematically accurate, at the expense of some sequential part.

The easiest way of parallelization is to split the linear system into different row-wise partitions and to separately and independently build aggregates within these partitions. To ensure determinism on the partitions, a "first come, first serve" mechanism is implemented for the variables that belong to more than one aggregate and, thus, could be involved in different partitions. A wide range of benchmarks show that this gives a remarkably robust parallelized aggregation structure and, therefore, serves as our default.

The mathematically accurate method starts with the same partitions as the simple parallelization. But the variables included in these partitions are ordered internally, such that variables with couplings to other partitions are at the beginning. Those variables that couple to other partitions are then aggregated in a serial manner. Afterwards, the remaining variables are aggregated in parallel. By this, the coarsening result is closer to the sequential process presented in Section 3.2. However, this comes at the expense of some sequential part. This is typically growing with an increasing hierarchy level, as in most cases the coarser level matrices are denser.

To decrease the sequential part and to increase the parallel performance of the setup, the size of the aggregates can be increased. By this increment, the relation between multiply used variables and only single-used variables in the aggregation process is reduced. Thus, we limit the serial part of the coarsening algorithm on partition borders. But we

have to remind that the aggregates are constructed by a diameter minimization idea that includes the Floyd-Warshall algorithm (see Section 3.2 or Algorithm 3.1), whose runtime is cubically related to the aggregate size. Therefore, an unlimited increase of the aggregate size for parallelization reasons is not possible. Moreover, an increase of the aggregate size would also have a negative impact on the robustness of the coarsening approach.

After the coarsening process is parallelized, the interpolation is natively parallelized due to the localized formulation. Each aggregate has a strictly local calculation of the interpolation, so that a distribution of the aggregates already realizes the parallelization. As in the serial case, the edges are interpolated via their primary aggregate.

We evaluate the setup parallelization for a three-dimensional elasticity problem in Figure 3.13. Each bar shows the complete setup time in sum. By using dark and light colors, we split the setup into both relevant parts - coarsening and interpolation.

As in Section 3.3, for comparison we use partition-local Ruge-Stüben coarsening and the aggregation approach. The aggregate size for the aggregation coarsening is specifically chosen such that the constructed matrix hierarchy has a coarse-level size in the same order as for AM-AMG. The convergence rate for aggregation and AM-AMG is similar (roughly 30 iterations until convergence is reached). As expected and previously seen, Ruge-Stüben is quite better in this case (roughly 8 iterations until convergence is reached).

The performance behavior for AM-AMG is similar to Ruge-Stüben and aggregation coarsening for two and four OpenMP threads. The parallel performance gain for 8 OpenMP threads is worse than for the both other strategies, as the search effort at partition borders with the function types (vertex, edge and interior) has a strong influence in this case.

We split the normalized setup time in Figure 3.13 to illustrate the difference of the parallelization potential of the two setup parts of AM-AMG. This discrepancy, as the interpolation parallelization is quite more efficient, can be turned to an advantage. As the interpolation is easily parallelly recalculable, it is possible to reuse only the coarsening. If AMG technology is used as a linear solver in a full simulation run, e.g., a time-evaluation with a fixed discretization mesh, the linear systems only has in many cases a slight coefficient changes compared to the previous ones and the overall structure remains. The separation of the coarsening and the calculation of the interpolation operators between the different hierarchy levels can be done separately with AM-AMG. Thus, the expensively computed matrix hierarchy can efficiently be reused in consecutive similar linear solver calls, as the accuracy of the solution process is ensured by a frequent re-calculation of the interpolation operator.

This is especially interesting for transferring AMG technology to graphic processing units (GPUs): The main disadvantage of AMG algorithms for GPU usage is the se-

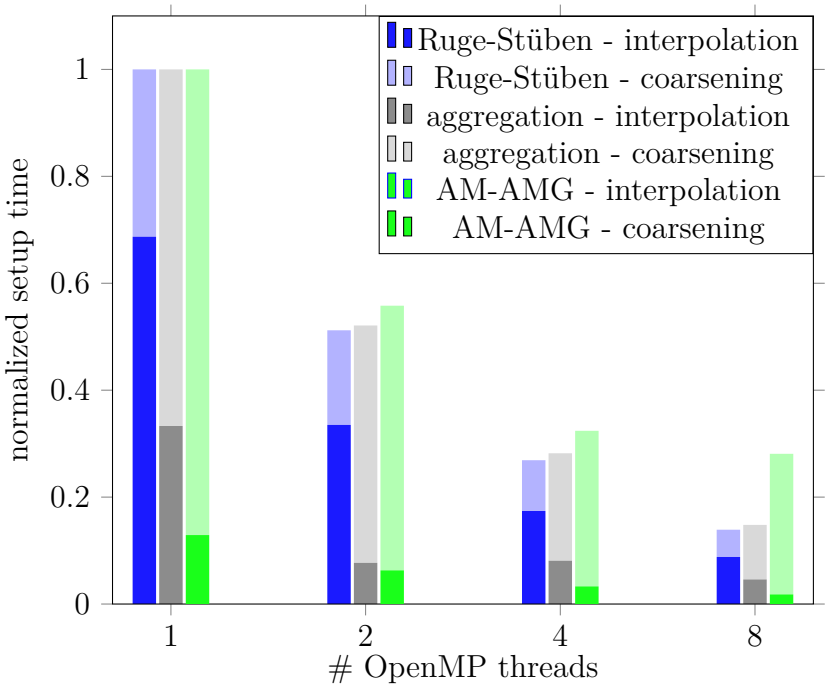


Figure 3.13: Normalized setup time for different amounts of parallelism for the three-dimensional elasticity problem on a discretized cube with 100 discretization nodes per side.

quential and memory intense coarsening process. To have a high reusability of the coarsening reduces the impact of this algorithmic part for the overall runtime. By a highly parallelized interpolation, this is enhanced further.

CHAPTER 4

EP-AMG - New Generic AMG Solution Approach for Eigenproblems

In this chapter, we are going to discuss the iterative solution of generalized eigenproblems with AMG (EP-AMG). Contrarily to the problems from the previous chapter, these are non-linear problems. In the most general case, these have the form

$$Av_k = \lambda_k M v_k \text{ for } k = 1, \ldots, n \tag{4.1}$$

with the matrices $A, M \in \mathbb{C}^{n \times n}$ and the eigenvectors $v_k \in \mathbb{C}^n$ corresponding to the eigenvalues $\lambda_k \in \mathbb{C}$. In principle, a direct solution of this problem is not possible due to the non-linearity of the problem. Depending on the requirement in the application field where the generalized eigenproblem occurs, all eigensolutions, or only a few smallest/largest eigenvalues in terms of the absolute value, or only a few eigenvectors corresponding to the smallest/largest eigenvalues in terms of the absolute value are to be computed.

As there is a wide range of what exactly to solve for, there already exist various iterative eigensolvers in the literature. The simplest eigensolution solvers are the power iteration and the inverse iteration to calculate only one single eigensolution, corresponding to the largest or smallest eigenvalue in terms of the absolute value [6, 126]. In these iterative methods the matrix A or its inverse A^{-1}, if it exists, is successively applied to a random start vector. An extension to these methods is the usage of a preconditioner to speed up the convergence [109, 76, 77, 78, 79].

Another very common eigensolver algorithm is the so-called Krylov-Schur method, as this has been shown to be very robust regarding various matrix properties [143, 144, 69, 121]. It combines a Krylov method for approximating the eigensolutions with a Schur decomposition to orthogonalize the eigensolutions. In Section 4.1.3, we give a more detailed overview on generalized iterative eigenproblem solvers.

In this work, we will integrate AMG without any pre-requisites as a preconditioner to eigensolver strategies. Thus, we can exploit the AMG efficiency for a vast range of applications. As a general case, we consider a generalized eigenproblem

$$Av_k = \lambda_k M v_k \text{ for } k = 1, \ldots, n \tag{4.2}$$

with the matrices $A = \left(a_{ij}\right)_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n}$ and $M = \left(m_{ij}\right)_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n}$ to be sparse, symmetric, positive definite and M-matrices and the eigenvectors $v_k = \left(v_{k,i}\right)_{i=1,\ldots,n} \in \mathbb{R}^n$ and the corresponding eigenvalues $\lambda_k \in \mathbb{R}$ for $k = 1, \ldots, n$. When M equals the identity $\mathbb{1} \in \mathbb{R}^{n \times n}$, the generalized eigenproblem reduces to a so-called plain eigenproblem.

In some applications, for instance with Graph Laplacians, the matrix A is only positive semi-definite. This is then difficult for using AMG, as the theory is only valid for positive definite matrices. But by slightly increasing the diagonal of A, we overcome this issue without any effects for the calculated eigenvectors and the diagonal increase is easily revertible for the eigenvalues by subtraction.

In most established eigensolver algorithms the calculation of the smallest eigenvalue(s) is based on the inverse of A, see Section 4.1 for detailed information. A calculation of the inverse matrix, however, is not exactly possible with reasonable efforts in most cases. Hence, an iterative solution is the best approximation. Where the matrices A and M of the generalized eigenproblem fulfill the application criteria of AMG, it is a good option for solving the sparse linear systems that occur during the linearization of the eigensolution calculation. This is the case in Equation (4.2). However, AMG can not be applied for the matrices A and M independently, but the eigenproblem has to be represented by the same matrix hierarchy. As we want to exploit benefits of the AMG technology to the full eigenproblem and not only invert the matrix A.

Thus, we are starting with the calculation of an initial guess of approximate eigensolutions by exploiting the AMG hierarchy in Section 4.2. With this initial guess, we apply an inverse iteration that uses AMG inside in Section 4.3. The further approximated eigensolutions are finally iterated with a Krylov-Schur method that extensively exploits setup re-usage of AMG by only applying the AMG solution phase in Section 4.4. By integration and consecutively executing of these three algorithms using AMG, a robust and effective eigensolver method is achieved.

These three algorithms and their fine tuning options are evaluated on the Poisson eigenproblem as reference model problem that is introduced in Section 4.1.4. Finally, we apply our algorithms to various real-world examples in Section 4.5.

We assume a symmetric (generalized) eigenproblem as in Equation (4.2). This ensures that we have real-valued eigensolutions and no complex-valued ones. Theoretically, the presented algorithms are also suitable for complex-valued eigensolution. We only restrict

ourselves to real-valued for implementational reasons and we only will be concerned with such in the example Section 4.5.

4.1 Brief Overview of Eigenproblems

4.1.1 Industrial Application Examples of Eigensolutions

Eigenvalues and -vectors are useful and necessary in many application fields. As we are considering the eigensolution calculation with AMG technology, we elaborate two relevant usage modes in this thesis in more details:

- internal application of eigensolutions:
  For very ill-conditioned matrices, it is quite difficult to construct a matrix hierarchy during the setup phase. For such cases, there exist algorithms, e.g., the so-called smoothed aggregation [152, 153, 151], that can improve the matrix hierarchy construction. The quality of these approaches can be improved by using eigenvectors [97].

  Due to increasing computation power the scalability of smoother should be improved. This can be reached by an eigenvalue estimation [1, 8, 31].

- external application of eigensolutions:
  There are many methods that rely on the knowledge of the smallest eigensolution(s). Very well-known application fields are machine learning [65, 16, 57, 47, 2, 38] but also modal analysis in structural engineering and electrodynamics [66].

We are starting with the improvement of the AMG solver for ill-conditioned problems by using approximate eigensolutions. In such cases it is promising to use so-called smooth vectors to enrich the setup process with additional information. The best-suited smooth vectors correspond to the eigenvectors with the smallest eigenvalues. During matrix hierarchy construction via aggregation methods, the hierarchy itself is improved by using these smooth vectors. These are used to transfer the near-kernel vectors on the coarser matrix hierarchy levels and, thus, to be considered in the coarsest level solver [153, 145]. Roughly speaking, the most challenging parts of the problem are shifted to the most robust solution part, the direct coarse-level solver. To use smoothed aggregation with eigenvectors, the eigensolutions are (roughly) approximated by an easily applicable matrix hierarchy construction and the eigenproblem solution algorithms that we present in the following three Sections 4.2 to 4.4. More information about smooth aggregation and results from real-industrial problems are given in Section 4.5.1.

Eigensolutions are also very common to be used in various machine learning algorithms for Big Data Analysis [65, 16, 57, 47, 2, 38]. Depending on the exact problem formulation, these algorithms need to know just a few or many of the smallest or largest eigenvalues

and/or eigenvectors. The aim of machine learning, sometimes also called artifical intelligence or statistical learning, can, e.g., be to find clusters inside the data or to define a lower dimensional representation of the data for easier handling. In both cases, a significant amount of data has to be processed. This is quite often done by analyzing the eigensolutions or doing a projection on the eigenspaces.

The approaches we develop are excellently suited for such algorithms that need only a few of the smallest eigensolutions, where the underlying linear systems fulfill the requirements to use AMG-like methods. This is precisely the case for spectral clustering of Graph Laplacians. Spectral clustering is a method that summarizes the data based on the (smallest) eigensolutions of the matrix representation. It aims to identify patterns or clusters of data points that share similar characteristics or properties. An overview of spectral clustering and various Graph Laplacian definitions is given in [156, 42]. These ideas rely on the basic idea of graph partitioning based on eigenvectors [45] that reduces the dimensionality of the graph by projection onto the smallest eigenspaces. Thus, we apply our eigensolver to various Graph Laplacians in Section 4.5.2. Another evaluation option for data is kernel PCA [65, 129, 130]. This method is a variation of PCA that needs the smallest eigenvalues for further analysis.

In the following, we shortly mention some more use cases that request the knowledge of eigensolutions. An exhaustive evaluation of these fields is beyond the scope of this work. But they are really promising, as we will see.

Another outlook on perspective applications of AMG-based eigensolvers concerns internal parallelization improvements. Originally, AMG is a serial linear solver approach, but due to increasing computational power over the last decade(s), more and more parallel computing gains attention. Due to this highly parallel smoothers are necessary. The performance and quality of these can be improved when a good approximation of the eigenvalues is known [1, 8, 31]. Perspectively, with growing difficulty of the linear solver problems and further increase of the parallelization requirements, an implementation that can approximately calculate the eigensolutions to improve the final solution phase will be required.

Last but not least, a final outlook will be given for Dynamic Mode Decomposition (DMD), which is a dimension reduction method to be applicable on time series of data [132, 131]. DMD enables the detection and extraction of significant patterns that have a serious impact for the measured time series. There exist various possibilities to calculate a DMD - some of them include eigenvectors. A broad overview of DMD and its application is given in [84]. In [75], this method is used in the context of reservoir simulation. There, eigenproblems need to be solved. Thus, a combination of EP-AMG with System-AMG for reservoir simulation [59, 60] could be a promising field of further research.

4.1.2 Theoretical Background for Iterative Eigensolvers

Usually, the Rayleigh-Quotient plays an important role for the theoretical and practical treatments of generalized eigenproblems as in Equation (4.2) [126, 94, 72]. The Rayleigh-Quotient $\mu(w)$ for an arbitrary vector $w \in \mathbb{R}^n$ is defined as

$$\mu(w) := \frac{\langle Aw, w\rangle}{\langle Mw, w\rangle}. \tag{4.3}$$

This precisely maps each eigenvector onto the corresponding eigenvalue.

As the matrices are symmetric in our generalized eigenproblems in Equation (4.2), the eigenvalues are real-valued and, thus, can be ordered in a consecutive way, i.e., $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. For such an ordering, the eigengap $n_{i,j}$ between two eigenvalues i and j with $1 \leq i < j \leq n$ is defined as

$$n_{i,j} := \frac{\lambda_j}{\lambda_i}. \tag{4.4}$$

In [13, 12, 46] the eigengap is used to expand the number of wanted eigensolutions during the iterative solution process to speed up the convergence. In this context, we explain the effect and the relevance of the eigengap in Section 4.3.2.

In the Sections 4.2 to 4.4, we present three iterative eigensolution algorithms that involve AMG. In all of these algorithms, we use the norm $\|\text{res}_k\|_2$ of the residual vector

$$\text{res}_k = A\widetilde{v}_k - \widetilde{\lambda}_k M\widetilde{v}_k \text{ for } k = 1, \ldots, n \tag{4.5}$$

as convergence criterion for approximate eigensolutions $\left(\widetilde{\lambda}_k, \widetilde{v}_k\right)$. The fact that this is a valid criterion goes back to the Bauer-Fike theorem [14].

As the matrix M in Equation (4.2) is symmetric and positive definite, there exists a Cholesky factorization L with $M = LL^T$ and $L \in \mathbb{R}^{n\times n}$. Using this Cholesky factorization, the generalized eigenproblem in (4.2) can be reformulated as

$$L^{-1}AL^{-T}u_k = \lambda_k u_k \text{ for } k = 1, \ldots, n \tag{4.6}$$

with $u_k = L^T v_k$. The matrix $L^{-1}AL^{-T}$ is symmetric and real-valued and, thus, especially hermitian. By this, all preliminaries for Corollary 3.3 in [126] are fulfilled. This corollary states:

4.1 Corollary (Corollary 3.3, [126])
Let $\widetilde{\lambda}$, $\widetilde{u}$ be an approximate eigenpair of a Hermitian matrix A, with $\|\widetilde{u}\|_2 = 1$ and let r be the corresponding residual vector. Then, there exists an eigenvalue λ of A such that

$$|\lambda - \widetilde{\lambda}| \leq \|r\|_2. \tag{4.7}$$

As for the transformation between the generalized eigenproblem to a plain eigenproblem, the eigenvalue λ_k remains the same with the normalization of u_k. Thus, the norm of the residual vector actually even is a good error bound for the approximated eigenvalue $\widetilde{\lambda}_k$.

Furthermore, it is possible to give a bound for the angle between the exact and the approximate eigenvector for plain eigenproblems as Theorem 3.9 in [126] proves. As we look at generalized eigenproblems, we have to take the eigenvector shift $u_k = L^T v_k$ into account. The main statement that the angle is bounded by the norm of the residual vector and some other fixed values remains, tough. For more details see the explanations and proofs in [142], especially Theorem 3.2 therein.

4.1.3 Overview on Iterative Eigensolvers

The simplest way to calculate the largest eigensolution, i.e., the eigenvector to the largest eigenvalue in terms of the absolute value, for a plain eigenproblem is the power iteration [6, 102]. The power iteration starts with a normalized random vector $\widetilde{w}_0 \in \mathbb{R}^n$ with $\|\widetilde{w}_0\| = 1$. The next iteration step is $\widetilde{w}_{j+1} = \frac{A\widetilde{w}_j}{\|A\widetilde{w}_j\|}$. The Rayleigh-Quotient $\mu\left(\widetilde{w}_j\right)$ then converges to the largest eigenvalue λ_n in terms of the absolute value and the vector $\widetilde{w}_k$ to the corresponding eigenvector v_n.

For a plain eigenproblem and A being non-singular, we have the relation

$$Av_k = \lambda_k v_k \Leftrightarrow \frac{1}{\lambda_k} v_k = A^{-1} v_k \text{ for } k = 1, \ldots, n. \tag{4.8}$$

As a result, the smallest eigenvalue in terms of the absolute value for the matrix A is the largest eigenvalue in terms of the absolute value of the inverse matrix A^{-1}. Hence, the inverse iteration calculates the smallest eigensolution (λ_1, v_1) as a power iteration using A^{-1} [6, 161].

In both cases, power iteration and inverse iteration, basically only one eigensolution is calculated. This can be extended similarly for both algorithms. One option is to shift the matrices in the iteration process, i.e. by using the matrices $A - \gamma\mathbb{1}$ or $(A - \gamma\mathbb{1})^{-1}$. With this shift, the largest/smallest eigensolution near to γ is calculated. Other options are summarized as deflation techniques [126] where the rank of the original matrix A is reduced by the unwanted eigensolutions.

When calculating more eigensolutions, an orthogonalization of the approximate eigenvectors will become necessary, as the eigenspaces are orthogonal to each other. The simplest algorithm to achieve this is Gram-Schmidt orthogonalization [17, 73]. But Gram-Schmidt is numerically unstable, as it is quite sensitive to round-off errors. Hence, it leads to a loss of orthogonality [73]. As the loss of orthogonality often occurs only for higher eigenspaces, only calculation a very low number of eigensolutions is possible. The modified Gram-Schmidt orthogonalization overcomes partly this instability problem

[73].

As in most cases more than one eigensolution is necessary, more efficient algorithms than power iteration or inverse iteration are needed. One well-known algorithm is the Arnoldi iteration [4]. Essentially, the Arnoldi iteration is based on a modified Gram-Schmidt algorithm and computes normalized orthogonal base vectors $q_1, \ldots, q_m \in R^n$ of the Krylov space $\mathcal{K}_m(A, w) := span\left(w, Aw, \ldots, A^{m-1}w\right)$ for an initial vector $w \in \mathbb{R}^n$. During this computation, a so-called Hessenberg matrix $H_m \in \mathbb{R}^{m \times m}$ is constructed. In the Krylov-Schur method [69, 121, 143, 144, 82], this Hessenberg matrix is used to solve a smaller eigenproblem based on H_m. That eigenproblem is much smaller than the original one and efficiently treatable by a direct solver. Based on this results of the so-called Ritz problem, the approximate eigensolutions are updated. By using the Krylov space $\mathcal{K}_m\left(A^{-1}, w\right)$ for the Krylov-Schur algorithm, the smallest eigensolutions can be calculated instead of the largest one.

As a direct inversion of matrices in most cases is computational inefficient, an iterative linear solver can be used instead. For sparse matrices, AMG is a suitable choice, when the necessary requirements, cf. Section 2.4, are fulfilled. For the generalized eigenproblems as formulated in Equation (4.2), the requirements for AMG are satisfied. We present the combination of using AMG with inverse iteration in Section 4.3 and with Krylov-Schur in Section 4.4.

In [123, 27, 20], AMG is directly used to solve for the smallest eigensolutions of a sparse linear system, but with a different, much more complex, setup phase of AMG than we use in this thesis. More precisely, the different setup phase approaches for AMG are necessary as AMG is used for different algorithmic parts of the eigensolution approximation.

To be correct, an eigenproblem is a nonlinear problem and, hence, a Full Approximation Scheme (FAS) [28, 26, 67] for AMG can be used to calculate the eigensolution. FAS respects the non-linearity of the problem in the setup phase - in contrast to the description in Section 2.4. But instead of applying the computationally complex FAS, our approach allows to shield the AMG application from the non-linearity. This allows for a simpler, more efficient application of AMG.

We take over the idea of using a Ritz projection for orthogonalization. But adapt it by controlling in which iterations to apply it in. In particular this is not in each iteration as in the references, cf. Section 4.3.1. During the setup phase of AMG, we use a different approach for constructing the Galerkin operators.

Another option for using AMG is the Exact Interpolation Scheme (EIS) [28, 83, 154]. In this case, the interpolation is adapted in every iteration step, such that the current eigenvector approximation is exactly interpolated. This has the advantage that no correction scheme is necessary in the solution phase of AMG. But it comes with additional computational costs, as in every iteration step the interpolation is recalculated. Thus,

no distinction between setup and solution phase is possible.

In this thesis, we use the separation in setup and solution phase to save computational costs by avoiding to recalculate AMG's setup in each step of the presented eigensolution algorithms.

4.1.4 Reference Model Problem: Poisson Eigenproblem

In the following sections, we explain and evaluate inverse iteration, see Section 4.3, and Krylov-Schur method, see Section 4.4, along with some algorithmic fine tuning options. We evaluate the effect of both algorithms and the fine tuning options on the solution process of (generalized) eigenproblems with the Poisson eigenproblem. The Poisson eigenproblem is really grateful for this purpose, as the size of the matrix can easily be scaled and the solutions of the eigenproblem are known analytically, see Table 4.1.

We chose the vibration analysis of a rectangular membrane as a prototypical example. The solution of this physical problem is a basic benchmark problem for eigenproblems and commonly used. The discretization of the rectangle is realized with a uniformly structured rectangular grid. A full mathematical and physical description of this vibration analysis can be found in [43]. In the following, we briefly summarize the necessary information for the following benchmark examples.

The differential equation for the vibration analysis on a rectangular membrane $[0, a] \times [0, b]$ is $\Delta u = u_{tt}$. To solve this differential equation, a splitting approach with $u(x, y, t) = v((x, y))g(t)$ is chosen. Further evaluation of the differential equations leads to the eigenproblem $\Delta v = -\lambda v$. To ensure a unique solution Dirichlet boundary conditions are defined. Then the eigenvalues are given by $\lambda_{n,m} = \pi^2 \left(\frac{n^2}{a^2} + \frac{m^2}{b^2} \right)$ with the corresponding discrete eigenfunctions $v_{n,m}(x, y) = \cos\left(\frac{n\pi x}{a}\right) \cos\left(\frac{m\pi y}{b}\right)$ for $n, m = 1, 2, \ldots$.

For reasons of simplicity, we take the unit square for our benchmarks, i.e., $[0, 1] \times [0, 1]$. Due to the symmetry of the problem, this eigenproblem then inherits the difficulty of multi-dimensional eigenspaces, e.g., $\lambda_{1,2} = \lambda_{2,1}$. In the following benchmarks, we want to calculate the first 9 eigensolutions, as this request combines many difficulties. In Table 4.1, we present the first 13 exact eigenvalues to see the difficulties occurring when only calculating the first 9 eigensolutions. We directly see that the eigenvalues 9 and 10 are identical, thus, we split this eigenspace with our request for 9 eigensolutions. Furthermore, the following eigengaps are really small and, thus, slow convergence could be expected.

We will see that our eigensolver approach can cope with these difficulties.

number of eigenvalue	(n, m)	**eigenvalue**	**eigengap**
1	(1,1)	19.7392...	
2	(1,2)	49.3480...	2.5
3	(2,1)	49.3480...	1
4	(2,2)	78.9568...	1.6
5	(1,3)	98.6960...	1.25
6	(3,1)	98.6960...	1
7	(2,3)	128.3049...	1.3
8	(3,2)	128.3049...	1
9	(4,1)	167.7833...	1.31
10	(1,4)	167.7833...	1
11	(3,3)	177.6529...	1.06
12	(4,2)	197.3921...	1.11
13	(2,4)	197.3921...	1

Table 4.1: The first 13 smallest eigenvalues for the vibration analysis on the unit square with belonging eigengap.

4.2 Initial Guess for the Eigensolutions Using the AMG Hierarchy

The inverse iteration, see Section 4.3, and Krylov-Schur method, see Section 4.4, need to have some approximate eigensolutions to start with. The more accurate these approximate eigensolutions are, the better the algorithms will converge. We want to exploit the matrix hierarchy to calculate such an initial guess of eigensolutions. That is, we only use the setup phase of AMG, cf. Section 2.3. In this setup phase, we neglect the fact that originally we have a non-linear problem to solve. The setup phase interprets A as a linear system to solve. We describe the extension of the setup phase to include A and M, as we mainly look at generalized eigenproblems in Section 4.2.1.

Our initial guess algorithm starts on the coarsest level of the matrix hierarchy, similar to [123, 27]. Its full pseudo code is depicted in Algorithm 4.1. We use the small size of the coarsest matrix hierarchy level to apply a direct solver for the coarsest (generalized) eigenproblem. The way how we formulate the generalized eigenproblem on the coarsest level, based on A and M, will be discussed in Section 4.2.1. For the moment, we take it as given.

For the approximation on the next finer level we start with the interpolation of the eigenvectors (Line 4 in Algorithm 4.1), followed by Gram-Schmidt orthogonalization and normalization (Lines 5 and 6 in Algorithm 4.1). With this result, the Rayleigh-Quotient as an eigenvalue approximation is calculated (Line 7 in Algorithm 4.1) and, based on this, the residuum (Line 8 in Algorithm 4.1). As it is computationally expensive, but

especially regarding orthogonalization, ends in a better approximation, an optional Ritz projection is done adaptively. We will further discuss the optional Ritz projection in Section 4.3.1.

Afterwards, the approximated eigensolution that has been gained on the current level is further relaxed to improve its quality (Lines 10 to 20 in Algorithm 4.1). This process is iteratively repeated until the finest level is reached.

On every level i we check if the user-defined stopping/convergence criterion $\mathscr{C}\left(\widetilde{\lambda}_j^i, \widetilde{v}_j^i\right)$ is reached. This criterion depends on the current approximate eigensolutions $\left(\widetilde{\lambda}_j^i, \widetilde{v}_j^i\right)$ for $j = 1, \ldots, n_{\text{ev}}$ and could be a maximum number of iterations or a requested residual reduction.

The process of successively using the matrix hierarchy is visualized in Figure 4.1, starting on the coarsest matrix hierarchy level. It has the form of a "growing" V-cycle or the second half of an F-cycle [160]. The growing form is transferring the eigensolutions to the next finer matrix hierarchy level, as in Line 4 in Algorithm 4.1. The V-form is resulting from using parts of the matrix hierarchy to solve the defect correction in Line 13 in Algorithm 4.1.

The strength of this procedure is that the initial guess of the eigensolutions $\left(\widetilde{\lambda}_i, \widetilde{v}_i\right)$ for the inverse iteration algorithm, cf. Algorithm 4.2, or the Krylov-Schur algorithm, cf. Algorithm 4.4, is essentially based on a direct solution that is processed with the robust AMG interpolation. Thus, we efficiently provide a rather accurate initial guess, depending on the quality of the interpolation and restriction operators.

We need to discuss the handling of smallest eigenvalues, when they have no representation on the coarsest level. A possible solution approach is to use a rough iterative first approximation on each matrix hierarchy level of the eigenvalues and to compare this with the approximation gained on the coarser level. But this comes with computational overhead costs. However, since we use an AMG hierarchy, we can simply avoid this overhead: AMG constructs its hierarchy such that low error frequencies are represented properly on the coarse levels [23, 137]. These low error frequencies correspond to small eigenvalues of A. Thus, as long as the coarsest level problem size is significantly larger than the number of eigensolutions to be computed, we can safely avoid the overhead.

The initial guess is requested to calculate the first n_{ev} approximate eigensolutions. In Line 9 in Algorithm 4.1, we increase this value to n_{gap}. It is known that the convergence can be improved by calculation more eigensolutions than requested [13, 12, 46]. Especially, the orthogonalization process benefits from this fact. The specific criteria how the eigengap n_{gap} in relation to n_{ev} and a user parameter is defined is explained in Section 4.3.2.

In Line 11 in Algorithm 4.1, we set the variable n_{start} that we use to further control the relaxation of our approximate eigensolution on the current matrix hierarchy level.

Algorithm 4.1: Pseudo Code Initial Guess using AMG hierarchy

Input: A, M: fine-level matrices
(if M not explicitly is given, it's the identity)
n_{ev}: number of requested eigenvalues
$\mathscr{C}\left(\widetilde{\lambda}_j^i, \widetilde{v}_j^i\right)$: stopping/convergence criterion
depending on current approximations $\left(\widetilde{\lambda}_j^i, \widetilde{v}_j^i\right)_{1,\ldots,n_{\text{ev}}}$

Data: $lev_{\max}$: number of created levels
n_{gap}: maximum number of calculated eigenvalues ($n_{\text{ev}} \leq n_{\text{gap}}$)
I_{i+1}^i: Interpolation operator from level $i+1$ to i
A_i, M_i: Galerkin product of matrices A, M on level $i = 2, \ldots, lev_{\max}$
$\widetilde{\lambda}_j^i$: eigenvalue approximation on level i for eigensolution j
$\widetilde{v}_j^i$: eigenvector approximation on level i for eigensolution j
res_j^i: residuum vector on level i for eigensolution j

```
1  for i = lev_max, ..., 1 do
2      if i ≠ lev_max then
3          for j = 1, ..., n_gap do
4              ṽ_j^i = I_{i+1}^i ṽ_j^{i+1}
5              Gram-Schmidt orthogonalization of ṽ_j^i
6              normalization of ṽ_j^i
7              Rayleigh-Quotient λ̃_j^i = ⟨A_i ṽ_j^i, ṽ_j^i⟩ / ⟨M_i ṽ_j^i, ṽ_j^i⟩
8              calculate residuum res_j^i = A_i ṽ_j^i − λ̃_j^i M_i ṽ_j^i
9          set n_gap (eigengap evaluation)
10         while not C(λ̃_j^i, ṽ_j^i) do
11             set n_start (convergence already reached and/or too small eigengap)
12             for j = n_start, ..., n_gap do
13                 solve A_i x = res_j^i
14                 ṽ_j^i = ṽ_j^i + x
15                 Gram-Schmidt orthogonalization of ṽ_j^i
16                 normalization of ṽ_j^i
17                 Rayleigh-Quotient λ̃_j^i = ⟨A_i ṽ_j^i, ṽ_j^i⟩ / ⟨M_i ṽ_j^i, ṽ_j^i⟩
18                 calculate residuum res_j^i = A_i ṽ_j^i − λ̃_j^i M_i ṽ_j^i
19             if do_Ritz then
20                 Ritz-Projection
21     else
22         direct solver for A_{lev_max} ṽ_j^{lev_max} = λ̃_j^{lev_max} M_{lev_max} ṽ_j^{lev_max} for
           j = 1, ..., n_gap
```

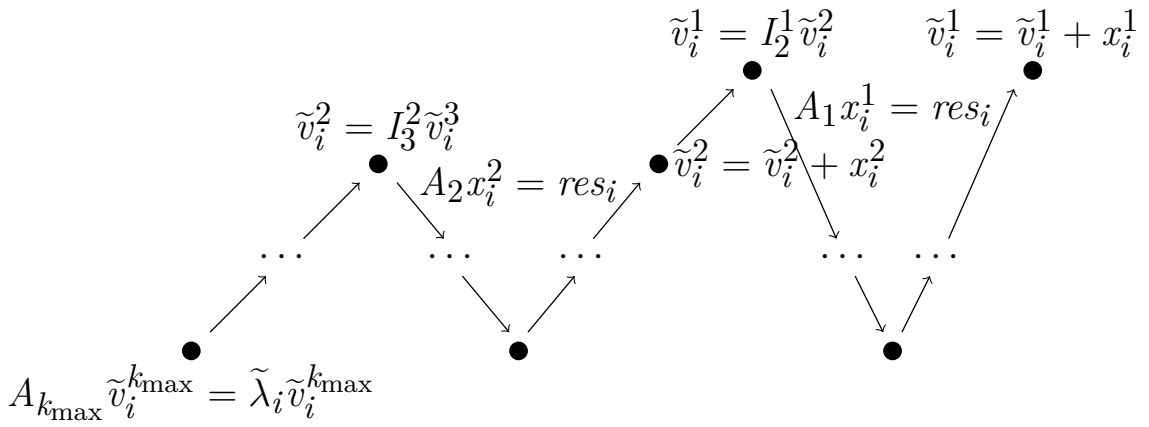


Figure 4.1: Sketch of initial guess algorithm

When eigensolutions already reached the convergence criteria, there is no necessity to further relax them. By neglecting them in the following relaxation process, we can save computational resources. A special focus has to be given to such eigensolutions with a very small eigengap or multiple eigenspaces. Then all of these cases have to be recognized in the following relaxation, as this is completed with the orthogonalization process. We observe that the orthogonalization process is error-prone when the eigensolutions are quite close or span a multidimensional eigenspace.

Within our hierarchical approach, the Rayleigh-Quotient for the eigenvalue approximation is a good choice, as it is invariant when interpolating the eigenvector approximation from a coarse matrix hierarchy level to the next finer matrix hierarchy level. The Rayleigh-Quotient for an interpolated eigenvector approximation $\widetilde{v}_j^i = I_{i+1}^i \widetilde{v}_j^{i+1}$ on matrix hierarchy level i for the j-th eigensolution approximation reads as

$$\begin{aligned}
\widetilde{\lambda}_j^i &= \frac{\langle A_i \widetilde{v}_j^i, \widetilde{v}_j^i \rangle}{\langle M_i \widetilde{v}_j^i, \widetilde{v}_j^i \rangle} \\
&= \frac{\langle A_i I_{i+1}^i \widetilde{v}_j^{i+1}, I_{i+1}^i \widetilde{v}_j^{i+1} \rangle}{\langle M_i I_{i+1}^i \widetilde{v}_j^{i+1}, I_{i+1}^i \widetilde{v}_j^{i+1} \rangle} \\
&= \frac{\langle I_{i+1,t}^i A_i I_{i+1}^i \widetilde{v}_j^{i+1}, \widetilde{v}_j^{i+1} \rangle}{\langle I_{i+1,t}^i M_i I_{i+1}^i \widetilde{v}_j^{i+1}, \widetilde{v}_j^{i+1} \rangle} \\
&= \frac{\langle I_i^{i+1} A_i I_{i+1}^i \widetilde{v}_j^{i+1}, \widetilde{v}_j^{i+1} \rangle}{\langle I_i^{i+1} M_i I_{i+1}^i \widetilde{v}_j^{i+1}, \widetilde{v}_j^{i+1} \rangle} \\
&= \frac{\langle A_{i+1} \widetilde{v}_j^{i+1}, \widetilde{v}_j^{i+1} \rangle}{\langle M_{i+1} \widetilde{v}_j^{i+1}, \widetilde{v}_j^{i+1} \rangle} = \widetilde{\lambda}_j^{i+1}
\end{aligned} \tag{4.9}$$

and, thus, is equal to the Rayleigh-Quotient for the eigenvector approximation on matrix hierarchy level $i+1$. To have the correct relation between the Rayleigh-Quotient

approximation of the eigenvalues on different matrix hierarchy levels, we always have to include the right-hand side matrix. This further emphasizes the previously discussed necessity to use the right-hand side matrix in all eigenproblem cases - plain and generalized. The relation between restriction and interpolation then is also reflected in the denominator of the Rayleigh-Quotient.

The interrelation between the fine- and the coarse-level matrix eigenvector approximation depends on the quality of the matrix hierarchy construction, especially depending on the interpolation I_{i+1}^{i} and restriction I_{i}^{i+1} operators. The finer eigenproblem equation on level i with the j-th eigensolution approximation is related to the coarse-level eigenproblem on level $i+1$ in the following way:

$$
\begin{aligned}
& A_i \widetilde{v}_j^i = \widetilde{\lambda}_j^i M_i \widetilde{v}_j^i \\
\Leftrightarrow & A_i I_{i+1}^i \widetilde{v}_j^{i+1} = \widetilde{\lambda}_j^i M_i \widetilde{v}_j^i \\
\Rightarrow & I_i^{i+1} A_i I_{i+1}^i \widetilde{v}_j^{i+1} = \widetilde{\lambda}_j^i I_i^{i+1} M_i \widetilde{v}_j^i \qquad (4.10) \\
\Leftrightarrow & A_{i+1} \widetilde{v}_j^{i+1} = \widetilde{\lambda}_j^i I_i^{i+1} M_i \widetilde{v}_j^i \\
\Leftrightarrow & A_{i+1} \widetilde{v}_j^{i+1} = \widetilde{\lambda}_j^i I_i^{i+1} M_i I_{i+1}^i \widetilde{v}_j^{i+1} \\
\Leftrightarrow & A_{i+1} \widetilde{v}_j^{i+1} = \widetilde{\lambda}_j^{i+1} M_{i+1} \widetilde{v}_j^{i+1}. \qquad (4.11)
\end{aligned}
$$

The suitability of the eigenvectors from the coarser eigenproblem to the finer eigenproblem is related to the quality of the interpolation and restriction operators. As the left-scaling with I_i^{i+1} in Equation (4.10) is not an equivalent transformation, the reversibility/"equivalence" of the relation between the eigenvalue problem on different matrix hierarchy levels depends on the quality of the interpolation and restriction operator, especially on the reversibility of them, i.e., how well do $I_i^{i+1} I_{i+1}^i \in \mathbb{R}^{n_{i+1} \times n_{i+1}}$ and $I_{i+1}^i I_i^{i+1} \in \mathbb{R}^{n_i \times n_i}$ approximate the identity. The last equivalent transformation for Equation (4.11) uses the invariance of the Rayleigh-Quotient for matrix hierarchy levels, as proven in Equation (4.9).

Figure 4.2 shows the effects of Equation (4.11) for a 2D Poisson problem as prototypical eigenproblem. In Section 4.1.4, we have introduced the Poisson eigenproblem and its analytical solutions. Additionally, the Poisson eigenproblem fulfills all requirements for setting up an optimal AMG hierarchy. Thus, the solution of the coarser level eigenproblem(s) is quite well related to the fine-level eigenproblem and the analytical solution. In Figure 4.2, we see the relative error between the eigenvalue on the matrix hierarchy levels and the analytical eigenvalue. As expected, the error is decreasing for finer matrix hierarchy levels, as the error in setting up the hierarchical eigenproblems decreases.

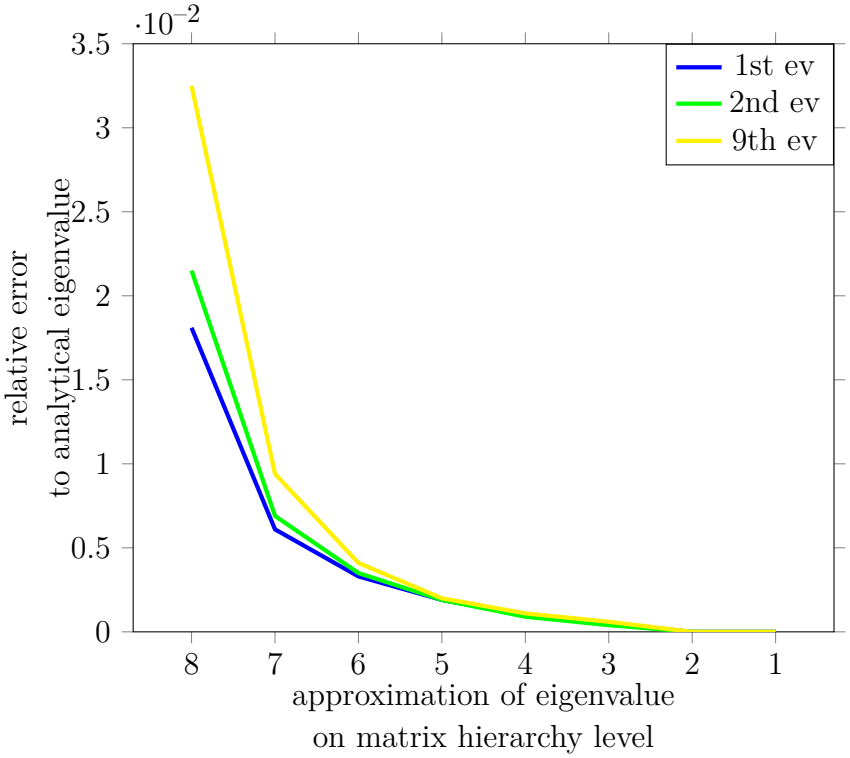


Figure 4.2: Relative error of the eigenvalue approximation to the analytical solution for the complete matrix hierarchy for a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048.

4.2.1 Extension of AMG Hierarchy for Generalized Eigenproblems

It is more common to solve a generalized eigenproblem as in Equation (4.2). Thus, we have to modify the setup of the AMG hierarchy, as it normally is only constructed for a single matrix A.

In our first approximation algorithm we set up a generalized eigenproblem on each matrix hierarchy level. The matrix hierarchy and the interpolation and restriction operators are still calculated based on the matrix A. Additionally, with the same operators, the Galerkin product for the right-hand side matrix M is calculated to obtain the generalized eigenproblem on the full matrix hierarchy. This means, for a two-level scheme, the coarse-level operator is defined as $M_c := I_f^c M I_c^f$. Because then, when v fulfills Equation (4.2), $v^c = I_c^f v$ fulfills Equation (4.2) left-scaled with I_f^c, and A_c can remain a classical Galerkin product, so that classical AMG ideas carry over.

For plain eigenproblems, different experiments have shown that using the Galerkin operators of the identity on coarser matrix hierarchy levels improves the first approximation of the eigensolutions. This coincides with our above considerations for generalized eigenproblems. Due to this fact, in all presented algorithms, we use the generalized eigenproblem formulation and set the right-hand side matrix M to identity $\mathbb{1}$, where no right-hand side matrix is explicitly given.

The matrix structure for the matrices A and M may be quite different. Thus, building the matrix hierarchy for AMG only on the matrix A may appear questionable: Important aspects from matrix M for the generalized eigenproblem formulation may be lost on coarser matrix hierarchy levels. But for generalized eigenproblems arising from physical applications, e.g., vibrations, the matrix structure of A and M typically is closely related. In addition, the coarse-level transfer of the eigenproblem is algebraically correct in either case. The question is only whether the AMG hierarchy could be enriched by information from matrix M. The aspects of constructing a matrix hierarchy based on both matrices is an aspect of future research. In plain eigenproblems from data analysis this do not matter.

The potentially different non-zero patterns of A and M need to be considered in the implementation of the Galerkin product. Our implementation is aware of this fact and still exploits synergies in data structures.

4.2.2 Using Eigenvalue Shift for Coarse-Level Operators

In the literature of eigenproblem solvers, it is well known that the calculation range of eigenvalues can be adapted by shifting the matrix [6]. The previously described first approximation works the best for near-zero eigenvalues. Additionally, this holds for the algorithms described in Sections 4.3 and 4.4. The eigenproblem (A, M) in Equation

(4.2) can be reformulated as follows:

$$\hat{A}v_k := (A - \omega M)\, v_k = (\lambda_k - \omega)\, Mv_k =: \hat{\lambda} Mv_k. \tag{4.12}$$

The eigenproblem $\left(\hat{A}, M\right)$ has the same eigenvectors v_k as the original eigenproblem (4.2), but with the shifted eigenvalues $\hat{\lambda} = \lambda - \omega$. The first approximation for this eigenproblem also works the best for near-zero eigenvalues. But these near-zero eigenvalues are now related to eigenvalues λ near ω of the original eigenproblem (A, M). When applying AMG to $\hat{A}$, the shift ω has to be chosen careful, as the matrix $\hat{A}$ can become singular if ω equals an eigenvalue λ_k of the matrix A.

It is well possible that we can fasten up our eigensolution calculation when we do such an eigenvalue shift for higher eigenvalues. We can extend the eigenvalue shift to all matrix hierarchy levels, instead of using it only at the finest one. Just not on the coarsest one, as we use a direct solver there. During the loop over all calculated not-converged eigensolutions (Lines 12 to 18 in Algorithm 4.1), we have the opportunity to shift the matrix by the current eigenvalue approximation. But,we have to remind that we are also using the matrix for a linear solution inside the approximation process (Line 13 in Algorithm 4.1). Hence, by this shift, the matrix on the current matrix hierarchy level can become near-singular. Thus, the induced linear system may be hard to solve.

The objective of this algorithmic switch is a trade-off between convergence speed and solution time per iteration: A matrix shift, i.e., $A_i \Rightarrow A_i - \lambda_k M_i$, on each matrix hierarchy level i for each calculated eigenvalue k increases the computational time. But it is expected to lead to a better convergence.

When testing various formulations of the Poisson eigenproblem (discretization size, number of requested eigensolutions), we observe a very low influence on improving the eigenvalue convergence, but a drastic solution time increase due to the additional matrix operations. Thus, we decided to neglect this algorithmic improvement in the following.

By comparing the solution time of the eigenproblem with shifted and unshifted matrix operators, we should have in mind that the implementation for the shifting and re-shifting of the various matrices on the constructed levels may exploit better synergies. But as we do not observe a significant improvement in the number of iterations, an efficient implementation would not change the decision to neglect this algorithmic variation.

To improve the overhead costs by the eigenvalue shift, it is considerable not to apply the eigenvalue shift for every eigenvalue itself, but for a "significant" eigenvalue of higher order. Maybe it is useful to combine such a decision criterion on which eigenvalue should be used to shift the matrix with the evaluation of eigengaps, see Section 4.3.2. This variation is not further followed in this work, but may be investigated additionally

in the context of more ill-conditioned problems.

Currently, our algorithms (including inverse iteration in Section 4.3 and Krylov-Schur method in Section 4.4) are only possible to calculate eigensolutions in increasing order away from 0. By a general matrix shift before the AMG hierachy is constructed, i.e. the setup-phase of AMG is done for $\hat{A} := A - \omega M$, all of the EP-AMG algorithms presented in this thesis can be used to calculate eigensolutions with eigenvalues λ near ω in terms of the absolute value. The implementation of this algorithmic feature is beyond the scope of this work.

4.3 Inverse Iteration Using AMG as Preconditioner

As shortly explained in Section 4.1, the inverse iteration is the simplest iterative eigensolver for the smallest eigensolution. The eigenproblem in Equation 4.2 fulfills all requirements to use AMG instead of a direct inversion of A. Additionally, we enrich the inverse iteration with the concept of defect correction on a given matrix hierarchy. Contrary to the basic inverse iteration, we want to calculate more than one smallest eigensolution.

Additionally, we add some modifications to standard inverse iteration for a better performance.

- As Gram-Schmidt orthogonalization is numerically unstable regarding orthogonalization. We extend the inverse iteration by a Ritz projection to provide stabilization, cf. Line 12 in Algorithm 4.2. We further explain the Ritz-projection in Section 4.3.1.

- For improving the overall convergence behavior, an eigengap turns out to be useful, cf. Line 1 in Algorithm 4.2. In Section 4.3.2, we explain the criterion for calculating this eigengap.

The effect of both modifications is evaluated using the previously introduced Poisson eigenproblem as reference model problem, see Section 4.1.4.

In the following, we give an overview over our inverse iteration using AMG in combination with an additional preconditioner and defect correction, shortly described in Algorithm 4.2. We starts with the eigensolutions by our initial guess algorithm, described in the previous Section 4.2.

During inverse iteration, we want to calculate n_{ev} eigensolutions, whereby the termination criteria are the required residuum for each eigenpair and the maximum number of iterations. The number n_{ev} of required eigenpairs can be increased to $n_{\text{ev}} \leq n_{\text{gap}}$ to have better computational performance by using a better suitable eigengap, see Section 4.3.2 for more details. The inverse iteration is executed until a user-defined

stopping/convergence criterion $\mathscr{C}\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$ is reached. This criterion depends on the current approximate eigensolution $\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$ for $j = 1, \ldots, n_{\text{ev}}$ and could be a maximum number of iterations or a requested residual reduction.

Instead of simply using AMG to exchange the inverse of A in the inverse iteration algorithm, we want to fully benefit from the AMG hierarchy with smoothing different error components on various matrix hierarchy levels. Thus, we use a defect correction regarding the current residual of eigensolution j in the main part of the method, cf. Lines 5 and 6 in Algorithm 4.2. This is similar to the idea of defect correction in Section 2.4. The correction linear system $Ax = \text{res}_j$ for a fixed eigenpair j is solved via GMRES [127] and uses AMG as a preconditioner inside, cf. Lines 5 and 6 in Algorithm 4.2. We choose GMRES as it is a robust method for solving linear systems. Of course, other methods such as Bi-CG [52, 85], CG [71] or GCR [49, 157] can be used analogously. With the solution x, the previous approximation of the eigenvector $\widetilde{v}_j$ is corrected.

Afterwards, the updated eigenvector approximation $\widetilde{v}_j$ is orthogonalized and normalized. Then, an updated eigenvalue approximation $\widetilde{\lambda}_j$ via Rayleigh-Quotient is calculated, cf. Lines 7 to 9 in Algorithm 4.2. This allows for the merely linear application of AMG, as we have seen in Section 4.2. When all eigenvectors are updated, optionally a Ritz-Projection is done to have a full orthogonalization of the required eigenspace, cf. Lines 11 and 12 in Algorithm 4.2. We explain the Ritz-Projection in detail and why this is optional, respectively only done every few iteration steps, in Section 4.3.1.

The variable n_{start} in Line 3 of Algorithm 4.2 is introduced to save computational time, similar as for the first approximation in Section 4.2. When an eigensolution is converged, the iteration starts with the next unconverged solution. An already converged eigensolution is only taken into account when the eigengap to the unconverged eigensolution is too small. Because then, the orthogonalization in the multidimensional eigenspace is stabilized. Already converged eigensolutions are only slightly improved by further iterations, in contrast to the drastically higher solution time this accuracy gain needs.

4.3.1 Using a Parameter Controlled Ritz Projection

The Ritz projection is one of the most computationally expensive parts of Algorithm 4.2. But it ensures good orthogonalization results for the complete eigenspace. The complete mathematical background and explanations can, for instance, be found in [159] or [126]. In [27, 123], Ritz projection already is used for the eigenspace orthogonalization in an AMG algorithm. But they only use it at the finest level due to the expensive costs of a Ritz projection. As we introduce a parameter controlled Ritz projection, the execution of a Ritz projection on every matrix hierarchy level is possible without enormous computational costs. Simply because the execution frequency is controllable.

To save computational costs in the overall algorithm, it is possible to not apply the Ritz projection in every iteration step. The effect of the Ritz projection is to enable faster

Algorithm 4.2: Pseudo Code Inverse Iteration

Input: A, M: fine-level matrices
(if M not explicitly is given, it's the identity)
n_{ev}: number of requested eigenvalues
$\mathscr{C}\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$: stopping/convergence criterion
depending on current approximations $\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)_{1,\ldots,n_{\text{ev}}}$

Data: n_{gap}: maximum number of calculated eigenvalues ($n_{\text{ev}} \leq n_{\text{gap}}$)
$\widetilde{\lambda}_j$: eigenvalue approximation for eigensolution j
$\widetilde{v}_j$: eigenvector approximation for eigensolution j
($\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$ initial guess using the AMG hierarchy as in Section 4.2)
res_j: residuum vector for eigensolution j

```
1  set n_gap (possible eigengap evaluation, see Section 4.3.2)
2  while not C(λ̃_j^1, ṽ_j^1) do
3      set n_start (convergence already reached and/or too small eigengap)
4      for j = n_start, ..., n_gap do
5          solving A x = res_j with GMRES and AMG as preconditioner
6          ṽ_j = ṽ_j + x
7          Gram-Schmidt orthogonalization of ṽ_j
8          Normalization of ṽ_j
9          Rayleigh-Quotient λ̃_j = ⟨A ṽ_j, ṽ_j⟩ / ⟨M ṽ_j, ṽ_j⟩
10         calculate residuum res_i = A ṽ_j − λ̃_j M ṽ_j
11     if do_Ritz then
12         Ritz projection
```

convergence and improved orthogonalization of the eigenvectors that would not be achieved by the simpler Gram-Schmidt orthogonalization. Furthermore, Ritz projection is not as sensitive for round-off errors with ill-conditioned problems as Gram-Schmidt orthogonalization is. The key-objective is to find a balance between convergence improvement and computational costs. Hence, the Ritz projection is not necessary in every iteration, but cannot be suspended in too many iterations.

Another potential for algorithmic variation is the consideration of eigenvectors for the Ritz projection. Three scenarios are possible:

- all eigenpairs are included in and updated by the Ritz projection,

  or

- all eigenpairs are included in, but only the non-converged eigenpairs are updated by the Ritz projection,

  or

- only the non-converged eigenpairs are included in and updated by the Ritz projection.

We tested all three possible execution scenarios for the Ritz projection. We have found the best balance between solution time and convergence rate for the second scenario. Thus, we use only this implementation for further testing of the influence of the Ritz projection interval in the following and the extended practical example tests in Section 4.5.

In the following, we evaluate the effect for the convergence rate and the overall runtime when varying the number of iterations after which the Ritz projection is performed. As evaluation example we chose the Poisson eigenproblem as reference model problem, see Section 4.1.4. But we should have in mind that the Poisson eigenproblem is a well-conditioned problem. Thus, it is not heavily, but still observably, affected by round-off effects during the orthogonalization process. For this reason, the necessity of doing a Ritz projection will increase further with more ill-conditioned problems. For the results of practical examples in Section 4.5, we use an appropriate frequency of Ritz projection in relation to the condition of the eigenproblem.

Now we explain the results for the Poisson eigenproblem and why choosing a Ritz projection interval of every five iterations is a good default value. In Figure 4.3, we show the impact of the application interval of the Ritz projection for the total number of iterations for the whole initial guess algorithm. We evaluate the total number of iterations for the first 9 eigensolutions. The number of iterations per matrix hierarchy level is restricted to 500. Convergence for an eigensolution is assumed when the residual gains eight orders of magnitude. We observe that a smaller Ritz projection

interval is better for the convergence rate. This effect is growing for higher eigenvalues, as the orthogonalization space for these is getting smaller. To counteract the small orthogonalization space for higher eigensolutions is possible by using an eigengap. This will increase the numerical calculation space, see Section 4.3.2 for a detailed explanation. When the Ritz projection interval is chosen too high, also the convergence rate of smaller eigensolutions decreases, as seen for the Ritz projection interval of 50.

In Figure 4.4, we show the number of iterations in the inverse iteration for a varying Ritz projection interval. For the very first eigensolutions, in this case for the first three, we observe no difference in the convergence behavior. For the eigensolution later on, we observe worse convergence. Especially for higher eigensolutions, e.g., the 9th eigensolution is most sensitive to a growing Ritz projection interval. This is explainable, as the 9th eigensolution should be orthogonal to all previous eigenspaces. But is only rarely orthogonalized to all eigenspaces, as for higher eigensolution the Gram-Schmidt orthogonalization is really error-prone.

As the Ritz projection every time includes the solution of a small eigenproblem, it is computational expensive. Thus, we have to find the balance between a good convergence rate, which we depict in Figures 4.3 and 4.4, and an efficient computational time. Based on the runtime results in Table 4.2, we figured out that doing a Ritz projection every 5 iterations is a reasonable compromise between convergence rate and computational time.

When evaluating a Ritz projection interval of more than 10 iterations, we observed that the probability of wrong eigenpairs is significantly growing when the Ritz projection interval is greater than the number of iterations for the initial guess. Thus, we need to ensure that during the initial guess computation a Ritz projection is done on each level of the matrix hierarchy, before interpolating the results to the next level. This ensures that the eigenspaces are orthogonalized in any case.

Algorithm 4.3: Pseudo Code Ritz Projection

Input: A, M: fine-level matrices
(if M not explicitly is given, it's the identity)
$\widetilde{\lambda}_j$: eigenvalue approximation for eigensolution j
$\widetilde{\lambda} = \left(\widetilde{\lambda}_j\right)$: vector of eigenvalue approximations
$\widetilde{v}_j$: eigenvector approximation for eigensolution j
$\widetilde{v} = \left(\widetilde{v}_j\right)$: matrix of eigenvector approximations

Data: n_{gap}: maximum number of calculated eigenvalues ($n \leq n_{\text{gap}}$)

```
1 for i = 1, ..., n_gap do
2     for j = 1, ..., n_gap do
3         calculate A_proj(i, j) = ⟨ṽ_i, Aṽ_j⟩
4         calculate B_proj(i, j) = ⟨ṽ_i, Bṽ_j⟩
5 solve A_proj v_proj = λ_proj B_proj v_proj
6 λ̃ = λ_proj
7 v = v · v_proj
```

Ritz projection interval	**runtime [sec]**
1	1098,52
2	711,02
3	589,47
5	492,27
10	657,33

Table 4.2: Complete runtime for the inverse iteration, including an initial guess over the matrix hierarchy with 10 iteration steps per level, with various Ritz projection intervals. The eigenproblem is a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048.

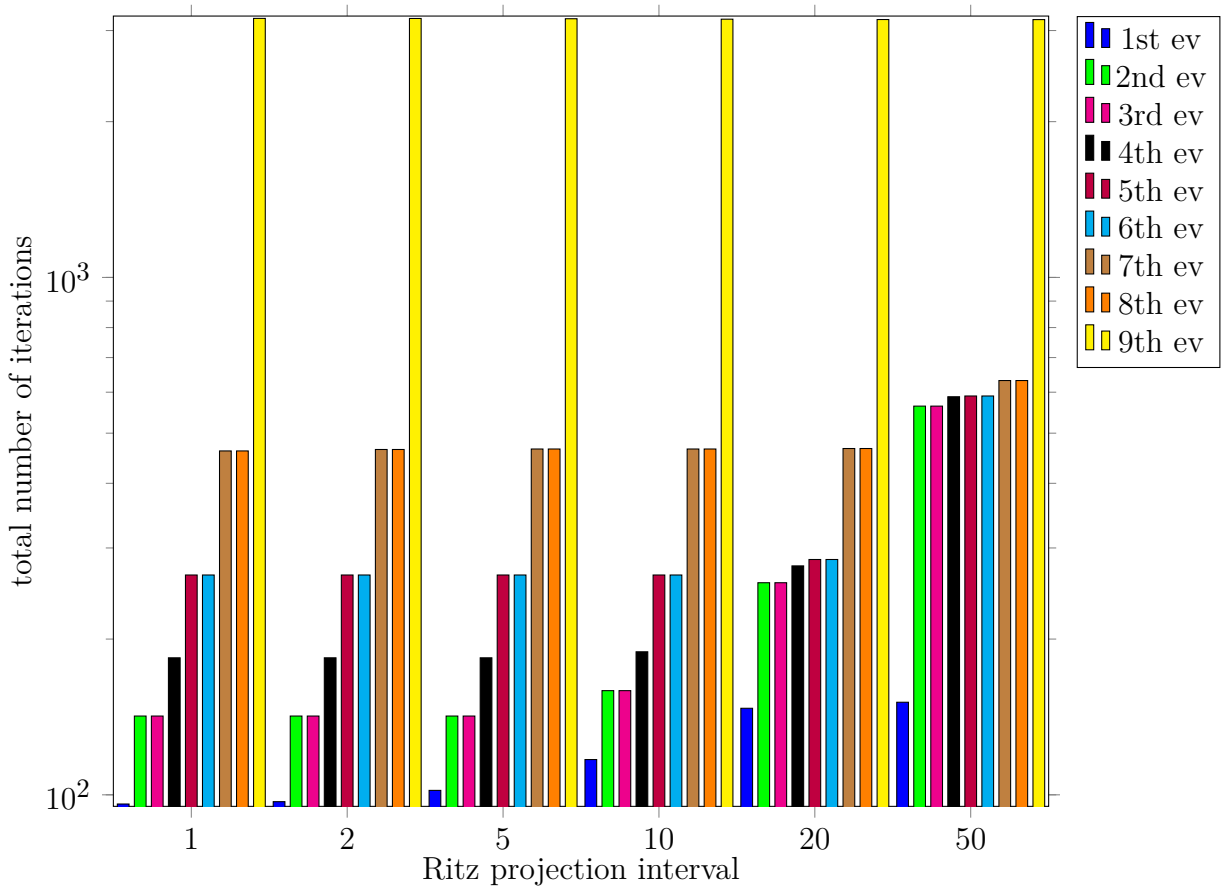


Figure 4.3: Total number of iterations in the entire initial guess algorithm including the iterations on every matrix hierarchy level for different intervals of Ritz projection with a maximum iteration number of 200 for the initial guess on every matrix hierarchy level. The eigenproblem is a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048.

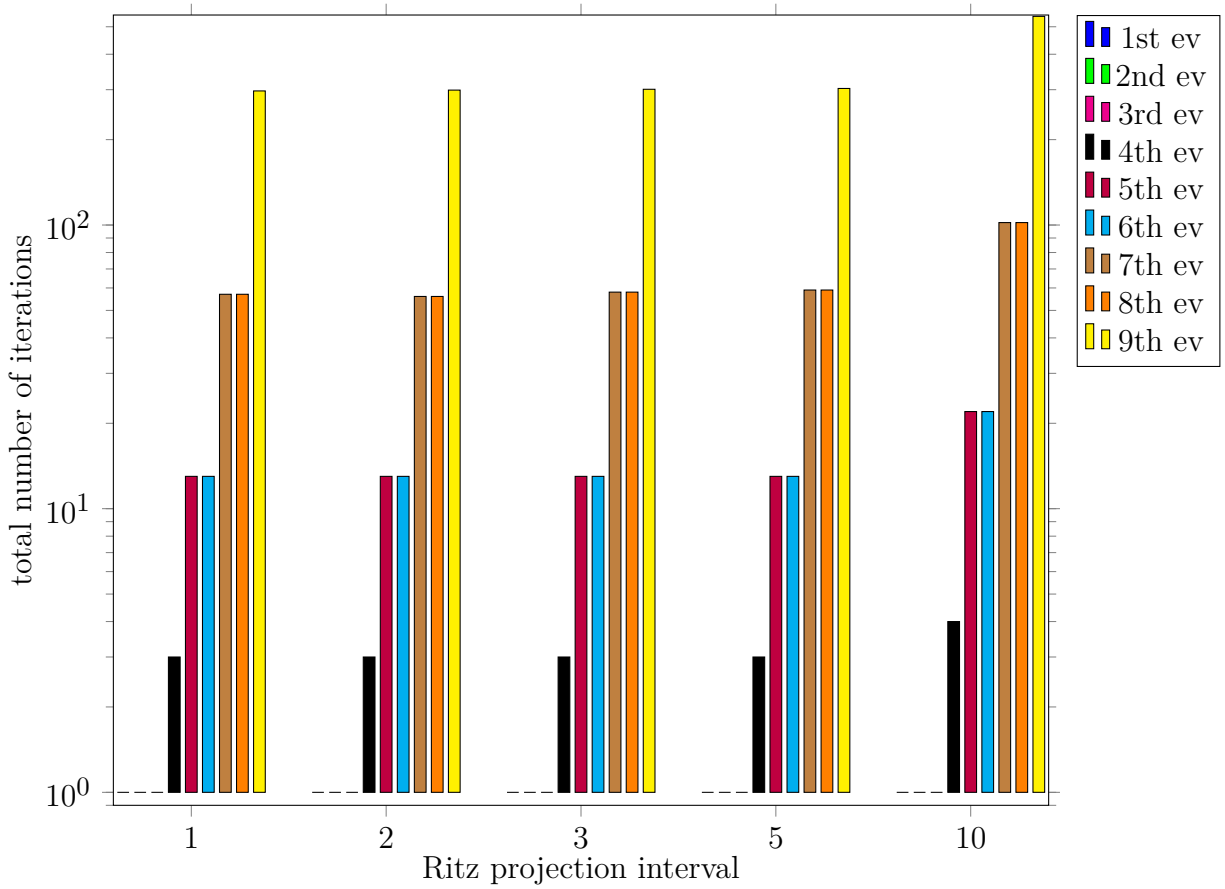


Figure 4.4: Total number of iterations in the entire inverse iteration without the iterations for the initial guess for different intervals of Ritz projection with a maximum iteration number of 5000 for the inverse iteration and 10 iterations for the initial guess on every matrix hierarchy level. The eigenproblem is a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048. For the convergence of the first three eigensolution, we see no difference depending on the Ritz projection interval. The first eigensolution is already converged after the initial guess. The eigenspace spanned by the second and third eigensolutions is converged after one step of inverse iteration.

4.3.2 Using an Eigengap to Improve the Convergence Behavior

The eigengap analyzes the relative difference between two neighbored eigensolutions[1], see the definition in Section 4.1.2. By calculation up to a significant eigengap n_{gap}, which is greater than the required number of eigenpairs n_{ev}, i.e., $n_{\text{ev}} < n_{\text{gap}}$, the runtime can be fastened up. The eigensolution algorithms then run for all eigenpairs up to the eigengap n_{gap}. But the convergence is only checked for the requested number of eigenpairs n_{ev}. In the following, we will see that the effect of using a greater eigenspace is especially noticeable during the orthogonalization process, as this is stabilized.

In [13, 12], already first impacts of choosing higher subspace dimension than requested eigenvalues are mentioned. Further analysis of the influence on computational costs and runtime, beside other factors, can be found for subspace iterations in [46].

During the initial guess algorithm, the eigengap has to be recalculated on each matrix hierarchy level, as the relation of the eigenvalues may be changed by the interpolation and scaling effects on the eigenproblem solutions. The eigenvalue approximations via Rayleigh-Quotient, see Section 4.1.2, is not similar on every matrix hierarchy level, as the eigenvector approximations do change. The value of the eigengap n_{gap} may slightly change on each level of the matrix hierarchy. But on the coarsest matrix hierarchy level, already a first hint for a reliable eigengap can be seen. To be on the safe side, we can save the entire requested eigenspace solutions - including the eigengaps - on all matrix hierarchy levels. And then recheck the position of the eigengap on each matrix hierarchy level individually. Thus, we can benefit from the runtime optimization on every matrix hierarchy level. This especially holds when we have many matrix hierarchy levels. But this comes at the price of more memory, as more full eigensolution approximations are saved. Furthermore, on each matrix hierarchy level, the initial guess for the complete requested eigenspace is calculated. Until a decision for the setting of the eigengap on the finest matrix hierarchy level is defined.

Before the inverse iteration starts, the eigengap is calculated based on the results of the initial guess approximations. The eigengap calculation includes two parts - a relative and an absolute part. The upper bound for the eigengap is given as a user parameter and defines the upper number n_s for searching such an eigengap. As default, we have set $n_s = n_{\text{ev}} + 10$. We check whether the quotient of two neighbored eigenvalues λ_i and λ_{i+1} is greater than 1.10, i.e., $\frac{\lambda_{i+1}}{\lambda_i} > 1.10$ for $n_{\text{ev}} \leq i < n_s$. When this eigenvalue relation is fulfilled, the eigengap is set to $i+1$, i.e., $n_{\text{ev}} \leq n_{\text{gap}} = i+1 \leq n_s$. And the inverse iteration is done up to the eigenvalue n_{gap}. As it is possible that a difference of 10% between two neighbored eigenpairs is not reached, we add an additional relative check. For the relative check, the eigengap is set to the index of the eigenvalue that has the largest eigenquotient, i.e., $n_{\text{gap}} = \arg\max_{n_{\text{ev}} < i \leq n_s} \frac{\lambda_i}{\lambda_{i-1}}$.

[1] For real-valued eigensolutions, just a consecutive ordering of the eigenvalues. For complex-valued eigensolutions, it would use the absolute value.

For testing purposes, we chose the calculation of 9 eigenpairs of a discretized Poisson eigenproblem, see Section 4.1.4, as we then have all relevant effects included:

- We split an eigenspace, as the eigenvalues 9 and 10 are equal.
- For the following eigenvalues, we hit the absolute eigenvalue gap between the eigenvalues 11 and 12.

For having a complete benchmark, we also include higher eigengap requests. We will see that these fall back to 12. In Table 4.3, we see the computation time for reaching 9 converged eigensolutions with different eigengaps. As intended, it pays off to calculate more eigenvalues. The orthogonalization process then is more stable and efficient.

To set the requested eigengap n_s drastically higher than necessary is better than even slightly too small. This is due to a reset of the used eigengap n_{gap} with the previously described criterion. By this, the only overhead in runtime is the higher eigenspace for the initial guess algorithm. Thus, it is better to choose the number of requested eigensolutions higher in most cases, as the convergence improvements overcome the runtime losses. This also holds with general cases, as we always apply the bounds for n_{gap}.

For the Poisson eigenproblem we already see the necessity that the requested eigengap n_s is not too small. To fully profit from the automatic detection of the eigengap n_{gap} the evaluation interval between requested eigensolutions n_{ev} and eigengap n_s should be sufficient. If not other stated the default is $n_s = n_{\text{ev}} + 10$ but can be changed as a user parameter.

In the particular case for the Poisson eigenproblem, requesting 10 eigensolutions is even worse than the minimal number of 9: The iteration number per eigensolution is equal or lower for higher eigensolutions than for the first ones, as Figure 4.5 shows. For the first four eigenpairs, we see no difference in the convergence behavior, i.e., in the necessary number of iterations until the requested residuum is reached. For the convergence of the 9th eigenpair, we observe the most significant effect of using an eigengap. For a higher eigengap (11 or higher in this case), we have the best convergence behavior. This coincides best with the eigengap and no computational overhead is done. Thus, the orthogonalization process is very effective.

requested eigengap n_s	**calculated eigengap** $n_{\mathbf{gap}}$	**runtime [sec]**
9	9	462,297
10	10	543,031
11	11	352,125
12	11	403,812
14	11	409,422
19	11	438,844

Table 4.3: Complete runtime for the inverse iteration, including an initial guess over the matrix hierarchy, with various requested eigengaps n_s to calculate 9 eigenpairs using a Ritz projection every 5 iterations. The eigenproblem is a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048.

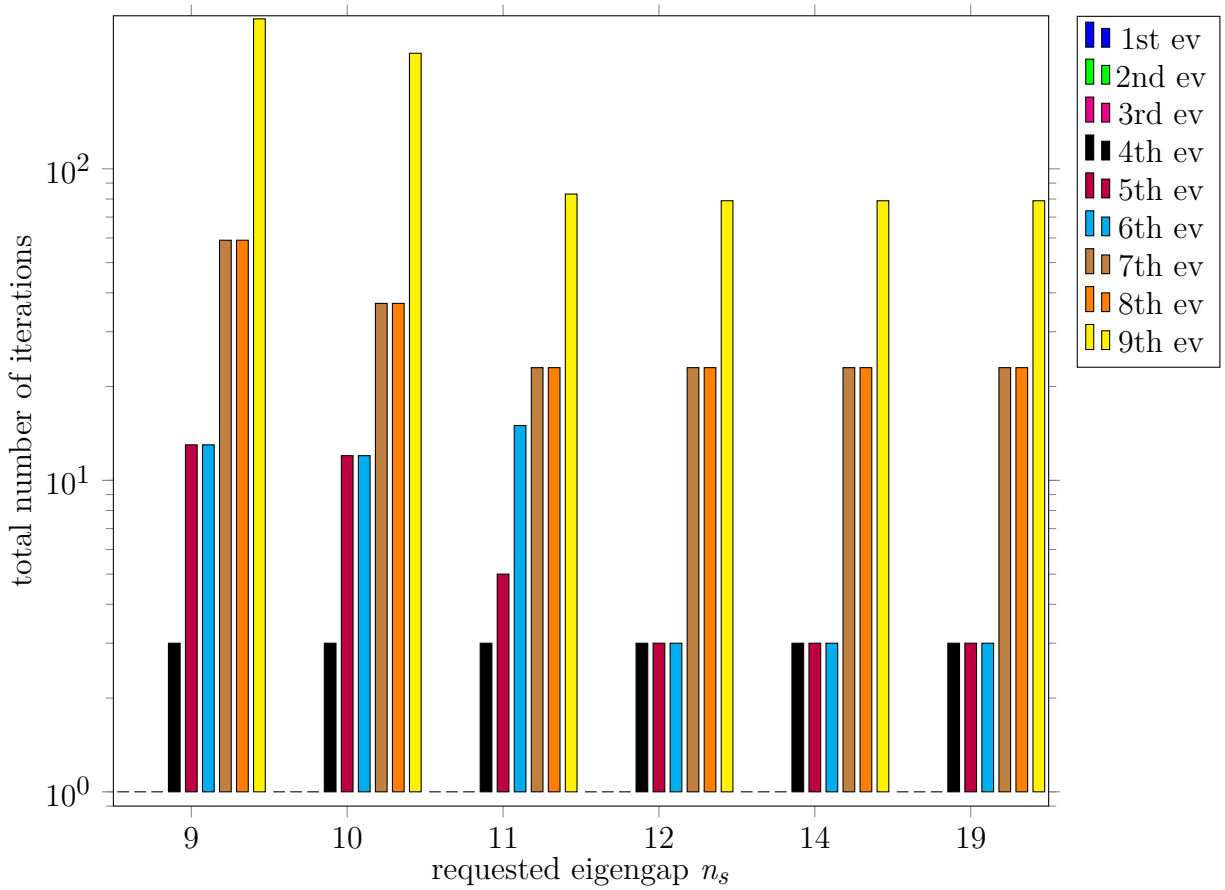


Figure 4.5: Total number of iterations in the entire inverse iteration without the iterations for the initial guess with a Ritz projection every 5 iterations and various requested eigengaps n_s and 10 iterations for the initial guess algorithm on every matrix hierarchy level. The eigenproblem is a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048.

4.4 Krylov-Schur Method Using AMG as Preconditioner

For more challenging situations, the previously presented initial guess algorithm along with the inverse iteration are not sufficient. Such cases are higher eigensolutions or eigensolutions in a nearly-related range. But this also holds for ill-conditioned matrices in the (generalized) eigenproblems. Therefore, we implemented the Krylov-Schur method in combination with AMG. The Krylov-Schur method is used as final iterative algorithm stage for the current state of work, as it is the most computational expensive algorithm of all presented algorithms in this thesis.

The Krylov-Schur Algorithm 4.4 is used to iterate the approximation gained from the initial guess algorithm in Section 4.2 or from the inverse iteration described in Section 4.3, up to the requested accuracy. The idea behind this can be found in [69, 121, 143, 144, 82]. The Krylov-Schur method itself works independently of the previously described initial guess and inverse iteration. But the overall convergence can be fastened up by a more accurate start approximation. Additionally, we reuse the efficient AMG setup a third time for the solution of recurring linear systems inside the Krylov-Schur algorithm.

A sophisticated saving and evaluation inside our Krylov-Schur implementation increases the recycling potential of the constructed Krylov space, especially for higher approximated eigensolutions by the initial guess or inverse iteration. Thus, a reduction of the number of iterations and solution time is possible. The ability of the reusability of the results of the initial guess and the simultaneous calculation of more than one eigensolution in the same setup of the Krylov space has further optimization potential.

For non-generalized eigenproblems, this approach is working quite well. For generalized eigenproblems, the convergence behavior can be stabilized, especially when calculating higher eigensolutions. As a good starting point for further improvements and stabilization we see the interplay of the matrices A and M, particularly for the matrix hierarchy construction.

The Krylov-Schur algorithm calculates one eigensolution per iteration step. As a by-product, also higher eigensolutions can be further iterated. But this should not be relied on. It is rather a beneficial side effect to an unpredictable extent. The algorithm does not rely on this, though. But it checks for this effect to avoid doubled computation.

The Krylov-Schur algorithm is executed until a user-defined stopping/convergence criterion $\mathscr{C}\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$ is reached. This criterion depends on the current approximate eigensolution $\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$ for $j = 1, \ldots, n_{\text{ev}}$ and could be a maximum number of iterations or a requested residual reduction.

The algorithm starts with a Arnoldi decomposition, cf. Line 3 in Algorithm 4.4. This decomposition takes the previously calculated eigenspace approximation into account:

Whenever a new Krylov space for an eigensolution is set up, we start with the previously calculated approximation. Additionally, the linear system inside the Arnoldi decomposition is solved via one GMRES iteration with AMG as a preconditioner, where we reuse our constructed matrix hierarchy.

Already during the Arnoldi decomposition, the following Ritz problem is set up. As this is a really small-sized eigenproblem (subspace dimension of the Krylov space), we use the direct solver *hseqr* from LAPACK [3], cf. Line 14. Based on the Ritz problem results, the currently calculated eigenspace is updated and the marker n_{start} is set to the next eigenspace or recalculated if convergence is not fulfilled.

Based on the results from the eigengap modification, as described in Section 4.3.2, we modify the subspace dimension of our Krylov-Schur implementation. This is especially relevant for the matrix dimensioning, by an adaptation of the subspace dimension in relation to a significant eigengap.

The best eigengap is detected similarly to the description in Section 4.3.2. The subspace dimension then is greater than the significant eigengap plus one, i.e. $k = n_{\text{gap}} + 1$. This is reasonable, as the starting eigenvector for the subspace setup is the current valid approximation of the eigenvector. Then, the subspace is filled with all upcoming eigenvector approximations. It is thinkable to introduce a user parameter x to control the subspace dimension, i.e., $k = n_{\text{gap}} + x$. The effect of the calculation up to the eigengap, as explained in Section 4.3.2, is so significant that we will not neglect this. But a greater subspace can be useful when the eigenproblems are very ill-conditioned. The benefit is then similar to Krylov-spaces in preconditioners.

Finally, the current eigenvalue approximation $\widetilde{v}_{n_{\text{start}}}$ is updated, cf. Line 15 in Algorithm 4.4. This function can be extended to include multi-dimensional eigenspaces and the next higher eigensolutions. For a multi-dimensional eigenspace, which is measured by the eigengaps, we only update the entire eigenspace. This prevents switching the order of the calculated eigenvectors inside such multi-dimensional eigenspaces. Then we also check for convergence in the complete multi-dimensional eigenspace. For the next higher eigensolutions, the current eigensolution approximation can be calculated and temporarily saved. When this approximation is an improvement to the currently used eigensolution approximation, that one is updated.

For evaluation of the Krylov-Schur algorithm, we consider the same Poisson eigenproblem as previously, see Section 4.1.4. To increase the difficulty of solving this eigenproblem, we again want to calculate 9 eigenpairs. We choose 10 initial guess iteration steps on every matrix hierarchy level and an interval of 5 Ritz projections inside, as the previous benchmarks had shown this to be the most efficient parameter combination. We use no inverse iteration, as we intend to completely concentrate on the effects of the Krylov-Schur method. For these parameters, we evaluate the influence of the eigengap, which is correlated to the used subspace dimension.

Algorithm 4.4: Pseudo Code Krylov-Schur Method

Input: n_{ev}: number of requested eigenvalues
A, M: fine-level matrices
(if M not explicitly is given, it's the identity)
$\mathscr{C}\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$: stopping/convergence criterion
depending on current approximations $\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)_{1,\ldots,n_{\text{ev}}}$

Data: n_{gap}: maximum number of calculated eigenvalues ($n_{\text{ev}} \leq n_{\text{gap}}$)
n_{start}: current calculated eigensolution
$k(n_{\text{ev}}, n_{\text{gap}})$: subspace dimension
$H, U \in \mathbb{R}^{k \times k}$: dense matrix
$V \in \mathbb{R}^{n_f \times k}$: dense matrix
$\widetilde{\lambda}_j$: eigenvalue approximation for eigensolution j
$\widetilde{v}_j$: eigenvector approximation for eigensolution j
($\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$ initial guess using the AMG hierarchy as in Section 4.2)
res_j: residuum vector for eigensolution j

$n_{\text{start}} = 1;\ V(:,1) = \widetilde{v}_1$
while *not* $\mathscr{C}\left(\widetilde{\lambda}_j, \widetilde{v}_j\right)$ **do**
  Function `Arnoldi():`
    $H = 0.0$
    for $i = n_{start}, \ldots, k-1$ **do**
      $V(:, i+1) = V(:, i)/\widetilde{\lambda}_{n_{\text{start}}}$
      solve $AV(:, i+1) = MV(:, i)$ using GMRES and AMG
      for $j = 1, ..., i$ **do**
        $H(j, i) =< V(:, i+1), MV(:, j) >$
        $V(:, i+1) = V(:, i+1) - H(j, i)\, V(:, i)$
      $H(i+1, i) = \sqrt{< V(:, i+1), MV(:, i+1) >}$
      $V(:, i+1) = \frac{V(:, i+1)}{H(i+1, i)}$
  Function `Ritz problem():`
    solve $HU = \theta U$ using with a direct solver
  Function `Update V():`
    $\widetilde{\lambda}_{n_{\text{start}}} = \frac{1}{\theta(n_{\text{start}})}$
    $V = VU$
    $V(:, n_{\text{start}}) = \frac{V(:, n_{\text{start}})}{\| V(:, n_{\text{start}}) \|}$
    $\widetilde{v}_{n_{\text{start}}} = V(:, n_{\text{start}})$
  calculate residuum $\text{res}_{n_{\text{start}}} = A\widetilde{v}_{n_{\text{start}}} - \widetilde{\lambda}_{n_{\text{start}}} M\widetilde{v}_{n_{\text{start}}}$
  if $\mathscr{C}\left(\widetilde{\lambda}_{n_{start}}, \widetilde{v}_{n_{start}}\right)$ **then**
    $n_{\text{start}} = n_{\text{start}} + 1$

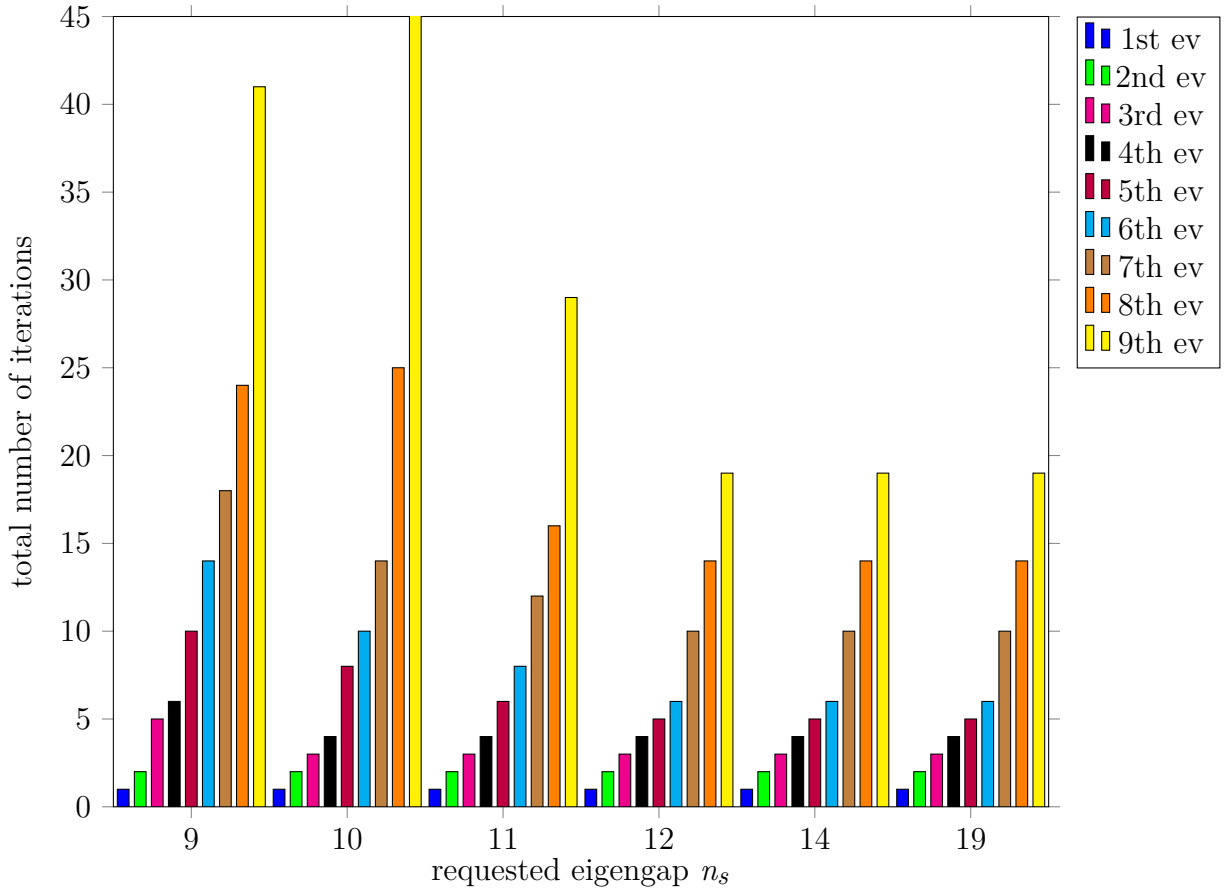


Figure 4.6: Total number of iterations without the iterations for the initial guess in the entire Krylov-Schur algorithm with various requested eigengaps n_s and 10 iterations and every 5 iterations a Ritz projection for the initial guess on every matrix hierarchy level. The eigenproblem is a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048.

requested eigengap n_s	**calculated eigengap $n_{\mathbf{gap}}$**	**used subspace dimension**	**runtime [sec]**
9	9	10	837.578
10	10	10	436.715
11	11	11	466.473
12	11	12	488.77
14	11	12	446.406
19	11	12	499.941

Table 4.4: Complete runtime for the Krylov-Schur algorithm, including an initial guess over the matrix hierarchy, with various requested eigengaps to calculate 9 eigenpairs using a Ritz projection every 5 iterations during the initial guess. The eigenproblem is a 2D homogeneous Poisson eigenproblem discretization with a five-point stencil on the unit square with a mesh size of 2048x2048.

In Table 4.4, we compare the requested eigengap with the automatically chosen subspace dimension and the runtime for full convergence of the first 9 eigenpairs. For requested eigengaps greater or equal to 12, we observe a stagnation in the runtime. The downsizing of the used subspace dimension has the effect to ensure the best balance between runtime and iteration count. But for the well-conditioned Poisson eigenproblem the Krylov-Schur algorithm is too expensive. Only the setting of an requested eigengap of 11, i.e., $n_s = 11$, is faster than the similar setting for inverse iteration. As a consequence we can conclude that the utilization of the Krylov-Schur method is only necessary when higher eigensolutions are requested or a more ill-conditioned problem should be solved.

The numbers of iterations, separated for every eigenpair, and the different requested eigengaps are presented in Figure 4.6. Therein, we see no difference in the iteration counts for eigengaps greater or equal than 12.

4.5 Numerical Examples and Results for Eigenvalue Calculation

In the following section, we explain some real-world applications for an eigenproblem solution using AMG.

An important application field for eigenproblem solutions is the internal usage for AMG itself by using so-called smooth vectors. The convergence rate of AMG can be improved by using smooth vectors to add additional information for the setup phase of AMG. One possibility is the usage of the smallest eigenvectors, as outlined in Section 4.1.1. These correlate to the low-frequency errors and should be appropriate handled on coarser matrix level to ensure a reduction of the correlated error componentes on the finest level. In Section 4.5.1, we show results for linear systems that occur for energy conservation

in structural mechanics. We will also shortly explain the functionality of AMG with these so-called smooth vectors based on eigenvector approximations.

Further application fields are the approximation of the smallest eigensolution for Graph Laplacians that occur in machine learning algorithms or for analyzing cryptocurrency transactions, see Section 4.5.2, or tomographic reconstruction in combination with pattern recognition.

All benchmarks have been performed on a node with Intel Xeon Gold 6130F dual 16-core CPU and 192 GB RAM.

4.5.1 Energy Conservation in Structural Mechanics

During the smoothed aggregation process it is possible to ensure the exact interpolation of given vectors, called smooth vectors [152, 153, 151]. By default, AMG only ensure to exactly interpolate constant vectors. This is perfectly sufficient for a method in most AMG applications. In ill-conditioned cases, as they arise in the wide field of elasticity problems, however, it can be beneficial to exactly interpolate more vectors where the smoother would be out-of-effect for. Such smoother vectors correspond to eigenvectors with the smallest eigenvalues [97].

By this, a doubled setup and solution process of AMG is established. In a first step, the eigenvectors are roughly approximated with a "classical" coarsening strategy. Afterwards, a new matrix hierarchy is constructed that uses smoothed aggregation with these approximated eigenvectors. This should give a better suitable coarse matrix representation and, thus, ensure better convergence rates. As this twiced setup process comes with much computational overhead, it is only relevant for ill-conditioned problems. The approximate nature of the eigenvectors does not matter then, they still improve the setup. Other solver methods often enough completely fail in such cases.

This use case of eigenvector calculation with AMG for improvements of the AMG strategy is evaluated on linear systems that result from elliptic energy conservation conditions. This problem is discretized with Finite Elements and, thus, in principle is suited for AMG technology. But the discretization scheme includes high aspect ratios and huge contrasts in stencil sizes. This leads to very high condition numbers and standard techniques are not suited anymore. The convergence rate of these ill-conditioned problems can be drastically improved by the twiced AMG solution process. A general application of smoothed aggregation for structural mechanical problems is described in [36].

For this kind of problems, the so-called alternating Schwarz approach is a useful supplement to AMG methods. The Schwarz approach was first described in [133]. The general convergence proof for elliptic boundary values is presented in [100] and a wider description can be found in [101]. The alternating Schwarz approach is a subclass of

Figure 4.7: Schematic visualization of alternating Schwarz approach with overlap. The green and blue rectangles visualize the subproblems that are defined on the to solving linear system. These subproblems are solved independently. The black rectangle is the overlapping part of the subproblems. This overlap is updated accordingly after each subproblem iteration.

the so-called domain decomposition methods. A wide explanation of these methods can be found in the books [120, 135] and the references therein.

The idea of the alternating Schwarz approach is to divide the problem in two subproblems with an overlapping part. Then, alternatingly the subproblems are solved: The solution of one subproblem is used for the overlap of the other subproblem and vice versa. A schematic visualization of this is given in Figure 4.7.

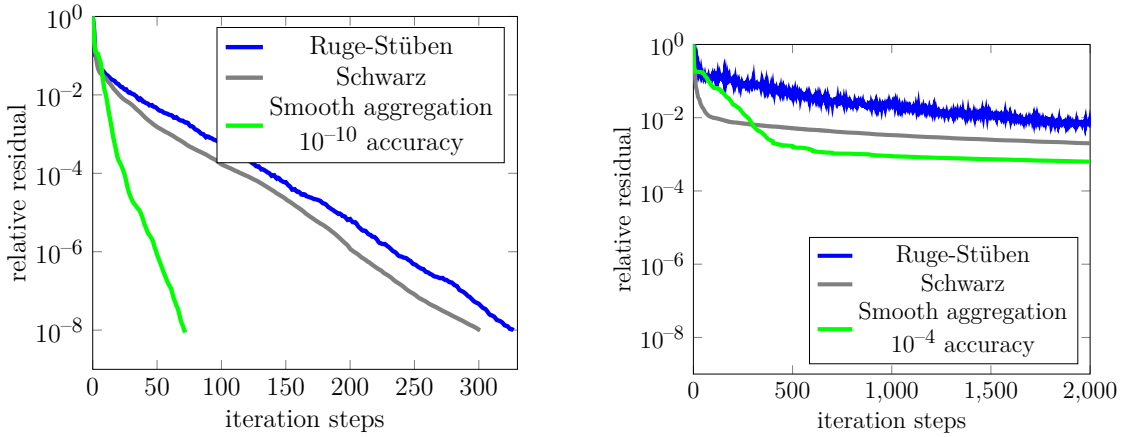
We use this idea to eliminate those parts of the given system that are difficult to handle for AMG. These are handled in a separate block by a direct solver or an iterative method. The remaining part of the system is solved by an AMG approach. As this is currently the best solution approach for the given problems, we compare this with our improved smoothed aggregation by using fitted eigenvectors to this problem.

In the following, we present the results for two user test case matrices. These are quite difficult to solve, as the condition number for the first test is in the order of 10^9 and the second one of 10^{14}. We compare three AMG solution approaches for these industrial use cases. As a first approach, standard AMG, i.e., Ruge-Stüben coarsening with standard interpolation, is used. The second approach is the Schwarz approach or domain decomposition. Finally, we compare this to smoothed aggregation, where the previously calculated eigenvectors are used to improve the convergence rate. For smoothed aggregation, we present two results, as we want to emphasize that the calculated eigenvector approximation do not need to be that exact. Thus, we compare

	iterations	**residual**	**solution time [sec]**
user case 1			
Standard	327	$2.27 \cdot 10^{-8}$	60.1
Schwarz	301	$2.27 \cdot 10^{-8}$	91.9
Smoothed aggregation 10^{-10} eigensolution accuracy	72	$2.00 \cdot 10^{-8}$	380.4
Smoothed aggregation 10^{-5} eigensolution accuracy	104	$2.17 \cdot 10^{-8}$	555.9
user case 2			
Standard	10000	$2.72 \cdot 10^{-4}$	2088.9
Schwarz	10000	$5.63 \cdot 10^{-4}$	4171.9
Smoothed aggregation 10^{-4} eigensolution accuracy	10000	$8.34 \cdot 10^{-5}$	77785.3
Smoothed aggregation 10^{-2} eigensolution accuracy	10000	$8.40 \cdot 10^{-4}$	84391.4

Table 4.5: Results for two user test cases with a standard AMG-approach, i.e., Ruge-Stüben coarsening, a Schwarz elimination and smoothed aggregation with two accuracies for the eigensolutions. We compare the result with the iterations, the final residual and the solution time.

eigensolution approximations that are gained only with an initial guess and eigensolution approximations that are gained using inverse iteration and Krylov iteration. As both user test cases have quite different condition numbers, the gained residual for the eigensolution approximations is different and added to the result table. In Table 4.5, we see the three main key facts to evaluate the solution approaches. For a better evaluation of the convergence, we plot the relative residual of the calculated solution after the iteration steps until convergence is reached for the first user test case in Figure 4.8. For the second, more ill-conditioned user test case, we plot until 2000 iterations. The plot shows that Ruge-Stüben has difficulties to reduce the relative residual and is oscillating.



(a) user case 1

(b) user case 2

Figure 4.8: We plot the convergence history for two user test cases with a standard AMG-approach, i.e., Ruge-Stüben coarsening, a Schwarz elimination and smoothed aggregation with two accuracies for the eigensolutions. The convergence rate for the first user test case is plotted until a residual reduction of eight order of magnitude is reached. For the second user test case we plot the convergence behavior of the first 2000 iterations.

4.5.2 Graph Network Problems

Graph Laplacians in matrix form describe the connectivity between their nodes and edges [40]. The definition is based on the idea of a discrete version of the Laplace operator in finite element theory. Due to this construction, using AMG is a canonical idea. But, per definition, Graph Laplacians are only positive semi-definite, whereas AMG requires positive definiteness. Our examples, Megaman and bitcoin, use a very slight increase in the diagonal to ensure positive definiteness.

As described, Graph Laplacians do not directly fulfill the requirements for the application of AMG. In [19], the classical AMG theory, as we shortly introduced in Chapter 2, is extended to Graph Laplacians. The suitability of AMG for Graph Laplacians is strictly related to the coarsening and interpolation methods. For classical AMG theory, these define the success of the linear solver, as they have to fulfill special requirements to ensure convergence of the linear solver, e.g., see Theorem 3.1. Furthermore, in [30], the usage of aggregation based coarsening strategies for Graph Laplacians and the resulting convergence rate is analyzed. Using aggregation AMG for Graph Laplacians is also described in [107, 108]. Another approach for Graph Laplacians is a Lean AMG, which is introduced in [55, 56]. This is directly applicable to the semi-definite Graph Laplacians. Due to the way the matrices are generated in our application, this is not necessary here, though.

Megaman

Manifold learning, e.g. spectral embedding [156, 110], local linear embedding [122, 128] or isomap [147, 15], is used to calculate a non-linear smaller representation of high-dimensional data. By this lower-dimensional representation, for instance, a visualization in the Euclidean space is then possible or clustering into significant features of the data. All of these mentioned methods need an approximation of the smallest eigensolutions.

The very shortened ideas of the mentioned algorithms are:

- Spectral embedding [156, 110] exploits spectral properties of the similarity graph of the data.
- Local linear embedding [122, 128] defines in a first step a neighborhood graph of the data. Afterwards the embedding into a lower-dimensional space is optimized such that the reconstruction error of this process is minimal.
- Dimension reduction by isomapping [147, 15] aims at preserving the distances between the data points. The relation between the data points are summarized in a neighborhood graph. For this graph then an optimized low-dimensional representation is calculated.

All three algorithms have in common to define a neighborhood or similarity graph. A very common way to do so is by calculating a Graph Laplacian.

For our benchmark tests we use the Megaman software. The target application field of the Megaman software is molecular dynamic, chemistry and astronomy with graphs up to millions of data points [98, 99]. In this python package the previously mentioned dimension reduction algorithms and variants of Graph Laplacians are already implemented. Additionally, a link to Stanford' large network dataset collection [90] is given and, thus, a wide variation of applications is available. Since most algorithms perform better with positive definite matrices, the Megaman software already applies a slight increase to the diagonal as most Graph Laplacians, by definition, are only positive semi-definite.

At the moment, a before-hand defined number of eigensolutions is calculated in the Megaman software. But less are actually used in the algorithms later on. Thus, exchanging the hard limit of calculate eigensolutions by a case dependent eigengap for the estimation of necessary calculable eigensolutions is of interest. But currently, it is not clear if the requirements for the eigengap inside the Megaman algorithms are similar to the ones that we defined in Section 4.3.2.

In Table 4.6, we show the results for various Graph Laplacians from the Megaman project. Two of the three example graphs we already introduced in Section 3.3.3. The additional example "Oregon" is a communication graph between routers in the

number of eigensolutions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
as-Caida	0	8	9	9	9	9	9	9
Oregon	0	17	29	33	63	75	111	111
soc-Slashdot0811	13	35	57	57	195	310	310	407

Table 4.6: Number of inverse iterations of eigensolution approximations limited to 1000 iterations, and a first approximation with maximal 10 iterations on every matrix hierarchy level for various Graph Laplacians from the Megaman project.

US-state Oregon [88]. As we already have seen in Section 3.3.3, using AM-AMG as setup approach, for these examples is a necessary condition for the applicability of AMG.

In EP-AMG we use the inverse iteration with a maximum iteration number of 1000 and 10 iterations during the first approximation for every matrix hierarchy level. Due to the uncertain requirements for the limit of calculable eigensolutions, we do not use an eigengap and set the requested number of eigensolutions to 20. As the eigenvalues are quite close to each other, Ritz projection is necessary after every iteration step to ensure good orthogonalization results.

Transaction Graphs for Bitcoins

Transaction graphs are a useful analysis method for transactions of digital cryptocurrencies as explained in [51, 103, 134, 58]. To analyze these transactions and to identify transaction clusters eigensolutions are used. The main problem in this application field is the inhomogeneity of the transactions that ends in inhomogeneous graph stencils.

The used transaction graphs are following the definition in [37]. The nodes of the transaction graph are either addresses or transactions themselves. A directed edge between an address and a transaction exists if the address inits the transaction. Alternatively, a directed edge between a transaction and an address exists if the transaction has this address as an outcome. In one transaction, multiple addresses are concluded.

The most heavily traded cryptocurrency is bitcoin. Thus, we have selected this for analysis purpose. We analyze our eigenvalue algorithm for two transaction graphs, one based on a transaction day in 2011 and one in 2020 [114]. During this time period, the transactions per day have drastically increased and, thus, the direct solver is not able to solve the eigenproblem for the later example. For the graph from 2011, we can use a direct solver and, thus, can compare the result (accuracy and solution time) with our implemented eigenvalue algorithm. The transaction graph from 2011 has 19882 nodes and 65030 graph connections. The graph from 2020 has 1361684 nodes and 4516090 graph connections.

	2011 direct	**2011 AMG**	**2020 direct**	**2020 AMG**
simulation time [sec]	65893	2065	fail	137567
eigensolution 1				
eigenvalue	1.0	1.0	/	1.0
residual	$0.12 \cdot 10^{-14}$	$0.14 \cdot 10^{-15}$		$0.22 \cdot 10^{-5}$
eigensolution 2				
eigenvalue	1.0	1.0	/	1.0001
residual	$0.70 \cdot 10^{-15}$	$0.14 \cdot 10^{-15}$		$0.15 \cdot 10^{-3}$
eigensolution 3				
eigenvalue	1.0	1.0	/	1.0002
residual	$0.36 \cdot 10^{-15}$	$0.45 \cdot 10^{-15}$		$0.81 \cdot 10^{-5}$
eigensolution 4				
eigenvalue	1.0	1.0	/	1.006
residual	$0.11 \cdot 10^{-15}$	$0.55 \cdot 10^{-11}$		$0.12 \cdot 10^{-3}$
eigensolution 5				
eigenvalue	1.0	1.0016	/	1.006
residual	$0.11 \cdot 10^{-15}$	$0.20 \cdot 10^{-9}$		$0.11 \cdot 10^{-4}$
eigensolution 6				
eigenvalue	1.0	1.0017	/	1.008
residual	$0.36 \cdot 10^{-15}$	$0.91 \cdot 10^{-9}$		$0.18 \cdot 10^{-3}$

Table 4.7: Evaluation of the two bitcoin transaction graphs with a direct solver and our presented eigensolution algorithm. We compare the simulation time and present the first six eigenvalues and the residual of the eigensolution

Our eigenvalue algorithm has the aim to calculate 6 eigensolutions with 100 first approximation iterations, 10000 inverse iterations and 100 Krylov Schur iterations. The eigengap search space is set to 16. It actually uses 16, as there is no further significant eigengap.

In Table 4.7, we compare the main keyfacts about the approximated eigensolutions. As expected, we see that the direct solver for the transaction analysis from 2020 fails due to the size of the transaction graph. With our AMG-based algorithm, however, it is perfectly possible to calculate the smallest eigensolutions in this case.

4.5.3 Tomography

A more further outlook for using the combination of AM-AMG and EP-AMG is the application of AMG in the field of tomography. The herein described methods are based on information in the context of tomography for reservoir analysis. For an unknown domain/image systematically X-rays are applied. Based on these results, the

domain/image should be reconstructed. Therefore, the domain can be a reservoir or the image of human body parts in the medicine.

Solving tomographic reconstruction problems [53] (inverse problems) leads to a least square problem that is noise-sensitive. Currently, direct solvers and a smoothing regularization, e.g., with Kaczmarz, is used to solve this. The problem is set up by rays/projections through the domain/image. This leads to a sparse rectangular coefficient matrix $A \in \mathbb{R}^{m \times n}$. When testing the reconstruction algorithms the description of the image/domain $x \in \mathbb{R}^n$ is well known. Thus, the tomographic result is $b = Ax \in \mathbb{R}^m$. As real measurements are not correct, some noise $\epsilon \in R^m$ is added to the "measured" image/domain.

To reconstruct the image, the least-square problem

$$\min \| A\bar{x} - \bar{b} \| \tag{4.13}$$

with $\bar{b} = b + \epsilon$ has to be solved. Following [117], we define the linear system

$$\bar{A} := \begin{pmatrix} \mathbb{1} & A \\ A^T & -\omega^2 \mathbb{1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \bar{b} \\ 0 \end{pmatrix} \tag{4.14}$$

with $\omega \in \mathbb{R}$ and $\hat{A} \in \mathbb{R}^{(m+n)\times(m+n)}$. For $\omega \to 0$ the solution vector $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ converges towards $\begin{pmatrix} \text{res} \\ \bar{x} \end{pmatrix}$. We chose the so-called damping parameter ω such that AMG is applicable to $\bar{A}$.

To show the proof-of-concept for this wide application field, we use the Matlab simulation AIR tools II [64]. We extract the sparse linear system $\bar{A}$ that can be formulated equivalently to the least-squares problem, as we have motivated previously, and solve it by AMG. Afterwards, we reread the solution vector to Matlab to visualize the reconstructed image. As default method in AIR tools II the tomographic method ART (algebraic reconstruction technique) with Kaczmarz is used [64].

As examples, we choose Shepp-Logan as a medical example for computer tomography and simulated grains in a crystalline medium [64] for reservoir domain analyzing purposes. Shepp-Logan is a commonly used example for tomographic reconstruction. The grain examples arises from the initial application field during reservoir simulation projects.

The arising linear system $\bar{A}$ is solvable with Ruge-Stüben AMG, but even better with AM-AMG. With aggregation coarsening the problem is not solvable as no valid matrix hierarchy is constructed. As the matrix arises from discrete beams during the tomography process, the density of the matrix in the "beams' direction" is higher. The AM-AMG coarsening approach can handle this quite well and, thus, ends up in the

	simulation time [sec]	**setup time [sec]**	**coarse level size**	**iterations**	**residual reduction**
Shepp-Logan					
Ruge-Stüben	900.9	841.3	40000	5	$0.141 \cdot 10^{-9}$
AM-AMG	45.2	38.6	936	68	$0.951 \cdot 10^{-8}$
grains					
Ruge-Stüben	895.6	835.6	40000	5	$0.104 \cdot 10^{-9}$
AM-AMG	51.3	44.9	936	64	$0.813 \cdot 10^{-8}$

Table 4.8: Results for the two reformulated least square problems arising from tomographic reconstruction. We exemplarily chose the Shepp-Logan example that is based on a medical CT and a standard test problem and the grain example, which is a random constructed image with Voronoi cell in a crystalline medium and occurs in reservoir simulation process.

drastically faster setup as the results in Table 4.8 show. The coarsening process with Ruge-Stüben has troubles to construct a reasonable coarse-level size for both examples and, thus, the runtime increases drastically.

When looking at only one setup approach, we see that the number of matrix hierarchy levels and coarse-level size is similar for both examples. This holds for Ruge-Stüben coarsening and AM-AMG separately. This emphasizes that these kind of problems would dramatically improve with a setup recycling, as the tomographic process results in a similar matrix structure for subsequent systems. Just the measured values that correlate to the matrix entries are differing.

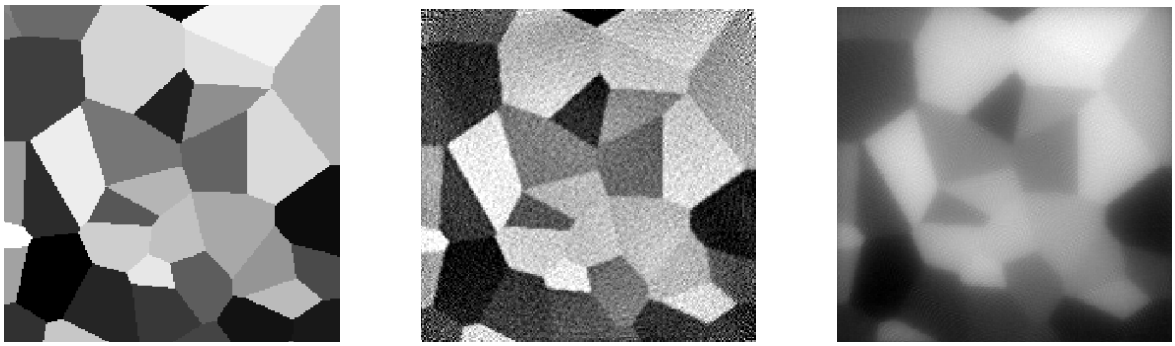
We should note that the objective here is not a perfect image reconstruction. We rather aim at reconstructing all significant image structures in an efficient manner. As outlined, AMG here is the more advantageous the higher the resolutions are. In full sample sets rather than simple examples moreover, setup re-usage further improves AMG's efficiency.

In Figure 4.9, we present the test case Shepp-Logan and in Figure 4.10 the grains. In both cases we plot the original raw data in part (a) of the figures. Based on this raw data, the tomography is executed and 2% relative noise are added. In the pictures we compare the default method ART with Kaczmarz of the Air Tool II suite implemented in Matlab with our AMG-based solution. As we can see, the solutions are quite similar - besides some scaling that results in different gray shades in the pictures. However, all details can be clearly seen in both cases.

As a perspective, the rays of tomography are most often fixed, e.g., X-ray CT problems. By this, a setup reuse of the coarsening with recalculated interpolation and restriction seems to be a promising approach, as the sparsity structure doesn't change.

(a) original (b) ART with Kaczmarz (c) AMG

Figure 4.9: In Figure 4.9a, we see the original picture of the Shepp-Logan medical phantom. On this picture the tomography is executed and some additional noise added. Subsequently these generated data are reconstructed with ART using Kaczmarz method in Figure 4.9b and with AMG for the reformulated least-square problem in Figure 4.9c



(a) original (b) ART with Kaczmarz (c) AMG

Figure 4.10: In Figure 4.10a, we see the original picture of the simulated grains. On this picture the tomography is executed and some additional noise added. Subsequently these generated data are reconstructed with ART using Kaczmarz method in Figure 4.10b and with AMG for the reformulated least-square problem in Figure 4.10c

The reconstructed images should be analyzed in most cases as tomography is used for imaging of unknown samples/domains. For this purpose, a feature recognition of the reconstructed images is an option. In section 4.5.2, we have seen that EP-AMG is working very well for the eigensolution calculation of such graph problems. As we needed our newly developed AM-AMG approach for the tomographic reconstruction , we expect that this is also relevant when solving the respective eigenproblems.

CHAPTER 5

Conclusion and Outlook

In this thesis, we presented a strategy to exploit AMG in eigenproblems from data science and beyond in a generic way. On the one hand, we introduced a new setup procedure for AMG, AM-AMG, that is well-applicable also for highly varying matrix patterns. On the other hand, we presented ways to exploit AMG in the eigensolution procedure, EP-AMG, in a very generic way. Moreover, with the combination of algorithms, we exploited synergies.

We started with introducing AM-AMG. AM-AMG is a setup approach that can handle strong heterogeneities in the sparsity pattern of matrices. This setup is an extension of the well-known aggregative coarsening, but enriched with overlapping aggregates. Additionally, the aggregate construction process introduces three function types for the variables. By this, we are able to construct a locally ideal interpolation per aggregate. Additionally, we have the possibility to fine-tune the aggregation process by various parameters. Thus, we are easily able to adapt the algorithm to matrices occurring from network problems. Benchmarks have shown that this form of coarsening is necessary to be able to construct a valid setup of AMG, even under memory restrictions.

Currently, this setup approach is parallelized with shared memory. An extension to distributed memory, however, is rather straight-forward to implement, as the parallelization algorithm is directly transferable. Further investigation is promising regarding the potential of using AM-AMG on a GPU. The setup phase of AM-AMG is also splitted into two parts. One part is constructing the coarsening, while the other calculates the interpolation afterwards. The one-time splitting is slightly expensive, but the interpolation calculations are inherently parallel. Thus, the potential of an efficient AMG algorithm for GPU results from the interpolation calculation in every simulation time step that can really benefit from the GPU acceleration.

As we target at the application of AMG for Big Data problems, not only mere linear systems are of relevance, but the calculation of a few (smallest) eigensolutions. To use

AMG for eigensolution calculations, a few adaptions are necessary as we have seen when describing EP-AMG. We presented three algorithms with whom we can efficiently exploit the matrix hierarchy of AMG. A starting point was the exploitation of the matrix hierarchy to construct an initial guess for the eigensolutions. Afterwards, we are continuing with this initial guess and the inverse iteration and Krylov-Schur method. While these are well-known algorithms to approximate a few eigensolutions, we needed to adapt them at different points to be able to benefit from AMG.

In various examples, e.g., network graph or bitcoin transactions, we have demonstrated that EP-AMG is working properly. Furthermore, we extend the usage of calculated eigensolutions to an internal improvement of AMG hierarchy for very ill-conditioned problems. In a real-world problem, we have seen that by this approach, for a first time it is possible to reach the relevant residual reduction.

In the example section, we have also pointed out even further optimization potential. For the Krylov-Schur algorithm, the simultaneous calculation of eigensolution when setting up the Arnoldi decomposition could be of interest. With this feature, the efficiency of the algorithm for more smallest eigensolutions would be increased further.

As we are talking about growing matrix data and increasing computational power, the eigensolution algorithms need to be parallelized. As we use the SAMG implementation for AMG that already is well parallelized, we only have to think about the eigensolution algorithm itself. The eigensolution algorithms are quite difficult to parallelize as they depend on an orthogonalization to the previous approximated eigensolutions. A first idea would be to use a wavefront parallelization. This means that a slightly shifted start of the eigensolution approximation on the parallel partitions. An alternative could be the orthogonalization regarding previous iterations of the eigensolution are accepted at some point.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] M. Adams, M. Brezina, J. Hu, and R. Tuminaro. Parallel Multigrid Smoothing: Polynomial Versus Gauss-Seidel. *Journal of Computational Physics*, 188(2):593–610, July 2003.

[2] J. Alzubi, A. Nayyar, and A. Kumar. Machine Learning from Theory to Algorithms: An Overview. *Journal of Physics: Conference Series*, 1142:012012, Nov. 2018.

[3] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users‘ Guide*. Society for Industrial and Applied Mathematics, Jan. 1999.

[4] W. E. Arnoldi. The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem. *Quarterly of Applied Mathematics*, 9(1):17–29, 1951.

[5] K. E. Atkinson. *An Introduction to Numerical Analysis*. John Wiley & Sons, Jan. 1989.

[6] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, Mar. 1994.

[7] K. Aziz and A. Settari. *Petroleum Reservoir Simluation*. Applied Science Publ. Ltd., London, UK, 1979.

[8] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang. Multigrid Smoothers for Ultraparallel Computing. *SIAM Journal on Scientific Computing*, 33(5):2864–2887, Jan. 2011.

[9] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. PETSc/TAO users manual. Technical Report ANL-21/39 - Revision 3.16, Argonne National Laboratory, 2021.

[10] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. PETSc Web page. `https://petsc.org/`, 2021.

[11] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[12] K.-J. Bathe. *Finite Element Procedures*. Prentice-Hall, 1996.

[13] K.-J. Bathe and E. L. Wilson. *Numerical Methods in Finite Element Analysis*. Prentice Hall, 1976.

[14] F. L. Bauer and C. T. Fike. Norms and Exclusion Theorems. *Numerische Mathematik*, 2(1):137–141, Dec. 1960.

[15] M. Bernstein, V. De Silva, J. C. Langford, and J. B. Tenenbaum. Graph Approximations to Geodesics on Embedded Manifolds. Technical report, Citeseer, 2000.

[16] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[17] Å. Björck. Numerics of Gram-Schmidt Orthogonalization. *Linear Algebra and its Applications*, 197-198:297–316, Jan. 1994.

[18] R. Blaheta. A Multilevel Method with Overcorrection by Aggregation for Solving Discrete Elliptic Problems. *Journal of Computational and Applied Mathematics*, 24(1-2):227–239, Nov. 1988.

[19] M. Bolten, S. Friedhoff, A. Frommer, M. Heming, and K. Kahl. Algebraic Multigrid Methods for Laplacians of Graphs. *Linear Algebra and its Applications*, 434(11):2225–2243, June 2011.

[20] A. Borzì and G. Borzì. Algebraic Multigrid Methods for Solving Generalized Eigenvalue Problems. *International Journal for Numerical Methods in Engineering*, 65(8):1186–1196, Feb. 2006.

[21] D. Braess. Towards Algebraic Multigrid for Elliptic Problems of Second Order. *Computing*, 55(4):379–393, Dec. 1995.

[22] A. Brandt. *Multigrid Techniques*. GMD, St. Augustin, 1984.

[23] A. Brandt. Algebraic Multigrid Theory: The Symmetric Case. *Applied Mathematics and Computation*, 19(1-4):23–56, July 1986.

[24] A. Brandt, J. Brannick, K. Kahl, and I. Livshits. Bootstrap AMG. *SIAM Journal on Scientific Computing*, 33(2):612–632, Jan. 2011.

[25] A. Brandt, J. Brannick, K. Kahl, and I. Livshits. Bootstrap Algebraic Multigrid: Status Report, Open Problems, and Outlook. *Numerical Mathematics: Theory, Methods and Applications*, 8(1):112–135, 2015.

[26] A. Brandt and O. E. Livne. *Multigrid Techniques*. Society for Industrial and Applied Mathematics, Jan. 2011.

[27] A. Brandt, S. McCormick, and J. Ruge. Multigrid Methods for Differential Eigenproblems. *SIAM Journal on Scientific and Statistical Computing*, 4(2):244–260, June 1983.

[28] A. Brandt and D. Ron. Multigrid Solvers and Multilevel Optimization Strategies. In *Combinatorial Optimization*, pages 1–69. Springer US, 2003.

[29] J. Brannick, M. Brezina, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. An Energy-Based AMG Coarsening Strategy. *Numerical Linear Algebra with Applications*, 13(2-3):133–148, 2006.

[30] J. Brannick, Y. Chen, J. Kraus, and L. Zikatanov. Algebraic Multilevel Preconditioners for the Graph Laplacian Based on Matching in Graphs. *SIAM Journal on Numerical Analysis*, 51(3):1805–1827, Jan. 2013.

[31] J. Brannick, X. Hu, C. Rodrigo, and L. Zikatanov. Local Fourier Analysis of Multigrid Methods with Polynomial Smoothers and Aggressive Coarsening. *Numerical Mathematics: Theory, Methods and Applications*, 8(1):1–21, Feb. 2015.

[32] J. Brannick, S. P. MacLachlan, J. B. Schroder, and B. S. Southworth. The Role of Energy Minimization in Algebraic Multigrid Interpolation. *arXiv: 1902.05157*, Feb. 2019.

[33] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Algebraic Multigrid Based on Element Interpolation (AMGe). *SIAM Journal on Scientific Computing*, 22(5):1570–1592, Jan. 2001.

[34] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. Adaptive Smoothed Aggregation (αSA) Multigrid. *SIAM Review*, 47(2):317–346, Jan. 2005.

[35] M. Brezina, C. Heberton, J. Mandel, and P. Vanek. An Iterative Method with Convergence Rate Chosen a Priori. *UCD/CCM Report*, 140:513–521, 1999.

[36] M. Brezina, C. Tong, and R. Becker. Parallel Algebraic Multigrids for Structural Mechanics. *SIAM Journal on Scientific Computing*, 27(5):1534–1554, Jan. 2006.

[37] C. Cachin, A. D. Caro, P. Moreno-Sanchez, B. Tackmann, and M. Vukolic. The transaction graph for modeling blockchain semantics. *Cryptoeconomic Systems*, Dec. 2020.

[38] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An Overview of Machine Learning. In *Machine Learning*, pages 3–23. Elsevier, 1983.

[39] M. Christie and M. Blunt. Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques. *SPE Reservoir Evaluation & Engineering*, 4(04):308–317, Aug. 2001.

[40] F. R. Chung. *Spectral Graph Theory*, volume 92. American Mathematical Society, 1997.

[41] T. Clees. *AMG Strategies for PDE Systems with Applications in Industrial Semiconductor Simulation*. PhD thesis, University of Cologne, 2005.

[42] R. R. Coifman and S. Lafon. Diffusion Maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, July 2006.

[43] R. Courant and D. Hilbert. *Methods of Mathematical Physics*. Wiley, Apr. 1989.

[44] J. De la Porte, B. Herbst, W. Hereman, and S. Van Der Walt. An Introduction to Diffusion Maps. In *Proceedings of the 19th Symposium of the Pattern Recognition Association of South Africa (PRASA 2008), Cape Town, South Africa*, pages 15–25, 2008.

[45] W. E. Donath and A. J. Hoffman. Lower Bounds for the Partitioning of Graphs. *IBM Journal of Research and Development*, 17(5):420–425, Sept. 1973.

[46] F. A. Dul and K. Arczewski. The Two-Phase Method for Finding a Great Number of Eigenpairs of the Symmetric or Weakly Non-symmetric Large Eigenvalue Problems. *Journal of Computational Physics*, 111(1):89–109, Mar. 1994.

[47] T. W. Edgar and D. O. Manz. Machine Learning. In *Research Methods for Cyber Security*, pages 153–173. Elsevier, 2017.

[48] S. Ehrmann, S. Gries, and M. A. Schweitzer. Generalization of Algebraic Multiscale to Algebraic Multigrid. *Computational Geosciences*, 24(2):683–696, June 2019.

[49] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational Iterative Methods for Nonsymmetric Systems of Linear Equations. *SIAM Journal on Numerical Analysis*, 20(2):345–357, Apr. 1983.

[50] R. Falgout. An Introduction to Algebraic Multigrid. *Computing in Science and Engineering*, 8(6):24–33, Nov. 2006.

[51] M. Fleder, M. S. Kester, and S. Pillai. Bitcoin Transaction Graph Analysis. *arXiv: 1502.01657*, Feb. 2015.

[52] R. Fletcher. Conjugate Gradient Methods for Indefinite Systems. In *Lecture Notes in Mathematics*, pages 73–89. Springer Berlin Heidelberg, 1976.

[53] L. Flores, V. Vidal, and G. Verdú. System Matrix Analysis for Computed Tomography Imaging. *PLOS ONE*, 10(11):e0143202, Nov. 2015.

[54] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, June 1962.

[55] A. Fox and T. Manteuffel. Algebraic Multigrid for Directed Graph Laplacian Linear Systems (NS-LAMG). *Numerical Linear Algebra with Applications*, 25(3):e2152, Jan. 2018.

[56] A. L. Fox. *Algebraic Multigrid for Graph Laplacian Linear Systems: Extensions of AMG for Signed, Undirected and Unsigned, Directed Graphs.* PhD thesis, University of Colorado at Boulder, 2017.

[57] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

[58] A. Greaves and B. Au. Using the Bitcoin Transaction Graph to Predict the Price of Bitcoin. *No data*, 8:416–443, 2015.

[59] S. Gries. *System-AMG Approaches for Industrial Fully and Adaptive Implicit Oil Reservoir Simulations.* PhD thesis, University of Cologne, 2015.

[60] S. Gries. On the Convergence of System-AMG in Reservoir Simulation. *SPE Journal*, 23(02):589–597, Jan. 2018.

[61] S. Gries. Informed Machine Learning to Maximize Robustness and Computational Performance of Linear Solver. In D. Schulz and D. Trabold, editors, *Informed Machine Learning.* Springer, To Appear 2023.

[62] W. Hackbusch. On the Multi-Grid Method Applied to Difference Equations. *Computing*, 20(4):291–306, Dec. 1978.

[63] W. Hackbusch. *Multi-Grid Methods and Applications.* Springer Berlin Heidelberg, 1985.

[64] P. C. Hansen and J. S. Jørgensen. AIR Tools II: Algebraic Iterative Reconstruction Methods, Improved Implementation. *Numerical Algorithms*, 79(1):107–137, Nov. 2017.

[65] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

[66] J. He and Z.-F. Fu. *Modal Analysis*. Elsevier, 2001.

[67] V. E. Henson. Multigrid Methods for Nonlinear Problems: An Overview. In C. A. Bouman and R. L. Stevenson, editors, *Computational Imaging*. SPIE, June 2003.

[68] V. E. Henson and U. M. Yang. BoomerAMG: A Parallel Algebraic Multigrid Solver and Preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, Apr. 2002.

[69] V. Hernández, J. E. Román, A. Tomás, and V. Vidal. Krylov-Schur Methods in SLEPc. *Universitat Politecnica de Valencia, Tech. Rep. STR-7*, 2007.

[70] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An Overview of the Trilinos Project. *ACM Transactions on Mathematical Software*, 31(3):397–423, Sept. 2005.

[71] M. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409, Dec. 1952.

[72] U. Hetmaniuk. A Rayleigh Quotient Minimization Algorithm Based on Algebraic Multigrid. *Numerical Linear Algebra with Applications*, 14(7):563–580, 2007.

[73] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Jan. 2002.

[74] I. T. Jolliffe and J. Cadima. Principal Component Analysis: a Review and Recent Developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, Apr. 2016.

[75] H. Klie and H. Florez. Data-Driven Discovery of Unconventional Shale Reservoir Dynamics. In *SPE Reservoir Simulation Conference*. SPE, Mar. 2019.

[76] A. V. Knyazev. A Preconditioned Conjugate Gradient Method for Eigenvalue Problems and its Implementation in a Subspace. In *Numerical Treatment of Eigenvalue Problems Vol. 5 / Numerische Behandlung von Eigenwertaufgaben Band 5*, pages 143–154. Birkhäuser Basel, 1991.

[77] A. V. Knyazev. Preconditioned Eigensolves - an Oxymoron. *Electronic Transition on Numerical Analysis*, 7:104–123, 1998.

[78] A. V. Knyazev. *Preconditioned Eigensolvers: Practical Algorithms*. University of Colorado at Denver, 1999.

[79] A. V. Knyazev. Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method. *SIAM Journal on Scientific Computing*, 23(2):517–541, Jan. 2001.

[80] T. V. Kolev and P. S. Vassilevski. AMG by Element Agglomeration and Constrained Energy Minimization Interpolation. *Numerical Linear Algebra with Applications*, 13(9):771–788, 2006.

[81] A. Krechel and K. Stüben. Parallel Algebraic Multigrid Based on Subdomain Blocking. *Parallel Computing*, 27(8):1009–1031, July 2001.

[82] D. Kressner. Numerical Methods for General and Structured Eigenvalue Problems. *Lecture Notes in Computatinal Science and Engineering*, 46, 2005.

[83] D. Kushnir, M. Galun, and A. Brandt. Efficient Multilevel Eigensolvers with Applications to Data Analysis Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1377–1391, Aug. 2010.

[84] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic Mode Decomposition*. Society for Industrial and Applied Mathematics, Nov. 2016.

[85] C. Lanczos. Solution of Systems of Linear Equations by Minimized Iterations. *Journal of Research of the National Bureau of Standards*, 49(1):33, July 1952.

[86] I. Lashuk and P. S. Vassilevski. On Some Versions of the Element Agglomeration AMGe Method. *Numerical Linear Algebra with Applications*, 15(7):595–620, Sept. 2008.

[87] J. A. Lee and M. Verleysen, editors. *Nonlinear Dimensionality Reduction*. Springer New York, 2007.

[88] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs Over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, Aug. 2005.

[89] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1):2, Mar. 2007.

[90] J. Leskovec and A. Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data`, 2014.

[91] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics*, 6(1):29–123, Jan. 2009.

[92] K.-A. Lie, O. Møyner, J. R. Natvig, A. Kozlova, K. Bratvedt, S. Watanabe, and Z. Li. Successful Application of Multiscale Methods in a Real Reservoir Simulator Environment. *Computational Geosciences*, 21(5-6):981–998, Mar. 2017.

[93] O. E. Livne and A. Brandt. Lean Algebraic Multigrid (LAMG): Fast Graph Laplacian Linear Solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, Jan. 2012.

[94] J. Mandel and S. McCormick. A Multilevel Variational Method for Au = λBu on Composite Grids. *Journal of Computational Physics*, 80(2):442–452, Feb. 1989.

[95] T. A. Manteuffel, S. Münzenmaier, J. Ruge, and B. Southworth. Nonsymmetric Reduction-Based Algebraic Multigrid. *SIAM Journal on Scientific Computing*, 41(5):242–268, Jan. 2019.

[96] T. A. Manteuffel, J. Ruge, and B. S. Southworth. Nonsymmetric Algebraic Multigrid Based on Local Approximate Ideal Restriction ($\updownarrow$ AIR). *SIAM Journal on Scientific Computing*, 40(6):A4105–A4130, Jan. 2018.

[97] S. McCormick and J. Ruge. Algebraic Multigrid Methods Applied to Problems in Computational Structural Mechanics. In *State-of-the-Art Surveys on Computational Mechanics*, pages 237–270, New York, 1989. ASME.

[98] J. McQueen, M. Meila, J. VanderPlas, and Z. Zhang. Megaman: Manifold Learning with Millions of Points. *arXiv: 1603.02763*, Mar. 2016.

[99] M. Meilă and W. Pentney. Clustering by Weighted Cuts in Directed Graphs. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Apr. 2007.

[100] S. Mikhlin. On the Schwarz Algorithm. In *Doklady Akad. Nauk SSSR (N.S.)*, volume 77, pages 569–571, 1951.

[101] K. Miller. Numerical Analogs to the Schwarz Alternating Procedure. *Numerische Mathematik*, 7(2):91–103, Apr. 1965.

[102] R. V. Mises and H. Pollaczek-Geiringer. Praktische Verfahren der Gleichungsauflösung. *ZAMM - Zeitschrift für Angewandte Mathematik und Mechanik*, 9(2):152–164, 1929.

[103] A. P. Motamed and B. Bahrak. Quantitative Analysis of Cryptocurrencies Transaction Graph. *Applied Network Science*, 4(1), Dec. 2019.

[104] O. Møyner and K.-A. Lie. A Multiscale Method Based on Restriction-Smoothed Basis Functions Suitable for General Grids in High Contrast Media. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2015.

[105] O. Møyner and K.-A. Lie. A Multiscale Restriction-Smoothed Basis Method for High Contrast Porous Media Represented on Unstructured Grids. *Journal of Computational Physics*, 304:46–71, Jan. 2016.

[106] B. Nadler, S. Lafon, I. Kevrekidis, and R. Coifman. Diffusion Maps, Spectral Clustering and Eigenfunctions of Fokker-Planck Operators. *Advances in neural information processing systems*, 18, 2005.

[107] A. Napov and Y. Notay. An Efficient Multigrid Method for Graph Laplacian Systems. *Electronic Transactions on Numerical Analysis*, 45:201–218, 2016.

[108] A. Napov and Y. Notay. An Efficient Multigrid Method for Graph Laplacian Systems II: Robust Aggregation. *SIAM Journal on Scientific Computing*, 39(5):S379–S403, Jan. 2017.

[109] K. Neymeyr. Solving Mesh Eigenproblems with Multigrid Efficiency. *Numerical Methods for Scientific Computing.*, 2003.

[110] A. Y. Ng, M. I. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an Algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[111] Y. Notay. Aggregation-Based Algebraic Multilevel Preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 27(4):998–1018, Jan. 2006.

[112] Y. Notay. An Aggregations-Based Algebraic Multigird Method. *Electronic Transaction on Numerical Analysis*, 37:123–146, 2010.

[113] Y. Notay and P. S. Vassilevski. Recursive Krylov-Based Multigrid Cycles. *Numerical Linear Algebra with Applications*, 15(5):473–487, 2008.

[114] D. Oeltz. Private Communication. 2023.

[115] C. W. Oosterlee and T. Washio. An Evaluation of Parallel Multigrid as a Solver and a Preconditioner for Singularly Perturbed Problems. *SIAM Journal on Scientific Computing*, 19(1):87–110, Jan. 1998.

[116] C. W. Oosterlee and T. Washio. Krylov Subspace Acceleration of Nonlinear Multigrid with Application to Recirculating Flows. *SIAM Journal on Scientific Computing*, 21(5):1670–1690, Jan. 2000.

[117] C. C. Paige and M. A. Saunders. LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, Mar. 1982.

[118] R. Plemmons. M-matrix Characterizations. I - Nonsingular M-matrices. *Linear Algebra and its Applications*, 18(2):175–188, 1977.

[119] H.-J. Plum, A. Krechel, S. Gries, B. Metsch, F. Nick, M. A. Schweitzer, and K. Stüben. Parallel Algebraic Multigrid. In *Scientific Computing and Algorithms in Industrial Simulations: Projects and Products of Fraunhofer SCAI*, pages 121–134. Springer International Publishing, 2017.

[120] A. Quarteroni and A. Valli. *Domain Decomposition Methods for Partial Differential Equations*. Oxford University Press, 1999.

[121] J. E. Roman. Practical Implementation of Harmonic Krylov-Schur. Technical Report STR-9, Universitat Politècnica de València, 2009. Available at https://slepc.upv.es.

[122] S. T. Roweis and L. K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, Dec. 2000.

[123] J. Ruge. *Multigrid Methods for Differential Eigenvalue and Variational Problems and Unigrid for Multigrid Simulation*. PhD thesis, Colorado State University, 1981.

[124] J. W. Ruge and K. Stüben. Algebraic Multigrid. In *Multigrid Methods*, pages 73–130. Society for Industrial and Applied Mathematics, Jan. 1987.

[125] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Jan. 2003.

[126] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics, Jan. 2011.

[127] Y. Saad and M. H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[128] L. K. Saul and S. T. Roweis. An Introduction to Locally Linear Embedding. Available at: http://www.cs.columbia.edu/ jebara/6772/papers/lleintro.pdf.

[129] B. Schölkopf, A. Smola, and K.-R. Müller. Kernel Principal Component Analysis. In *Lecture Notes in Computer Science*, pages 583–588. Springer Berlin Heidelberg, 1997.

[130] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319, July 1998.

[131] P. Schmid and J. Sesterhenn. Dynamic Mode Decomposition of Numerical and Experimental Data. In *Bulletin of the American Physical Society, Sixty-First Annual Meeting of the APS Division of Fluid Dynamics.*, volume 53, 2008.

[132] P. J. Schmid. Dynamic Mode Decomposition of Numerical and Experimental Data. *Journal of Fluid Mechanics*, 656:5–28, July 2010.

[133] H. A. Schwarz. *Über einen Grenzübergang durch alternierendes Verfahren*, pages 272–286. Zürich und Furrer, 1870.

[134] A. Sharma, A. Agrawal, A. Bhatia, and K. Tiwari. Bitcoin's Blockchain Data Analytics: A Graph Theoretic Perspective. In *Advanced Information Networking and Applications*, pages 459–470. Springer International Publishing, 2022.

[135] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[136] Software Library SAMG. https://www.scai.fraunhofer.de/en/business-research-areas/fast-solvers/products/samg.html.

[137] K. Stüben. An Introduction to Algebraic Multigrid. In *Multigrid*, pages 413–532. Academic Press, 2000.

[138] K. Stüben. A Review of Algebraic Multigrid. *Journal of Computational and Applied Mathematics*, 128(1-2):281–309, Mar. 2001.

[139] K. Stüben, J. W. Ruge, T. Clees, and S. Gries. Algebraic Multigrid: From Academia to Industry. In *Scientific Computing and Algorithms in Industrial Simulations: Projects and Products of Fraunhofer SCAI*, pages 83–119. Springer International Publishing, 2017.

[140] H. D. Sterck, R. D. Falgout, J. W. Nolting, and U. M. Yang. Distance-Two Interpolation for Parallel Algebraic Multigrid. *Journal of Physics: Conference Series*, 78:012017, July 2007.

[141] H. D. Sterck, U. M. Yang, and J. J. Heys. Reducing Complexity in Parallel Algebraic Multigrid Preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):1019–1039, Jan. 2006.

[142] G. Stewart. Pertubation Bounds for the Definite Generalized Eigenvalue Problem. *Linear Algebra and its Applications*, 23:69–85, Feb. 1979.

[143] G. W. Stewart. A Krylov–Schur Algorithm for Large Eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23(3):601–614, Jan. 2002.

[144] G. W. Stewart. Addendum to "A Krylov–Schur Algorithm for Large Eigenproblems". *SIAM Journal on Matrix Analysis and Applications*, 24(2):599–601, Jan. 2002.

[145] R. Tamstorf, T. Jones, and S. F. McCormick. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Transactions on Graphics*, 34(6):1–13, Nov. 2015.

[146] M. Ţene, M. S. A. Kobaisi, and H. Hajibeygi. Algebraic Multiscale Method for Flow in Heterogeneous Porous Media with Embedded Discrete Fractures (F-AMS). *Journal of Computational Physics*, 321:819–845, Sept. 2016.

[147] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, Dec. 2000.

[148] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid*. Elsevier, 2000.

[149] R. Tuminaro and C. Tong. Parallel Smoothed Aggregation Multigrid : Aggregation Strategies on Massively Parallel Machines. In *ACM/IEEE SC 2000 Conference*. IEEE, 2000.

[150] H. A. van der Vorst. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644, Mar. 1992.

[151] P. Vaněk, M. Brezina, and J. Mandel. Convergence of Algebraic Multigrid Based on Smoothed Aggregation. *Numerische Mathematik*, 88(3):559–579, May 2001.

[152] P. Vaněk, J. Mandel, and M. Brezina. Algebraic Multigrid on Unstructured Meshes. *UCD/CCM Report*, 34(123–146), 1994.

[153] P. Vaněk, J. Mandel, and M. Brezina. Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems. *Computing*, 56(3):179–196, Sept. 1996.

[154] P. Vaněk and I. Pultarová. Convergence Theory for the Exact Interpolation Scheme with Approximation Vector as the First Column of the Prolongator and Rayleigh Quotient Iteration Nonlinear Smoother. *Applications of Mathematics*, 62(1):49–73, Jan. 2017.

[155] P. S. Vassilevski. Sparse Matrix Element Topology with Application to AMG(e) and Preconditioning. *Numerical Linear Algebra with Applications*, 9(6-7):429–444, 2002.

[156] U. von Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, 17(4):395–416, Aug. 2007.

[157] C. Vuik. New Insights in GMRES-like Methods with Variable Preconditioners. *Journal of Computational and Applied Mathematics*, 61(2):189–204, July 1995.

[158] Y. Wang, H. Hajibeygi, and H. A. Tchelepi. Algebraic Multiscale Solver for Flow in Heterogeneous Porous Media. *Journal of Computational Physics*, 259:284–303, Feb. 2014.

[159] A. Weinstein and W. Stenger. *Methods of Intermediate Problems for Eigenvalues: Theory and Ramifications*. Elsevier Science & Techn., June 1972.

[160] P. Wesseling. Introduction to Multigrid Methods. Technical report, Institute for Computer Applications in Science and Engineering Hampton VA, 1995.

[161] H. Wielandt. Beiträge zur mathematischen Behandlung komplexer Eigenwertprobleme, Teil V: Bestimmung höherer Eigenwerte durch gebrochene Iteration. *Bericht B44/J/37*, 1944.

[162] U. M. Yang. Parallel Algebraic Multigrid Methods — High Performance Preconditioners. In *Numerical solution of partial differential equations on parallel computers*, pages 209–236. Springer-Verlag, 2006.

[163] H. Zhou. *Algebraic Multiscale Finite-Volume Methods for Reservoir Simulation*. PhD thesis, Stanford University, 2010.