LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-553254

Adaptive Algebraic Multigrid for Finite Element Elliptic Equations with Random Coefficients

D. Kalchev

April 30, 2012

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Sofia University
Faculty of Mathematics and Informatics
Department of Numerical Methods and Algorithms

Master Thesis

Adaptive Algebraic Multigrid for Finite Element Elliptic Equations with Random Coefficients

Delyan Kalchev
Computational Mathematics and Mathematical Modeling, M23012
delyank@gmail.com

Thesis advisor
Dr. Panayot S. Vassilevski,
Center for Applied Scientific Computing,
Lawrence Livermore National Laboratory,
vassilevski1@llnl.gov, http://people.llnl.gov/vassilevski1

Reviewer
Prof. Stefka Dimova,
Department of Numerical Methods and Algorithms,
Faculty of Mathematics and Informatics, Sofia University,
dimova@fmi.uni-sofia.bg,
http://www.fmi.uni-sofia.bg/en/lecturers/nummeth/dimova

Sofia, Bulgaria
2012

This page is intentionally left blank.

Abstract

This thesis presents a two-grid algorithm based on Smoothed Aggregation Spectral Element Agglomeration Algebraic Multigrid (SA-ρAMGe) combined with adaptation. The aim is to build an efficient solver for the linear systems arising from discretization of second-order elliptic partial differential equations (PDEs) with stochastic coefficients. Examples include PDEs that model subsurface flow with random permeability field. During a Markov Chain Monte Carlo (MCMC) simulation process, that draws PDE coefficient samples from a certain distribution, the PDE coefficients change, hence the resulting linear systems to be solved change. At every such step the system (discretized PDE) needs to be solved and the computed solution used to evaluate some functional(s) of interest that then determine if the coefficient sample is acceptable or not. The MCMC process is hence computationally intensive and requires the solvers used to be efficient and fast. This fact that at every step of MCMC the resulting linear system changes, makes an already existing solver built for the old problem perhaps not as efficient for the problem corresponding to the new sampled coefficient. This motivates the main goal of our study, namely, to adapt an already existing solver to handle the problem (with changed coefficient) with the objective to achieve this goal to be faster and more efficient than building a completely new solver from scratch. Our approach utilizes the local element matrices (for the problem with changed coefficients) to build local problems associated with constructed by the method agglomerated elements (a set of subdomains that cover the given computational domain). We solve a generalized eigenproblem for each set in a subspace spanned by the previous local coarse space (used for the old solver) and a vector, component of the error, that the old solver cannot handle. A portion of the spectrum of these local eigenproblems (corresponding to eigenvalues close to zero) form the coarse basis used to define the new two-level method of our interest. We illustrate the performance of this adaptive two-level procedure with a large set of numerical experiments that demonstrate its efficiency over building the solvers from scratch.

Keywords: algebraic multigrid (AMG), aggregation AMG, spectral element-based AMG (AMGe), element agglomeration, adaptive solvers, stochastic PDEs

Part of the work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

This page is intentionally left blank.

Contents

List of Figures

This page is intentionally left blank.

List of Tables

1

Introduction

The main purpose of this introductory chapter is to present the structure of the thesis, formulate the problem, and summarize some basic facts related to the finite element method and the resulting stiffness matrices.

In Section 1.1, we motivate the need for introducing randomness in the coefficients of the partial differential equations (or PDEs). We also specify the PDE coefficients and the difficulties they pose for the linear solver. Section 1.2 is devoted to a short outline of the structure of the thesis. In Section 1.3 we formulate the problem of our main interest. Section 1.4 provides a summary of facts about the finite element discretization we use and the properties of the resulting stiffness matrices.

1.1 Motivation

When a mathematical model is used in practice the parameters of the model are obtained by real-world measurements. The data collected by measurements cannot cover the whole spatial area, especially when the area is large, but measurements are conducted only at a number of locations. However, the model requires input data for the whole domain. This acts as a source of uncertainty in the input data which has impact on the quality of the output from solving the model (see [15]). The impact to the result is relevant, since the purpose of modeling is as realistic as possible representations of the reality. A way to reflect and deal with the uncertainty in the input data is to consider the parameters of the model random which results in randomness in the solution.

We consider elliptic PDEs with random coefficients, more specifically, we consider diffusion equation with a coefficient representing random *permeability field*. Such PDEs are typically solved by a *Monte Carlo (MC) method*. The random coefficient is simulated (many realizations of the coefficient are generated) and for each such realization the resulting deterministic PDE is solved numerically and hence (approximate) realizations of the solution are obtained. Thus, a resulting Monte Carlo simulation of the solution (or a related *quantity of interest*) is produced. The MC methods for the efficient simulation of the desired quantities of interest, and the ways the final desired results are assessed are separate questions (see [15]) that we do not consider in this study. Our main interest is in building an efficient method for solving the many arising

systems of linear equations which is the most computationally intensive part of the MC simulation.

There are several requirements on the solver that we want to construct that are imposed due to the fact of changing PDE coefficients. First, even a single sampled coefficient (which is a function over the computational domain) may be highly variable, i.e., it may oscillate with a large amplitude. We also want the method to handle discontinuous coefficients which may have large jumps (contrast). Other requirements are the capability to be ready to work with *irregular meshes* both in 2D and 3D. Secondly, once an efficient solver is constructed for one coefficient sample, we want to be able to utilize as much as possible that solver for a new (in some sense nearby) coefficient sample by *adapting* the old solver. This is a main challenge that we address in this study.

The method we implement is an *AMG (algebraic multigrid)* solver which in this study is only a *two-grid* one. The solver changes adaptively which means that once an AMG solver that is efficient for one realization of the coefficient has been built, it is later being adapted for a subsequent realization in contrast to building a new solver from scratch. For the purpose of this adaptivity to be efficient, we assume that the method for simulating the random coefficient is such that consequent realizations are "close" in some sense. The latter is the case in the so-called *Markov Chain Monte Carlo* simulations (cf., e.g., [16]). Of course, when we say we build a solver we mean building a (two-level) *hierarchy* and in AMG this hierarchy depends on the operator (matrix) and hence on the random coefficient.

We have implemented a serial (non-parallel) version of the method. Our implementation makes use of several computational software packages: MFEM [1] and its companion GLVIS [2], LAPACK, METIS [3], ARPACK [4] and it counterpart ARPACK++.

The implemented method, and also this thesis, is a result of an internship at Lawrence Livermore National Laboratory (LLNL) in the summer of 2011 under the supervision of Panayot Vassilevski[1] and with the great help of Christian Ketelsen[2] and Ilya Lashuk[3].

1.2 Outline of the Presentation

The thesis is structured in a few chapters and here we give a short layout of their contents.

Chapter 2 introduces some basic facts from numerical linear algebra. It also gives short information on how meshes can be considered as graphs (or relation tables).

[1]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, vassilevski1@llnl.gov, http://people.llnl.gov/vassilevski1.

[2]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, ketelsen1@llnl.gov, http://people.llnl.gov/ketelsen1.

[3]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, lashuk2@llnl.gov, http://people.llnl.gov/lashuk2.

Chapter 3 presents a quite basic and intuitive description of multigrid, including algebraic multigrid, and a description of the method we use.

Chapter 4 describes the details of the specific method used in an algorithm- and implementation-oriented manner. The method and its implementation are illustrated with numerical results from our experiments with the studied solver. We also point out to some key parts of the implementation which if modified approrpiatele may lead to better performance. The chapter finishes with conclusions and ideas for further development and improvements.

1.3 Problem Formulation

We now formulate the problem of our main interest.

Let Ω be the computational domain in $\mathbb{R}^2$ or $\mathbb{R}^3$ (polygon, in 2D, or polytope, in 3D) and let $\partial\Omega = \Gamma_{\mathfrak{D}} \cup \Gamma_{\mathfrak{N}}$, where $\Gamma_{\mathfrak{D}}$ has nonzero measure. Let $\mathbf{n}$ be the unit outward normal vector to the domain boundary.

For a given right-hand side f and boundary data g, we want to solve for u the second-order diffusion equation

$$-\operatorname{div}\left(k\left(x,\theta\right)\nabla u\left(x,\theta\right)\right) = f\left(x\right) \qquad \text{in } \Omega\times\mathcal{S}, \tag{1.1a}$$

$$u\left(x,\theta\right) = g(x) \qquad \text{on } \partial\Gamma_{\mathfrak{D}}\times\mathcal{S}, \tag{1.1b}$$

$$\frac{\partial u}{\partial \mathbf{n}} = 0 \qquad \text{on } \partial\Gamma_{\mathfrak{N}}\times\mathcal{S}. \tag{1.1c}$$

Here, the *conductivity field* $k\left(x,\theta\right) > 0$ for all $x \in \Omega$ and $\theta \in \mathcal{S}$. We assume that $f \in L_2\left(\Omega\right)$. The argument θ is a placeholder representing the stochastic dependence in k and u. $\mathcal{S}$ is just a notation for the "stochastic space."

We note that, in general, f (and also g) may also be subject to uncertainty but, as far as the linear solver is concerned, only k affects the matrices of the linear systems to be solved and, since the properties of these matrices influence the construction of the solver, we consider f and g to be deterministic.

We use the *finite element method (FEM)* for discretizing every single deterministic problem (i.e., for a given parameter θ). We consider *triangulation of* Ω, i.e., a *partitioning* of Ω into triangles (in 2D) or in tetrahedra (in 3D), and we consider piecewise linear, globally continuous finite elements (commonly denoted as $\mathcal{P}_1$ elements). In the case of piecewise linear functions, we may assume (without loss of generality) that the given coefficient k is a piecewise constant function on Ω such that it is constant on each *element* of the triangulation of Ω, for example an average of k on each element is taken. That is, the coefficient k is generally discontinuous.

The FEM part is readily implemented in the MFEM (see [1]) package which we have used for the work presented in this study.

1.4 Finite Element Discretization

This section presents a summary of finite element related facts that are relevant to our study.

We start by introducing some notation. Then, we give the weak formulation

of the problem, the finite elements used and the systems to be solved. Also, we outline the main properties of the resulting stiffness matrices. Finally, we show the meshes and the domain that we used in the numerical experiments to follow.

1.4.1 Notation

Here we introduce some notation.

We consider column vectors and they are denoted in boldface, e.g., the vector $\mathbf{v}$. The i-th component of vector $\mathbf{v}$ is denoted v_i.

The elements of a matrix A are denoted by the corresponding small letter, a_{ij}, which denotes the element of A in its i-th row and j-th column.

Here is a list of notation frequently used in what follows:

$$\mathbf{u}\cdot\mathbf{v}=\sum_{i=1}^{r}u_iv_i,\ \text{for } \mathbf{u},\mathbf{v}\in\mathbb{R}^r,$$

$$V_p=\left\{v\in H^1(\Omega):v=p \text{ on } \Gamma_{\mathfrak{D}}\right\},$$

$$a_k(u,v)=\int_\Omega k\nabla u\cdot\nabla v\,\mathrm{d}\mathbf{x},\ \text{for } u,v\in H^1(\Omega),\ k \text{ – conductivity field},$$

$$(u,v)=(u,v)_0=\int_\Omega uv\,\mathrm{d}\mathbf{x},\ \text{for } u,v\in L^2(\Omega),$$

$$\|u\|=\|u\|_0=\sqrt{(u,u)_0} \text{ and } L_2(\Omega)=\{u:\|u\|_0<\infty\}.$$

Assuming that u has all its first-order *weak derivatives*:

$$|u|_1=\|\nabla u\|_0=\int_\Omega \nabla u\cdot\nabla u\,\mathrm{d}\mathbf{x} \text{ (a } \textit{semi-norm}),$$

$$\|u\|_1=\sqrt{\|u\|_0^2+\|\nabla u\|_0^2} \text{ and } H^1(\Omega)=\{u:\|u\|_1<\infty\}.$$

Assuming u has all its weak derivatives up to second order:

$$\|u\|_2=\sqrt{\|u\|_1^2+|u|_2^2},$$

where the semi-norm $|u|_2^2$ denotes the sum of the squares of the L_2-norms of all second-order weak derivatives of u.

Assume we are given a tessellation $\mathcal{T}$ of Ω. We denote that τ is an element of the tessellation $\mathcal{T}$ by $\tau\in\mathcal{T}$. Then

$$h_\tau=\text{the longest side (edge) of } \tau\in\mathcal{T},$$

$$h=\max_{\tau\in\mathcal{T}}h_\tau.$$

To explicitly relate the tessellation, $\mathcal{T}$, to its *size*, h, we write $\mathcal{T}_h=\mathcal{T}$.

Finally, we introduce the local (element) *bilinear form*

$$a_k^\tau(u,v)=\int_\tau k\nabla u\cdot\nabla v\,\mathrm{d}\mathbf{x},\ \text{for } u,v\in H^1(\Omega),\tau\in\mathcal{T}_h.$$

Remark 1.1. Since we assumed that k is constant on each element, then

$$a_k^\tau(u,v) = k_\tau \int_\tau \nabla u \cdot \nabla v \, \mathrm{d}\mathbf{x},$$

where k_τ is the value of k on τ. That is, *the element stiffness matrix* (to be introduced later on) for τ will simply be the element stiffness matrix computed for $a_1^\tau(\cdot,\cdot)$ (the bilinear form for the *Poisson's equation*) multiplied by the constant k_τ.

1.4.2 Weak Formulation

The finite element method for discretizing the different coefficient realizations of the stochastic PDE uses the *weak formulation* of the problem.

The weak formulation of (1.1) simply reads (see [19]): Find $u \in V_g$ such that (s.t.)

$$a_k(u,v) = (f,v), \text{ for all } v \in V_0.$$

1.4.3 Finite Elements

Our efforts are devoted to solving the linear systems of equations arising from the discretizations of many realizations of the stochastic PDE (1.1). The systems are obtained as a result of discretizing the continuous PDE for any given coefficient realization. We briefly describe the finite element discretization and introduce the resulting linear systems.

Let the tessellation $\mathcal{T}_h$ of Ω have n vertices (nodes) with coordinates $\mathbf{x}_i$, $i = 1, \ldots, n$, including nodes on the boundary $\partial\Omega$. We consider the standard *Lagrangian basis functions (nodal basis)* ϕ_i, $i = 1, \ldots, n$, i.e., $\phi_i(\mathbf{x}_j) = 1$ when $i = j$ and 0 otherwise. That is, we identify the *degrees of freedom (dofs)* with the vertices. We also split the set of vertices, $\mathcal{N}_h$, into two sets: $\mathcal{N}_\mathfrak{D}$ – those on $\Gamma_\mathfrak{D}$ and $\mathcal{N}_\iota$ – the rest. The finite element approximation of the solution of (1.1) can be written as

$$u_h(\mathbf{x}) = \sum_{\mathbf{x}_j \in \mathcal{N}_\iota} u_j \phi_j(\mathbf{x}) + \sum_{\mathbf{x}_j \in \mathcal{N}_\mathfrak{D}} u_j \phi_j(\mathbf{x}),$$

where we have set $u_j = g(\mathbf{x}_j)$, $\mathbf{x}_j \in \mathcal{N}_\mathfrak{D}$ since these are given (known). Substituting u_h into the weak formulation, we get the following linear system of equations for the unknown coefficients $\{u_j\}_{\mathbf{x}_j \in \mathcal{N}_\iota}$

$$\sum_{\mathbf{x}_j \in \mathcal{N}_\iota} u_j a_k(\phi_j, \phi_i) = (f, \phi_i) - \sum_{\mathbf{x}_j \in \mathcal{N}_\mathfrak{D}} g(\mathbf{x}_j) \, a_k(\phi_j, \phi_i), \mathbf{x}_i \in \mathcal{N}_\iota. \tag{1.2}$$

Obviously $\phi_i \in V_0$ for $\mathbf{x}_i \in \mathcal{N}_\iota$. The bilinear form $a_k(\cdot,\cdot)$ is clearly *symmetric* and also, it is *positive definite* on V_0. That is, when $a_k(u,u) = 0$ then $\nabla u = 0$, hence u is a constant and the fact that $u = 0$ on $\Gamma_\mathfrak{D}$ implies $u = 0$. (Of course, the equality signs here have the meaning of *almost everywhere*.)

If $m = |\mathcal{N}_\iota|$ then we can define the $m \times m$ *(global) stiffness matrix* A with entries $a_{ij} = a_k(\phi_j, \phi_i)$, where we consider only the nodal basis functions corresponding to the nodes in $\mathcal{N}_\iota$. If, also, $\mathbf{u} = (u_i)_{\mathbf{x}_i \in \mathcal{N}_\iota}$ is a vector in $\mathbb{R}^m$ and

we denote the right hand side of (1.2) by b_i, $i = 1, \ldots, m$, (1.2) takes the form

$$A\mathbf{u} = \mathbf{b}, \tag{1.3}$$

where $\mathbf{b}$ is the vector with components b_i, $i = 1, \ldots, m$. Clearly, the solution of (1.3) gives the finite element approximation of the solution of (1.1).

We have the following *error estimates* (see [6, 19, 32])

$$\|u - u_h\|_1 \leq Ch\|u\|_2 \text{ hence } \|\nabla(u - u_h)\| \leq Ch\|u\|_2,$$
$$\|u - u_h\| \leq Ch^2\|u\|_2,$$

where $C > 0$ is a *generic* constant. The last, L_2, error estimate holds if the PDE is H^2*-regular*, i.e., for any r.h.s., $f \in L_2(\Omega)$, the solution of the PDE, u, is in $H^2(\Omega)$.

Remark 1.2. Every FEM related tool that we need (e.g., the stiffness matrix assembly, building the linear system (1.3) to be solved, mesh refinement etc.) is available in the MFEM package.

1.4.4 The Stiffness Matrix

This section presents properties of the stiffness matrix that are most relevant for the method we use.

We first show the symmetry and the positive definiteness of the stiffness matrix and next we refer to the sparsity of the matrix.

1.4.4.1 Symmetry and Positive Definiteness

Here we show the *symmetry* and *positive definiteness* of the stiffness matrix.

Definition 1.3. A *symmetric* matrix Υ, i.e., $\upsilon_{ij} = \upsilon_{ji}$, is called *(symmetric) positive definite or s.p.d.* if $\mathbf{v}^T\Upsilon\mathbf{v} > 0$ for every nonzero vector $\mathbf{v}$. Equivalently, a symmetric matrix is positive definite if all its *eigenvalues* are strictly positive (of course, it is well-known fact that the eigenvalues of a symmetric matrix are real).

Clearly, a s.p.d. matrix defines an inner product. For a s.p.d. matrix Υ we denote the corresponding inner product by $(\mathbf{v}, \mathbf{w})_\Upsilon = \mathbf{w}^T\Upsilon\mathbf{v}$ and the induced norm $\|\mathbf{v}\|_\Upsilon = \sqrt{(\mathbf{v}, \mathbf{v})_\Upsilon}$. We call them respectively *energy inner product* and *energy norm*.

The s.p.d. property of the bilinear form implies the s.p.d. property of the stiffness matrix. Namely, the bilinear form may be considered as an inner product in V_0, hence the stiffness matrix A is a *Gram matrix* for linear independent functions (vectors). The last implies the s.p.d. property of A.

Remark 1.4. Instead of taking only the nodal basis functions corresponding to nodes in $\mathcal{N}_\iota$, as in (1.2), and building the stiffness matrix as a $m \times m$ matrix, it is handy to take all the basis functions and build a $n \times n$ matrix and afterwards impose the *essential boundary conditions* (Dirichlet boundary conditions) on the matrix. The result is essentially the same. That is why, in what follows we

will not distinguish between m and n and will hence consider A as an $n \times n$ matrix.

Remark 1.5. If $\Gamma_{\mathfrak{D}}$ has zero measure, i.e., $\mathcal{N}_\iota = \mathcal{N}_h$, then the positive *semi-definiteness* of the bilinear form implies the positive semi-definiteness of A and $\mathbf{x}^T A\mathbf{x} = 0$ iff (if and only if) $\mathbf{x}$ is a constant vector.

1.4.4.2 Sparsity

A main property of the finite element method is that it leads to the following important property of the resulting stiffness matrix, namely, its *sparsity*.

Definition 1.6. A $n \times n$ matrix is called *sparse* if the number of nonzero elements per column and per row is $O(1)$ as $n \to \infty$ (or, in the finite element case, as $h \to 0$).

As a consequence, the number of nonzero elements in a sparse matrix is $O(n)$. Quite naturally, to represent a sparse matrix we need $O(n)$ *space (memory)* and this makes it possible to implement standard matrix vector operations (sparse matrix transposition, sparse matrix vector multiplication, sparse matrix times sparse matrix etc.) so that they can be computed in $O(n)$ *time (operations)*. This is achieved by the so called *compressed sparse row (CSR) format* (see [23], and also [24]) or its dual *compressed sparse column (CSC) format*. This is a crucial prerequisite for building *optimal* solvers, i.e., solvers that use linear ($O(n)$) space and time.

Remark 1.7. The MFEM package possesses an implementation of the CSR format and the operations based on it. It also has classes for operations with vectors and *dense* matrices. As a whole, it has a quite good tool-set enabling implementation of variety of linear solvers.

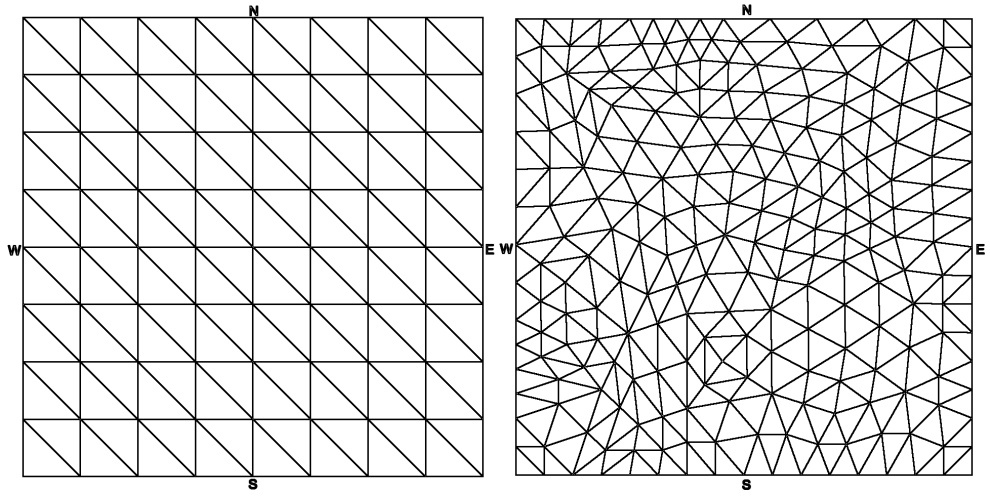
The local support of the nodal basis functions implies that the sparsity structure of the stiffness matrix is determined by the topology of the mesh. That is, a_{ij} is nonzero iff $\mathbf{x}_j$ and $\mathbf{x}_i$ are vertices of a common element. The latter implies that if the number of the neighbors for each node is kept uniformly bounded as $h \to 0$, which may be achieved by keeping the angles of the elements bounded away from zero, then the stiffness matrix remains sparse.

Definition 1.8. A family of partitions $\{\mathcal{T}_h\}$ is called *shape regular* if there is an independent of h constant $\beta > 0$ s.t. $\rho_\tau \geq \beta h_\tau$ for all $\tau \in \mathcal{T}_h$, where ρ_τ is the diameter of the inscribed in τ circle (or sphere).

That is, the elements cannot be arbitrary thin which guarantees the sparsity of A. Thus, we assume we have shape regular partitions.

1.4.5 Domain and Meshes for the Experiments

We end this introductory chapter by introducing the meshes and the domain used for the numerical experiments we have performed for this study.



(a) The regular mesh **(b)** The irregular mesh

Figure 1.1: Used mashes.

The numerical experiments are in 2D with computational domain $\Omega = (0,1) \times (0,1)$. We use *regular* mesh shown in fig. 1.1a whereas in fig. 1.1b we show the *irregular (unstructured)* mesh we use. Both meshes are possibly subject to one or more steps of *uniform refinement*, i.e., each triangle is split into four congruent ones by connecting the midpoint of their edges. The sides of the unit square are denoted E (east), W (west), N (north), S (south) and respectively $\partial\Omega = \Gamma_E \cup \Gamma_W \cup \Gamma_N \cup \Gamma_S$.

2

Basic Linear Algebra Facts

This chapter introduces a few basic facts from numerical linear algebra that we need and describes the way finite element meshes can be viewed as graphs which is useful for partitioning the elements into agglomerated elements and the vertices into aggregates.

In Section 2.1 we recall the notion of matrix conditioning and we point out the ill-conditioning of the finite element matrices that we deal with. Section 2.2 discusses a few facts about stationary iterative methods relevant to our presentation. Section 2.3 is devoted to the classical singular value decomposition, and Section 2.4 to the Gram-Schmidt orthogonalization, both of which we use. In Section 2.5, we describe ways to view a mesh as a graph which is later relevant to the implementation of our method.

2.1 Condition Number

We start by recalling the important notion of *condition number* in the case of s.p.d. matrices.

Definition 2.1. Let Υ be a s.p.d. matrix. We call

$$\kappa(\Upsilon) = \frac{\lambda_{max}(\Upsilon)}{\lambda_{min}(\Upsilon)},$$

the *(spectral) condition number* of Υ. Here, $\lambda_{max}(\Upsilon)$ and $\lambda_{min}(\Upsilon)$ are the maximal and the minimal eigenvalues of Υ respectively.

Also, we recal that for a s.p.d. matrix Υ, we have the following property of the extreme values of the *Rayleigh quotient*:

$$\lambda_{min}(\Upsilon) = \min_{\mathbf{v}} \frac{\mathbf{v}^T \Upsilon \mathbf{v}}{\mathbf{v}^T \mathbf{v}},$$
$$\lambda_{max}(\Upsilon) = \max_{\mathbf{v}} \frac{\mathbf{v}^T \Upsilon \mathbf{v}}{\mathbf{v}^T \mathbf{v}}.$$

In general, matrices of linear systems arising from finite element (or finite difference) discretizations of elliptic PDEs are *ill-conditioned*, i.e., $\kappa(A) \to \infty$ as $h \to 0$. For example, for the Poisson's equation ($-\Delta u = f$) in 2D, on a *quasi-uniform* mesh (shape regular mesh for which there exists a mesh-independent constant $\gamma > 0$ s.t. $h_\tau \geq \gamma h$ for all $\tau \in \mathcal{T}_h$) the condition number of the stiffness matrix can be estimated as $O(h^{-2})$ which is asymptotically sharp (see

[19]). The presence of a discontinuous coefficient with large jumps will make the condition of the matrix even worse. This makes the problem of solving (1.3) computationally challenging in the case of large-scale matrices (i.e., for $h \to 0$).

2.2 Stationary Iterative Methods

In this section we summarize some facts about the stationary iterative methods.

We start with a definition and a basic convergence result. Next, we discuss convergence in A-norm (i.e., A-convergence). We finish the section with a short description of the notions of convergence factor, stalled convergence, and spectral equivalence.

2.2.1 Definition of Stationary Iterative Methods and Convergence

We first present the definition of a stationary iterative method. We also give a characterization of its convergence.

Consider the linear system

$$A\mathbf{x} = \mathbf{b}, \tag{2.1}$$

where A is a $n \times n$ s.p.d. matrix. Let M be a $n \times n$ *easily invertible* matrix, i.e., linear systems $M\mathbf{y} = \mathbf{g}$ are (computationally) easy to solve. Examples for M are diagonal, lower or upper triangular matrices with $O(1)$ nonzero entries per row, etc.

Definition 2.2. For a $n \times n$ matrix C with eigenvalues $\lambda_1, \ldots, \lambda_n$ we define the *spectral radius* $\rho(C) = \max\limits_{i=1,\ldots,n} |\lambda_i|$. (Clearly, $\rho(C) = \rho(C^T)$.)

For a given *initial iterate (or approximation, or guess)* $\mathbf{x}_0$, we consider the *iteration process*

$$M\left(\mathbf{x}_{k+1} - \mathbf{x}_k\right) = \mathbf{b} - A\mathbf{x}_k, \text{ for } k = 0, 1, \ldots,$$

or, equivalently,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}\mathbf{r}_k,$$

or, equivalently,

$$\mathbf{x}_{k+1} = \left(I - M^{-1}A\right)\mathbf{x}_k + M^{-1}\mathbf{b},$$

where $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ is the *residual (defect)* and $E_M = I - M^{-1}A$ is the *iteration matrix*. Clearly, the exact solution of (2.1) $\mathbf{x}$ is a *fixed point* for the iteration process. M is called a *preconditioner*.

If we denote the *(algebraic) error* $\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k$, we have $A\mathbf{e}_k = \mathbf{r}_k$ (i.e., the error solves the *defect equation*) and, also, $\mathbf{e}_{k+1} = E_M\mathbf{e}_k$. Hence, $\mathbf{e}_k = E_M^k\mathbf{e}_0$. Thus, the iteration process is convergent for arbitrary $\mathbf{x}_0$ iff $\lim\limits_{k\to\infty} E_M^k = O$. For the convergence it is sufficient to have $\|E_M\| < 1$ for some norm $\|\cdot\|$. In general, it is not a necessary condition for convergence. The convergence is characterized by the spectral radius of the iteration matrix, i.e., we have (see

[5, 23, 12]):

Theorem 2.3. $\lim_{k\to\infty} E_M^k = O$ *iff* $\rho(E_M) < 1$, *i.e., the iteration process is convergent for any initial guess iff* $\rho(E_M) < 1$.

The theorem above holds not only for iteration matrices of stationary iterative processes but for any square matrix in place of E_M.

For a vector $\mathbf{z}$, $E_M\mathbf{z}$ can be computed by one iteration with $\mathbf{x}_0 = \mathbf{z}$ and $\mathbf{b} = \mathbf{0}$ and actually this is the way to "simulate" the effect of the iteration process on the error. If we want to compute $M^{-1}\mathbf{z}$, then we run one iteration with $\mathbf{b} = \mathbf{z}$ and $\mathbf{x}_0 = \mathbf{0}$. Computing $M^{-1}\mathbf{r}_k$ is even easier, since $M^{-1}\mathbf{r}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$. Thus, we won't ever need to explicitly produce M^{-1} or E_M even if M is explicitly given. For complicated algorithms neither M nor M^{-1} may be explicitly available (or too expensive to form). Nevertheless, if the iteration process is computationally feasible (through perhaps a complicated algorithm) then its convergence can be readily studied as described above.

2.2.2 A-convergence

We consider the (monotone) convergence in A-norm (i.e., *A-convergence*) when the matrix A of the system is s.p.d.

We use the fact that for a symmetric $n \times n$ matrix Υ the *spectral decomposition* $\Upsilon = Q\Lambda Q^T$ holds, where Q is *orthogonal*, i.e., $Q^T = Q^{-1}$, and $\Lambda = \operatorname{diag}(\lambda_i)_{i=1}^n$ (λ_i are the eigenvalues of Υ) to define *square root of* Υ.

Definition 2.4. For s.p.d. Υ let $\Upsilon^{\frac{1}{2}} = Q\Lambda^{\frac{1}{2}}Q^T$, where $\Lambda^{\frac{1}{2}} = \operatorname{diag}\left(\sqrt{\lambda_i}\right)_{i=1}^n$. (Clearly, $\Upsilon = \Upsilon^{\frac{1}{2}}\Upsilon^{\frac{1}{2}}$.)

Consider the *Euclidean norm* $\|\cdot\|$. For a symmetric matrix Υ we have $\|\Upsilon\| = \rho(\Upsilon)$ and for an arbitrary matrix C we have $\|C\| = \sqrt{\rho(C^TC)}$.

From $\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^TA\mathbf{v}} = \|A^{\frac{1}{2}}\mathbf{v}\|$, using the invertibility of $A^{\frac{1}{2}}$, and denoting $\mathbf{w} = A^{\frac{1}{2}}\mathbf{v}$, we readily get

$$\sup_{\mathbf{v}\neq\mathbf{0}} \frac{\|(I - M^{-1}A)\mathbf{v}\|_A}{\|\mathbf{v}\|_A} = \sup_{\mathbf{v}\neq\mathbf{0}} \frac{\|A^{\frac{1}{2}}(I - M^{-1}A)\mathbf{v}\|}{\|A^{\frac{1}{2}}\mathbf{v}\|}$$
$$= \sup_{\mathbf{v}\neq\mathbf{0}} \frac{\|(I - A^{\frac{1}{2}}M^{-1}A^{\frac{1}{2}})A^{\frac{1}{2}}\mathbf{v}\|}{\|A^{\frac{1}{2}}\mathbf{v}\|} = \sup_{\mathbf{w}\neq\mathbf{0}} \frac{\|(I - A^{\frac{1}{2}}M^{-1}A^{\frac{1}{2}})\mathbf{w}\|}{\|\mathbf{w}\|}.$$

That is, $\|E_M\|_A = \|\mathcal{E}_M\|$, where $\mathcal{E}_M = I - A^{\frac{1}{2}}M^{-1}A^{\frac{1}{2}}$. Obviously, $(\mathcal{E}_M)^T = \mathcal{E}_{M^T}$.

Introduce the symmetric preconditioners $\overline{M} = M(M + M^T - A)^{-1}M^T$, $\widetilde{M} = M^T(M + M^T - A)^{-1}M$. The following relations with the *composite* iteration matrices hold:

$$E_{\overline{M}} = E_{M^T}E_M,$$
$$E_{\widetilde{M}} = E_M E_{M^T},$$
$$\mathcal{E}_{\overline{M}} = \mathcal{E}_{M^T}\mathcal{E}_M,$$

$$\mathcal{E}_{\widetilde{M}} = \mathcal{E}_M \mathcal{E}_{M^T}.$$

We summarize (see [31, 32]):

Proposition 2.5. *The following assertions are equivalent:*

(i) $\|E_M\|_A = \|\mathcal{E}_M\| < 1$;
(ii) $\|E_{M^T}\|_A = \|\mathcal{E}_{M^T}\| < 1$;
(iii) $\|E_{\overline{M}}\|_A = \|\mathcal{E}_{\overline{M}}\| < 1$;
(iv) $\|E_{\widetilde{M}}\|_A = \|\mathcal{E}_{\widetilde{M}}\| < 1$;
(v) $M + M^T - A$ *is s.p.d.;*
(vi) $\widetilde{M}$ *is s.p.d.;*
(vii) $\overline{M}$ *is s.p.d.*

We give the following definition of *A-convergence*:

Definition 2.6. An iteration process is *A-convergent* if the A-norm of the iteration matrix is less that one, or equivalently, if any of the conditions in Proposition 2.5 hold.

For any M, the A-convergence ($\|E_M\|_A < 1$) implies the convergence of the corresponding iteration.

Since $\mathbf{e}_{k+1} = E_M \mathbf{e}_k$ can be equivalently represented as $A^{\frac{1}{2}}\mathbf{e}_{k+1} = \mathcal{E}_M A^{\frac{1}{2}}\mathbf{e}_k$, it is clear that $\lim_{k\to\infty} E_M^k = O$ iff $\lim_{k\to\infty} \mathcal{E}_M^k = O$, i.e., $\rho(E_M) < 1$ iff $\rho(\mathcal{E}_M) < 1$. Actually, $E_M = A^{-\frac{1}{2}}\mathcal{E}_M A^{\frac{1}{2}}$, i.e., $\mathcal{E}_M$ and E_M are *similar*, and, thus, they have the same *spectrum*, hence $\rho(\mathcal{E}_M) = \rho(E_M)$.

We note $ME_MM^{-1} = \left(I - AM^{-1}\right) = (E_{M^T})^T$ and, by matrix similarity, $\rho(E_M) = \rho(E_{M^T})$, hence the iteration with M is convergent iff the iteration with M^T is convergent.

Consider now a symmetric M. Then, $\mathcal{E}_M$ is also symmetric. Hence, $\|E_M\|_A = \|\mathcal{E}_M\| = \rho(\mathcal{E}_M) = \rho(E_M)$. That is, for a symmetric preconditioner, M, the iteration is convergent iff it is A-convergent. Obviously, when M is symmetric the (A-)convergence implies that M is s.p.d. Actually, the corresponding iteration is (A-)convergent iff $\mathbf{x}^T M\mathbf{x} > \frac{1}{2}\mathbf{x}^T A\mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^n$.

Clearly, an iteration with M (possibly non-symmetric) is A-convergent iff the iteration with $\overline{M}$ (or, equivalently, with $\widetilde{M}$) is (A-)convergent (it is contained in Proposition 2.5).

Remark 2.7. Consider $n = 2$, $A = I$, and $M^{-1} = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$. Then, $E_M = \left[\begin{smallmatrix} 0 & -1 \\ 0 & 0 \end{smallmatrix}\right]$, $E_{M^T} = \left[\begin{smallmatrix} 0 & 0 \\ -1 & 0 \end{smallmatrix}\right]$, $E_{\overline{M}} = \left[\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right]$, and $E_{\widetilde{M}} = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right]$. It is straightforward to see that $\rho(E_M) = \rho(E_{M^T}) = 0$ and $\rho\left(E_{\overline{M}}\right) = \rho\left(E_{\widetilde{M}}\right) = 1$. That is, the iteration with M (and also with M^T) is convergent while the composite iterations are not (A-)convergent. Using the considerations above (or using $\|E_M\|_A = \|E_M\| = \sqrt{\rho\left(E_{\overline{M}}\right)} = 1$) we conclude that the iteration with M is not A-convergent although it is convergent. That is, in the non-symmetric case convergence and A-convergence are not generally equivalent.

Remark 2.8. As we saw above, using non-symmetric M we may end up with a convergent iteration that is not A-convergent. Consider a fixed $n < \infty$ (fixed h in the finite element case). Then, using the well-known fact that on a finite-dimensional real or complex linear space all norms are equivalent we can conclude that when an iteration is convergent it also converges in A-norm even if $\|E_M\|_A \geq 1$ (i.e., if it is not A-convergent in the sense of Definition 2.6).

We formulate the following well-known results for Gauss-Seidel method when A is s.p.d. split as $D + L + L^T$, where D is the diagonal of A and L is the strictly lower triangular part of A.

Corollary 2.9. *The* forward Gauss-Seidel *iteration (the one with $M = L+D$) is A-convergent. This also holds for* backward Gauss-Seidel *(the one with $M = D + L^T$) and the* symmetric Gauss-Seidel *(the one with $\overline{M}$ or $\widetilde{M}$ when M is the one for forward or backward Gauss-Seidel).*

2.2.3 Convergence Factor

We now describe a way to estimate the rate of convergence of a given iterative method that we use in our numerical experiments.

Let $\|\cdot\|$ be a given norm. The *convergence factor* of an iteration process with M is $\|E_M\|$. We estimate the convergence factor at the i-th step ($i > 0$) of the iteration by

$$\frac{\|\mathbf{e}_i\|}{\|\mathbf{e}_{i-1}\|}.$$

By definition, an *average convergence factor* at the i-th step ($i > 0$) of the iteration is given by the expression

$$\left(\frac{\|\mathbf{e}_i\|}{\|\mathbf{e}_0\|}\right)^{\frac{1}{i}}. \tag{2.2}$$

The *asymptotic convergence factor* is estimated by

$$\frac{\|\mathbf{e}_m\|}{\|\mathbf{e}_{m-1}\|}, \tag{2.3}$$

where m is the last step of the iteration, i.e., when the process stops due to satisfaction of some criterion. Commonly, $\rho(E_M)$ is referred to as (the exact) asymptotic convergence factor.

Another estimate of the (asymptotic) convergence factor we may use is based on the last $m_0 + 1$ ($1 \leq m_0 \leq m$) iterates. Namely,

$$\left(\frac{\|\mathbf{e}_m\|}{\|\mathbf{e}_{m-m_0}\|}\right)^{\frac{1}{m_0}}. \tag{2.4}$$

If $m_0 = 1$, then (2.4) turns into (2.3); if $m_0 = m$, then (2.4) turns into (2.2) for $i = m$. When $1 \leq m_0 \ll m$ (2.4) may be used as an estimate of the asymptotic convergence factor.

Sometimes we may use $\|\cdot\|^2$ instead of $\|\cdot\|$ and we typically use the A-norm.

2.2.4 Stalled Convergence

Very often in practice we encounter cases when the convergence stalls or is slow, i.e., when $\mathbf{e}_{k+1} \approx \mathbf{e}_k$. We are interested to obtain some information for the iteration process in such cases from the computed iterates.

Assume that after some number of iteration steps the convergence of the iteration $\mathbf{x}_{k+1} = E_M \mathbf{x}_k$ stalls, i.e., $\|\mathbf{x}_{k+1}\| \approx \|\mathbf{x}_k\|$. This is essentially a *power method* and $\mathbf{x}_k$ is an approximation of an eigenvector of E_M corresponding to the eigenvalue $\lambda_{\max}(E_M) \approx 1$, that is, an approximation of an eigenvector of $M^{-1}A$ corresponding to $\lambda_{\min}\left(M^{-1}A\right) \approx 0$.

Alternative, but still related way to look at this, is to say that $\mathbf{x}_k$ is rich in components of the error that the iteration process cannot *damp* (reduce) (or damps them quite slowly). In general, if we run the iteration with a random initial guess and the convergence after some number of steps is not satisfactory with respect to (w.r.t.) some criterion, then we consider $\mathbf{x}_k$ rich in components of the error that are not reduced with a satisfactory speed, i.e., the iteration process is not efficient enough on these error components.

2.2.5 Spectral Equivalence

We present here another notion in numerical linear algebra. It is relevant when optimal solvers are constructed.

Let M be a $n \times n$ s.p.d. preconditioner.

Definition 2.10. M is said to be *spectrally equivalent* to A if there exist constants $C > 0$ and $c > 0$ independent of n (independent of h, in the finite element case) s.t.

$$c\,\mathbf{v}^T M \mathbf{v} \leq \mathbf{v}^T A \mathbf{v} \leq C\,\mathbf{v}^T M \mathbf{v} \text{ for all } \mathbf{v} \in \mathbb{R}^n.$$

Clearly, this is an equivalence relation.

If an iteration process using spectrally equivalent preconditioner is convergent, then the convergence factor is uniformly bounded away from unity. Thus, a desired reduction of the error can be achieved in $O(1)$ iterations and the question regarding the optimality of the iteration process is reduced to the ability to implement the single iteration step in $O(n)$ operations. The multigrid methods that we consider later on have the potential to achieve this goal.

2.3 Singular Value Decomposition (SVD)

For the method we develop in this thesis we need to construct a linearly independent subset from a given set of vectors that spans the same space as the original set, i.e., we need to construct a computational basis for the space spanned by the given vectors. One way to achieve this is to use the popular singular value decomposition (or SVD) algorithm.

The following holds (see [18]):

Theorem 2.11. *For any matrix $C \in \mathbb{R}^{p\times q}$ there exist orthogonal matrices*

$$U = [\mathbf{u}_1, \ldots, \mathbf{u}_p] \in \mathbb{R}^{p\times p} \text{ and } V = [\mathbf{v}_1, \ldots, \mathbf{v}_q] \in \mathbb{R}^{q\times q}$$

such that we have the following singular value decomposition (SVD)*:*

$$C = U\Sigma V^T,$$

where $\Sigma \in \mathbb{R}^{p\times q}$, $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_k)$ ($k = \min\{p, q\}$) and

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$$

are uniquely determined singular values *of C.*

The vectors $\mathbf{u}_1, \ldots, \mathbf{u}_p$ and $\mathbf{v}_1, \ldots, \mathbf{v}_q$ are called respectively *left singular vectors* and *right singular vectors.*

Let r be s.t. $\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_k = 0$, i.e., the number of nonzero singular values. Then:

$$\begin{aligned} \operatorname{rank}(C) &= r, \\ \operatorname{Ker}(C) &= \operatorname{span}\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_q\}, \\ \operatorname{Range}(C) &= \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}. \end{aligned}$$

If we have a set of vectors $\mathbf{c}_1, \ldots, \mathbf{c}_q$, some of them might be linearly dependent, and we want to derive a linear independent set (basis) that spans the same space as the original set of vectors. To achieve this we use SVD. Namely, we form $C = [\mathbf{c}_1, \ldots, \mathbf{c}_q]$ and compute its SVD. Thus, we get the set of linearly independent (actually orthonormal) vectors $\mathbf{u}_1, \ldots, \mathbf{u}_r$ that span the *range (column space)* of C. Due to rounding error it is quite unlikely to get singular values that are precisely equal to zero. That is why to determine r we only take the singular values that are larger than a small tolerance ε times the largest singular value, i.e., $\varepsilon\sigma_1$, and the rest are considered *effectively* zero.

We compute SVD using the LAPACK routine `DGESVD`.

2.4 Gram-Schmidt Orthogonalization

When we have a set of orthogonal vectors and we need to add another vector we do it by orthogonalizing the new vector to the previous set. This can be achieved by the popular Gram-Schmidt algorithm.

Given a set of orthogonal, w.r.t. some inner product $\langle\cdot,\cdot\rangle$, vectors $\mathbf{q}_1, \ldots, \mathbf{q}_m$ and let $\mathbf{x}$ be another vector. Note that $\mathbf{x}$ can possibly be linearly dependent on the first vectors. We want to orthogonalize $\mathbf{x}$ to the first vectors, i.e., to find a vector $\mathbf{q}_{m+1}$ orthogonal to the others s.t. $\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_m, \mathbf{x}\} = \operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_m, \mathbf{q}_{m+1}\}$. We do it the simplest way, i.e., by subtracting from $\mathbf{x}$ the orthogonal projection of $\mathbf{x}$ on $\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ and the resulting difference is the desired vector. This is essentially *Gram-Schmidt orthogonalization.* That is, we look for $\mathbf{q}_{m+1} = \mathbf{x} + \sum_{i=1}^{m} \alpha_i \mathbf{q}_i$, where α_i are determined from

$\langle \mathbf{q}_{m+1}, \mathbf{q}_j \rangle = 0$ for all $j = 1, \ldots, m$. Thus,

$$\alpha_i = -\frac{\langle \mathbf{x}, \mathbf{q}_i \rangle}{\langle \mathbf{q}_i, \mathbf{q}_i \rangle}, i = 1, \ldots, m.$$

If $\mathbf{x}$ is linearly dependent on the first vectors, then $\mathbf{q}_{m+1} = \mathbf{0}$ and the initial system of vectors stays the same, otherwise $\mathbf{q}_{m+1}$ is added to the system. In the implementation: if $\langle \mathbf{q}_{m+1}, \mathbf{q}_{m+1} \rangle$ is smaller than a certain small threshold, $\mathbf{q}_{m+1}$ is considered effectively zero and $\mathbf{x}$ – linearly dependent on the others.

In the special case of orthonormal initial system $\alpha_i = -\langle \mathbf{x}, \mathbf{q}_i \rangle$ and to normalize the new vector $\mathbf{q}_{m+1}$ we divide it by $\sqrt{\langle \mathbf{q}_{m+1}, \mathbf{q}_{m+1} \rangle}$.

2.5 Meshes and Graphs

We finish this chapter by defining two graphs utilizing finite element meshes that are useful in our future constructions. We also refer to the graph partitioning problem. This topic is also relevant to the implementation of our solver.

2.5.1 Mesh Graphs

We are interested in the following two *mesh graphs*.

Definition 2.12. The *nodal graph* of a mesh is the (*undirected*) graph with *vertices* the nodes of the mesh. Two vertices of the graph are *adjacent* (connected by an edge) if they correspond to mesh nodes of a common element.

Definition 2.13. The *dual graph* of a mesh is the (undirected) graph with vertices the elements of the mesh. Two vertices of the graph (elements from the mesh) are adjacent if these corresponding elements share a side (or a face, in 3D).

2.5.2 Graph Partitioning

Here we define *graph partitioning* and refer to the METIS package which is, actually, what is relevant to the implementation of our solver.

Definition 2.14 (see [21])**.** The *graph partitioning* problem is finding k equal, in number of elements, pairwise disjoint subsets (*partitions*) of the set of vertices, of the graph, that *cover* the set of vertices, i.e., their union equals the set of vertices. The partitioning has an objective to minimize the *edgecut*, i.e., the number of edges that connect vertices from different subsets.

The graph partitioning problem is *NP-complete*. The package METIS implements *multilevel heuristics* to approximately and efficiently solve the problem.

An additional expectation we have is that the partitions (after removing the edges between different partitions) to be *connected* as much as possible. The version METIS 4.0.3 (that we currently use) tries to build connected partitions but sometimes may return results with some of the partitions not connected.

There are newer versions (5.0 and above) that can compute approximate solutions to the partitioning problem in which each partition is connected (see [3]).

This page is intentionally left blank.

3

Multigrid

This chapter presents basic facts and concepts of multigrid starting with the example of the smoothing property of classical iterative methods, touching upon the two-grid algorithm and the basic intuition behind multigrid, ending up with an introduction of algebraic multigrid in the setting suitable of presenting the method we study.

In Section 3.1 we give a classical example of the smoothing property of the weighted Jacobi method. Section 3.2 presents the two-grid algorithm which is the cornerstone of the solver we implement. Section 3.3 gives a short description of the basic multigrid idea. Section 3.4 is devoted to the algebraic multigrid including a motivation for the choices we make. In Section 3.5 we present the method we use.

3.1 Classical Iterative Methods as Smoothers

We start with two of the main concepts in multigrid – the concepts of *smooth* and *oscillatory* components of the error. We give an illustration of the smoothing property of the classical iterative methods based on the example of how the *weighted Jacobi* method acts on the components of the error.

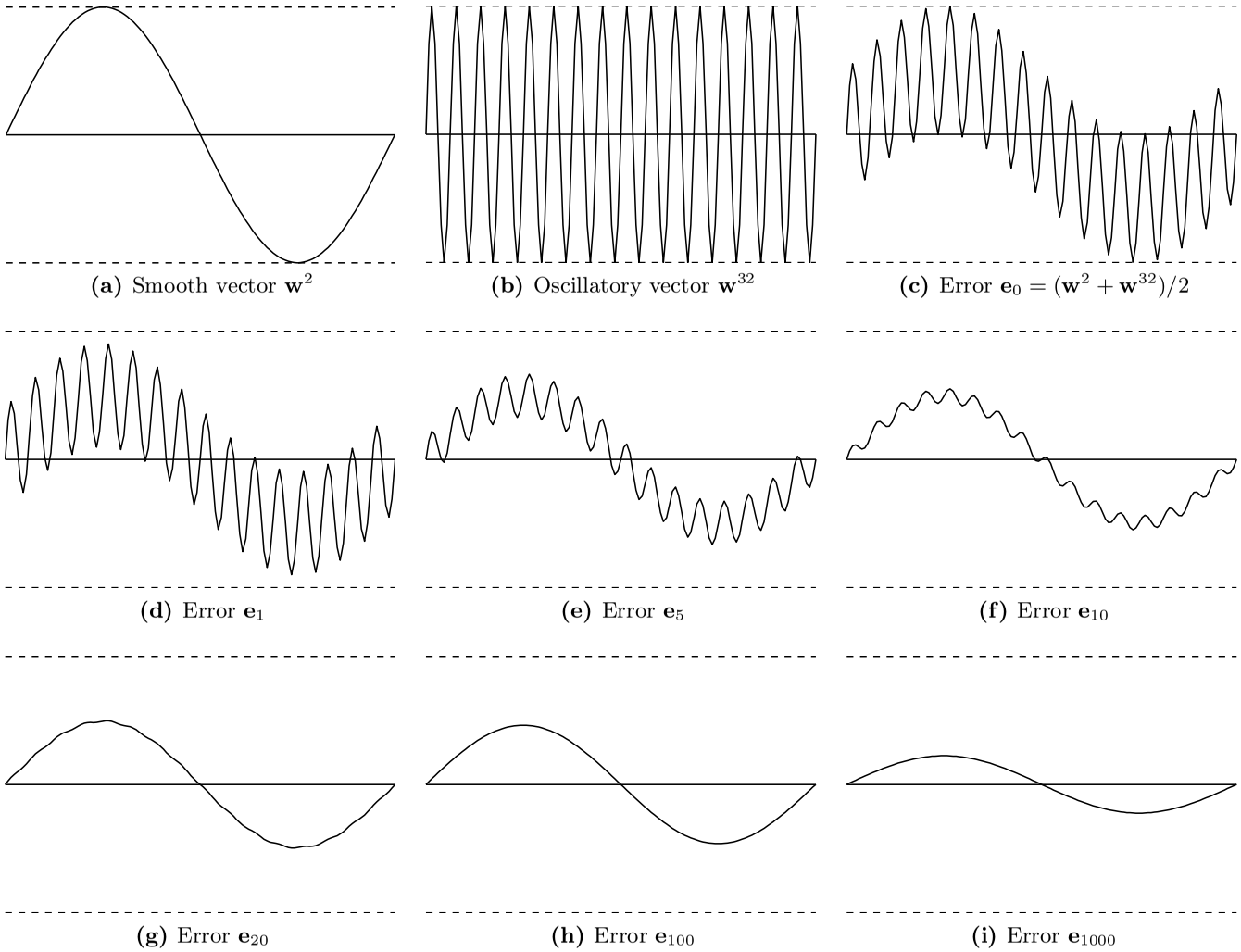
We consider the ordinary differential equation $-u'' = 0$ on $(0, 1)$ with zero boundary conditions and partition the interval in n equal segments. Using a finite difference discretization, we end up with a linear system having the $(n-1) \times (n-1)$ tridiagonal matrix $A = \text{tridiag}(-1, 2, -1)$. Also, consider the *weighted (damped) Jacobi* method with a *weighting factor* $\omega \in (0, 1]$ (an iteration process with $M = \frac{1}{\omega} D$, $D = 2\ I$ is the diagonal of A). The method is convergent with an iteration matrix $E_M = I - \frac{\omega}{2} A$. Thus, E_M and A have the same eigenvectors $\mathbf{w}^i$ which are (see [12])

$$w_j^i = \sin\left(\frac{ji\pi}{n}\right) \text{ for } i, j = 1, \ldots, n-1.$$

We also have the expressions for the respective eigenvalues

$$\lambda_i(A) = 4\sin^2\left(\frac{i\pi}{2n}\right),$$

$$\lambda_i(E_M) = 1 - 2\omega\sin^2\left(\frac{i\pi}{2n}\right).$$



(a) Smooth vector $\mathbf{w}^2$ **(b)** Oscillatory vector $\mathbf{w}^{32}$ **(c)** Error $\mathbf{e}_0 = (\mathbf{w}^2 + \mathbf{w}^{32})/2$

(d) Error $\mathbf{e}_1$ **(e)** Error $\mathbf{e}_5$ **(f)** Error $\mathbf{e}_{10}$

(g) Error $\mathbf{e}_{20}$ **(h)** Error $\mathbf{e}_{100}$ **(i)** Error $\mathbf{e}_{1000}$

Figure 3.1: The smoothing effect of the weighted Jacobi iteration with $\omega = \frac{2}{3}$ and $n = 128$.

To illustrate the smoothing property in question, the eigenvectors *(eigenmodes or Fourier modes)* of A are divided into *smooth (low-frequency)*, for $i < \frac{n}{2}$, and *oscillatory (high-frequency)*, for $i \geq \frac{n}{2}$.

If we consider the error represented like

$$\mathbf{e}_k = \sum_{i=1}^{n-1} a_i \mathbf{w}^i,$$

we readily get

$$\mathbf{e}_{k+1} = E_M \mathbf{e}_k = \sum_{i=1}^{n-1} a_i \lambda_i \left(E_M\right) \mathbf{w}^i.$$

That is, the i-th mode of the error is damped by a factor of $\lambda_i \left(E_M\right)$. So the reduction of the oscillatory modes (related to the upper eigenvalues of A) can be balanced by properly choosing ω (e.g., $\omega = \frac{2}{3}$) whereas the reduction factor of the smooth modes (related to the low eigenvalues of A) can be arbitrary close to unity as n becomes large. This phenomenon is referred to as the smoothing property of the damped Jacobi method (see fig. 3.1) – the oscillatory modes are reduced with a much larger factor than the smooth ones. This property is common for virtually all classical iteration methods (such as Gauss-Seidel, block-versions, overlapping Schwarz methods, incomplete LU factorization methods, etc.) and it actually reflects their slow convergence for discretized second order elliptic PDEs (not only for the above simple 1D example). It is related to the fact that the modes of A span a very wide range of frequencies – the eigenvalues of A can be arbitrary small in magnitude as $n \to \infty$. This, in turn, is related to the fact that A is ill-conditioned.

3.2 The Two-Grid Method

In this section we introduce the main algorithm we consider – the *two-level* (or *two-grid*) iteration process.

Let A be the $n \times n$ s.p.d. matrix of the system we want to solve and let P be a *full-rank* $n \times n_c$, $n_c < n$, *interpolation (prolongation) matrix (operator)*. Let R be a *restriction matrix (operator)*. In our setting $R = P^T$. R and P are called *(intergrid) transfer operators*. We define a *coarse matrix (operator)* via the *variational relation*

$$A_c = P^T A P \text{ (Galerkin property (relation))}.$$

Clearly, A_c is symmetric and the full-rank property, of P, implies that A_c is s.p.d. We also require P to be sparse resulting in sparse A_c.

We consider an iteration method with a preconditioner M. In *multigrid* context the role of M is the same as the damped Jacobi used earlier, namely to act as *smoother*, that is the role of M is to remove the oscillatory components of the error. The classical iteration methods are referred also as *relaxation* methods when used in a multigrid context, since they exploit local updates and require $O(n)$ operations per iteration for a given sparse matrix A.

Given a current iterate $\mathbf{x}$, we present next the *(symmetrized) two-grid (TG), or two-level (TL), iteration method* that computes the two-grid iterate $\mathbf{x}_{TG}$ for solving the given equation $A\mathbf{x} = \mathbf{b}$:

Algorithm 3.1 (TG algorithm)**.** *Having the current iterate* $\mathbf{x}$ *we compute the next TG iterate* $\mathbf{x}_{TG}$ *by performing the following steps:*

(i) "pre-smoothing"*:*

- *compute the intermediate iterate*

$$\mathbf{y} = \mathbf{x} + M^{-1}(\mathbf{b} - A\mathbf{x});$$

(ii) "coarse-grid correction"*:*

- *restrict the residual*

$$\mathbf{r}_c = P^T(\mathbf{b} - A\mathbf{y});$$

- *solve the coarse-grid defect equation*

$$A_c\mathbf{x}_c = \mathbf{r}_c;$$

- *interpolate and compute next intermediate iterate*

$$\mathbf{z} = \mathbf{y} + P\mathbf{x}_c;$$

(iii) "post-smoothing"*:*

- *compute the next two-grid iterate*

$$\mathbf{x}_{TG} = \mathbf{z} + M^{-T}(\mathbf{b} - A\mathbf{z}).$$

The algorithm can be recursively extended to a multigrid version. For example, a straightforward extension by recursively calling TG on coarser levels will result in the popular $V(1,1)$*-cycle*. Our focus in what follows is on the TG algorithm.

Define the operator $\pi_A = PA_c^{-1}P^TA$. It is clearly a *projection operator*, i.e., $\pi_A^2 = \pi_A$. If we consider the TG algorithm as a stationary iteration, $\mathbf{x} \mapsto \mathbf{x}_{TG}$, it can be rewritten in terms of the (two-grid) preconditioner (see [31, 32])

$$B_{TG}^{-1} = \overline{M}^{-1} + E_{M^T}PA_c^{-1}P^T\left(E_{M^T}\right)^T.$$

The (two-grid) iteration matrix corresponding to Algorithm 3.1 is related to B_{TG} as follows

$$E_{TG} = I - B_{TG}^{-1}A = E_{M^T}(I - \pi_A)E_M. \tag{3.1}$$

This is the case, under the assumption that *exact solve* is used on the coarse level. Clearly, B_{TG} is symmetric.

We assume that the iteration with M is A-convergent. This, clearly, implies that B_{TG}^{-1} (and B_{TG}) is s.p.d. It also implies (see [31, 32]) $\mathbf{v}^TA\mathbf{v} \leq \mathbf{v}^TB_{TG}\mathbf{v}$, for all $\mathbf{v}$, which, in turn, implies the A-convergence of the two-grid method.

The projection $I - \pi_A$ in (3.1) corresponds to the coarse-grid correction. It might be considered as a stationary iteration where A is preconditioned

by $C = P\left(P^T A P\right)^{-1} P^T$, i.e., C is kind of a (coarse) approximation of A^{-1}. Obviously, $\mathrm{Ker}(\pi_A) = \mathrm{Ker}(P^T A) \neq \{\mathbf{0}\}$ and in this kernel the coarse grid correction is not efficient and, as a matter of fact, it turns out the coarse-grid correction alone is not convergent.

Easily, the space where π_A acts as identity is equal to $\mathrm{Range}(P)$, hence $\mathrm{Ker}(I - \pi_A) = \mathrm{Range}(P)$. Thus, the coarse-grid correction is quite efficient in $\mathrm{Range}(P)$, i.e., the coarse-grid correction efficiently reduces the components of the error in $\mathrm{Range}(P)$. The intuition behind this fact is as follows. Solving the coarse-grid defect equation (see Algorithm 3.1) we get the coarse approximation $\mathbf{x}_c$ of the error. Then, the current intermediate iterate is corrected by $P\mathbf{x}_c$. Clearly, only the components of the error in $\mathrm{Range}(P)$ will be reduced. The remaining components of the error must be reduced by the smoother to have an overall efficient two-grid method. This concept is motivated in more details in the following section.

3.3 The Basic Multigrid Idea

We now summarize briefly the main intuition behind multigrid.

As already touched upon multigrid may be considered as an interplay between smoothing and coarse-grid correction in a *divide and conquer* manner – the smoother damps the oscillatory error components and the coarse-grid correction damps what the smoother is not able to reduce, i.e., the smooth error components, and vice versa. An important requirement is that the coarse-grid correction must not excite (too much) the oscillatory modes, i.e., to undo the work done by the smoother, and the smoother, in turn, not to undo the work done by the coarse-grid correction by exciting too much smooth components of the error. In the s.p.d. setting if we use A-convergent smoothers and coarse matrices obtained variationally from the fine-grid ones, the latter requirement is automatically satisfied.

In *geometric multigrid* (when actual "physical" grids are present) "smooth" and "oscillatory" have geometric meaning, i.e., vectors look smooth or oscillatory when plotted. Important property is that smooth vectors on a fine grid may look oscillatory on a coarse grid, i.e., when the resolution is decreased, thus coarse-level smoothers will be more efficient to damp modes that were smooth on the finer grid. This motivates the multigrid idea of applying the smoothers on a sequence of coarse grids. The fact that we may consider these smooth vectors on a coarse grid comes from the quite intuitive fact that smooth functions can be well represented (approximated) on a coarse grid (see [27, 11, 28, 12]). Tightly related to the last is the fact that coarse vectors interpolated on a fine grid look smooth, i.e., the vectors in $\mathrm{Range}(P)$ look smooth. Actually, we would expect smooth errors to be in $\mathrm{Range}(P)$, since, as we saw above, the coarse-grid correction is efficient on $\mathrm{Range}(P)$.

Everything above outlines the fast convergence of multigrid methods, i.e., a good reduction of the error is achieved in a small number of cycles (iterations).

For the method to be efficient it is necessary that each cycle have low complexity. The last is a consequence of the fact that all operations are considerably faster on coarser levels and all matrices and vectors take up considerably less memory on coarser levels. Thus, the divide and conquer approach (in the *frequency domain*) results not only in fast convergence but also in low complexity of the cycles and hence in a fast overall solver.

3.4 Algebraic Multigrid (AMG)

This section is devoted to a brief introduction to algebraic multigrid which is our choice for solving the problem of our main interest.

3.4.1 Introduction

We give here some basic thoughts about *algebraic multigrid (AMG)*.

Geometric multigrid is an efficient method for building fast solvers for systems arising from discretizations of elliptic boundary value problems. It is applied to solve a discretized PDE when the geometry of the problem is known – discretizations on increasingly finer grids are used with transfer operators between the grids based on the geometry of the grids. That is, the *coarsening process*, i.e., the process of selecting the coarse levels and defining the interpolation is fixed and, since it is usually kept as simple as possible, fine grids are usually uniform refinements of coarser grids and linear interpolation is used. In a finite element setting, the coarsening and interpolation are naturally predefined; if we use nested finite element spaces corresponding to a mesh refinement procedure the coarse matrices are automatically obtained from the fine-grid matrices variationally via the natural embedding of the coarse spaces into the fine-grid ones; if linear elements are used, the interpolation is naturally linear. The burden is on the proper choice of smoothers that reduce the components of the error not efficiently reduced by the coarse-grid correction, i.e., not in the range of interpolation (see [22, 26, 28, 32]).

For discretized PDEs the user may not have access to the hierarchy of meshes and discretization operators on coarse levels, or these may not actually exist if a single unstructured fine mesh is used. Also, even in the case when such hierarchy is available, the resulting MG method may not be as efficient (unless expensive smoothers are used). In all such cases an option to use the multigrid concept in order to achieve reasonable convergence is provided by the *algebraic* multigrid originally proposed by Brandt, McCormick and Ruge in 1981. Although grids and points might not be present, when algebraic multigrid is involved, the terms are still used although not related to "physical" grids. Actually, even the word multigrid is not very precise, unlike *multilevel*, it is still customarily used. In AMG the smoother is typically fixed and usually a simple smoother is chosen. The burden is on choosing a coarse space and interpolation s.t. the coarse-grid correction reduces the components of the error not efficiently damped by the smoother, i.e., the burden is on building P s.t.

Range(P) contains the components inefficiently reduced by the smoother. That is, the construction of P is kind of an *inverse problem*. The coarsening process is usually *automatic* and *operator-dependent* and uses information provided by the given matrix. Thus, the AMG concept allows developing methods that can be efficient for a wide range of problems, *robust*, and even resulting in so-called *black-box* solvers, i.e., ones that use no additional information about the origin of the system or the structure of the problem at hand and, in principle, these solvers can be developed separately and independently of the engines generating the systems to be solved without using any connection between them.

A main objective when building an AMG solver is to keep the number of operations per iteration minimal, the best being $O(n)$, and to achieve rate of convergence similar to geometric multigrid, i.e., optimal or almost optimal. This is a formidable task in general, which, however, has become feasible in a number of applications if the problem combined with a suitable solver (used as a smoother) allows for coarsening, i.e., identifying the slow convergence error components.

3.4.2 Motivation

We now describe our motivation for choosing AMG instead of geometric multigrid.

As we already mentioned, even for systems arising from discretizations of differential equations AMG may be preferred before geometric multigrid (see [22, 26, 28, 29]), since the burden on the smoother might be too much resulting in a too complicated smoother or a suitable smoother may not be known, especially in 3D. Or, switching the burden to choosing coarse levels and interpolation operators for a fixed smoother may result in a better multilevel method than the geometric one. Better means not only in convergence properties but also from an implementation point of view.

If the fine mesh is too irregular it may be very hard or impossible to end up with enough levels of mesh coarsening, especially in 3D, needed for the geometric multigrid. In practice, we need solvers that work with irregular meshes both in 2D and in 3D, this is much easier to achieve with AMG. In some cases it may be very hard, too expensive, or impossible to find a smoother that smooths the error enough, e.g., when discontinuous PDE coefficients are involved (which is our case), especially when the jumps in the coefficients are very large (which is also our case), then, an alternative is to compensate with proper (more expensive) coarse spaces, which can be achieved by AMG, which is our choice.

Considering everything above it is self-explanatory, at this point, why we choose AMG and not geometric multigrid although we solve systems arising from finite element discretization of elliptic boundary value problems. However, it might be tempting to use geometric multigrid considering we have to

solve the problem many times with little changes in the matrices. Geometric multigrid allows us to build once the set of grids and transfer operators and reuse them for all systems needed to be solved while with algebraic multigrid they are operator-dependent and may need to be changed (e.g., rebuilt) for each realization of the coefficient. Geometric multigrid is not a better choice for us because of the reasons described above, however, at the same time rebuilding the hierarchy for each realization of the coefficient may render algebraic multigrid inefficient. To resolve the latter issue we use *adaptation*[1] which allows us to adapt (and thus reuse) an already built hierarchy for a matrix for one realization of the coefficient to matrices for consequent realizations instead of rebuilding the hierarchy from scratch for each realization.

3.5 The Method We Use

Finally, we may present the method we use.

We should note that the method utilizes the fact that the systems we solve arise from finite element discretizations of partial differential equations. The way it uses the mesh is different compared to geometric multigrid and it does not break the robustness, i.e., it is directly applicable to irregular meshes and to problems with discontinuous coefficients with large jumps. It is also directly applicable to 3D problems.

In the sections that follow we first touch upon the smoothed aggregation AMG and the element-based AMG, which are key ingredients of our AMG method, and then we get into the main adaptive component of our method.

3.5.1 Smoothed Aggregation AMG

The AMG based on *aggregation* builds the coarse space by partitioning the set of variables into mutually disjoint subsets (*aggregates*) (see [29, 26, 28, 31, 32]). Each aggregate on a fine level brings down a degree of freedom (or several ones per aggregates, as it will be the case in our method later on) on the next coarser level. Of course, there is more than one way to choose the aggregates and the interpolation operators based on these aggregates. The aim is to achieve good convergence rate for minimal complexity of the V-cycle.

In the original paper [29] a few guiding heuristic principles are presented and an algorithm is proposed for building the aggregates and respective interpolation operators in the case of linear systems arising from finite element discretizations of second-order elliptic equations. Similarly to the classical AMG the coarsening exploits a notion of *strong coupling*. One of the guiding principles is that every coarse space should represent the constant exactly, since the constants are the *zero energy* modes, i.e., they have zero A-norm aside from essential boundary conditions.

The *tentative prolongation operator* is built as piecewise constant interpo-

[1] The words "adaptation", "adaptive", "adapt" here have nothing to do with adaptive mesh refinement. They are used in relation with the solver and the hierarchy for the solver which is the one that is being adapted aiming to be efficient for a modified operator.

lation, i.e., the value of a coarse-grid dof (which corresponds to an aggregate on the fine level) is spawned as a value of each fine-grid dof in the corresponding aggregate. In smoothed aggregation AMG the tentative interpolation operator is *smoothed* to produce the final prolongation operator. The smoothing is done by multiplying the tentative interpolation matrix by an iteration matrix or, more generally, by a matrix polynomial. An appropriate smoothing results in *energy boundedness* of the coarse basis.

In [29] it is assumed that the resulting coarse spaces can be viewed as being associated with a partition of the domain behaving similarly to finite elements and it is also claimed (which depends on the construction of the aggregates) that the method behaves well when strong anisotropy and discontinuous coefficients are present, despite the more restrictive assumptions when the convergence estimate is derived.

Quite generally it is proven (see [30, 31, 32]), that the $V(1,1)$-cycle has almost optimal convergence, i.e., the $V(1,1)$-cycle MG preconditioner B is almost spectrally equivalent to A. Namely, the upper bound of the constant K in the spectral equivalence estimate $\mathbf{v}^T A\mathbf{v} \leq \mathbf{v}^T B\mathbf{v} \leq K\mathbf{v}^T A\mathbf{v}$, for all $\mathbf{v}$, depends polynomially on the number of levels in the hierarchy.

3.5.2 Element-based AMG

The *element-based AMG (AMGe)* suggests a conceptually different way to look at the notion of "smooth error". Its aim is to avoid any premise about the nature of smooth errors. Instead, it assumes that smooth errors are characterized by the spectrum of the operator. Thus, a different guiding heuristic is suggested (see [7]) requiring the interpolation to approximate well eigenvectors corresponding to small eigenvalues of the matrix. In [7] two local measures are presented which require access to the local stiffness matrices (hence the "element-based" part in the name of the method). The purpose of these measures is to formulate a procedure of building the interpolation aiming to fulfill the heuristic principle above. Thus, element-based interpolation is achieved. The method requires a partitioning of the dofs into coarse and fine dofs to be available. It presents a way to build the interpolation based on the already chosen coarse dofs.

The *spectral AMGe (ρAMGe)* (see [13, 14, 31, 32]) suggests a method to build the entire hierarchy, not just the interpolation, without the necessity of an a priori available partitioning into coarse and fine dofs. The construction of intergrid transfer operators is based on the assumption that smooth errors may be described by means of the eigenvectors of the matrix corresponding to small eigenvalues. Since obtaining the eigenvectors of the global stiffness matrix is impractical, it is assumed that smooth errors are characterized by the low eigenvectors of the *local stiffness matrices* corresponding the so called *agglomerates* which are sets of fine-grid elements. Once obtained, the low eigenvectors of the local stiffness matrices are patched together to form the

global interpolation operator. Although a kind of coarse mesh of agglomerates is involved the coarsening is completely different compared to geometric multigrid and the coarse space is built algebraically. This method, however, uses additional information (not only the given (global) matrix), namely, the local element matrices and relies on algorithms to generate agglomerated elements.

3.5.3 The Used Method

Here we describe the method we use. Our considerations are based on [10] which describes the two-grid method we implement. This method is described first. We extend this two-grid method by using adaptation which is described at the end.

The method is a combination of both approaches above and the aggregates are in a way derived from the agglomerates. It is efficient for solving PDEs with coefficients like ours – discontinuous with large jumps. It is also directly applicable to irregular meshes and 3D problems. The discontinuities of the coefficients must be resolved by the discretization only on the fine grid which holds in our case, since we consider a PDE coefficient that is constant on each element.

Since our purpose is implementing a method for efficiently solving not just a single PDE but the stochastic problem, i.e., a sequence of PDEs, we supplement the method in [10] with adaptation allowing us to reuse much of the already built hierarchies for solving consequent realizations of the PDE.

In the sections that follow we describe the method for building the interpolation operator and the smoother that we use for the two-grid method. Then, we present the adaptation procedure we use.

Unless explicitly stated otherwise, the constants are considered positive in the following sections.

3.5.3.1 Smoothed Aggregation Spectral Element Agglomeration Algebraic Multigrid

This section is devoted to the method for producing the interpolation operator. The smoother we use in the two-grid algorithm is also described. This section is entirely based on [10].

We consider a tessellation $\mathcal{T}_h$ and let the respective set of nodes be $\mathcal{N}_h$. Let $k = k(\mathbf{x})$ be a given positive function that is piecewise constant w.r.t. the elements of $\mathcal{T}_h$. Let G be the *(k-weighted) mass matrix*, i.e., $g_{ij} = \int_\Omega k\phi_i\phi_j \,\mathrm{d}\mathbf{x}$, and let D_G be its diagonal. Clearly, G is s.p.d.

We assume we have constructed a set $\mathcal{T}_H$ of n_a non-overlapping *agglomerates (agglomerated elements (AEs))* $\{T\}$. An agglomerated element $T \in \mathcal{T}_H$ is, by definition, a connected set (in the sense of the dual graph of the fine mesh) of fine-grid elements. We assume that $\{T\}$ covers the set of fine-grid elements $\{\tau\}$. That is, $\mathcal{T}_H$ forms a partition of $\mathcal{T}_h$. Let H be the characteristic mesh size of $\mathcal{T}_H$. It is important to stress that we do not assume H to be comparable in

any way to h. For each agglomerated element T, we can assemble the $n_T \times n_T$ local stiffness matrix A_T. We denote its diagonal by D_T. We also assume we are given a set of n_a aggregates $\{\mathcal{A}_T\}$ where each $\mathcal{A}_T$ is entirely contained in a unique agglomerated element T, denoted by $\mathcal{A}_T \subset T$. That is, each $\mathcal{A}_T$ contains only nodes of elements in T and $\{\mathcal{A}_T\}$ forms a partition of $\mathcal{N}_h$. Of course, there are interface nodes, i.e., shared by multiple agglomerates. Each interface node is assigned to a single aggregate among the aggregates within the agglomerated elements sharing that node.

We solve for each $T \in \mathcal{T}_H$ the following generalized eigenvalue problem

$$A_T \mathbf{q}_r = \lambda_r D_T \mathbf{q}_r, \; r = 1, \ldots, n_T, \tag{3.2}$$

assuming the eigenvectors are ordered in ascending order w.r.t. the magnitude of their corresponding eigenvalues. We take the first $m_T \leq n_T$ eigenvectors and restrict them to the aggregate $\mathcal{A}_T \subset T$ by extracting only the entries of the vectors that correspond to the nodes in $\mathcal{A}_T$ and thus obtaining the vectors $\widetilde{\mathbf{q}}_1, \ldots, \widetilde{\mathbf{q}}_{m_T}$. We form $Q_T = [\widetilde{\mathbf{q}}_1, \ldots, \widetilde{\mathbf{q}}_{m_T}]$ and by applying SVD (see Section 2.3) we get the set of orthonormal vectors $\mathbf{p}_1, \ldots, \mathbf{p}_{m'_T}$ ($m'_T \leq m_T$) that span the same space as the columns of Q_T. Thus, we have as a result the *local tentative interpolation operator* $\overline{P}_{\mathcal{A}} = \left[\mathbf{p}_1, \ldots, \mathbf{p}_{m'_T}\right]$.

By applying the above procedure for all agglomerates $T \in \mathcal{T}_H$ and producing all local tentative prolongation operators, i.e., $\overline{P}_{\mathcal{A}_1}, \ldots, \overline{P}_{\mathcal{A}_{n_a}}$, we get as a result the following $n \times n_c$ *(global) tentative interpolation operator*

$$\overline{P} = \begin{bmatrix} \overline{P}_{\mathcal{A}_1} & 0 & & 0 \\ 0 & \overline{P}_{\mathcal{A}_2} & & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \ldots & \overline{P}_{\mathcal{A}_{n_a}} \end{bmatrix}.$$

The final $n \times n_c$ interpolation operator is obtained by smoothing the tentative one. That is, $P = S\overline{P}$, where S (typically in SA AMG) is an appropriate $n \times n$ matrix polynomial. Appropriate choice of the smoothing results in a *stable in energy interpolation*. We return to the choice of smoother later on.

For any sparse s.p.d. matrix A and its diagonal D_A we have (see [32]) $\mathbf{v}^T A \mathbf{v} \leq \kappa \, \mathbf{v}^T D_A \mathbf{v}$, for all $\mathbf{v}$, where κ is an upper bound of the number of nonzero elements per row of A, i.e., it is uniformly bounded with respect to h and the coefficient k. Hence $\|D_A^{-\frac{1}{2}} A D_A^{-\frac{1}{2}}\| \leq \kappa$, where $\|\cdot\|$ is the Euclidean norm. Similar uniform estimates are valid for any D spectrally equivalent to D_A. Given some s.p.d. matrix D spectrally equivalent to D_A we let b be s.t. $\|D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\| \leq b \in O(1)$.

Assuming $\mathcal{T}_h$ is quasi-uniform and using the fact that k is constant on the elements of the tessellation $\mathcal{T}_h$ the following uniform, w.r.t. h and k, equivalence relations hold (see [10])

$$c \, \mathbf{v}^T D_G \mathbf{v} \leq h^2 \, \mathbf{v}^T D_A \mathbf{v} \leq C \, \mathbf{v}^T D_G \mathbf{v}, \text{ for all } \mathbf{v}. \tag{3.3}$$

We also assume that the preconditioner M (see Algorithm 3.1) possesses the following "smoothing" property uniformly w.r.t. h and H (see [10])

$$\mathbf{v}^T\overline{M}\mathbf{v} \leq \beta\left[\mathbf{v}^T A\mathbf{v} + \frac{b}{(H/h)^2}\|\mathbf{v}\|_D^2\right], \text{ for all } \mathbf{v}, \tag{3.4}$$

and let the following uniform, w.r.t. h and H, coercivity estimate hold (see [10])

$$\mathbf{v}^T\left(M + M^T - A\right)\mathbf{v} \geq \alpha\,\mathbf{v}^T A\mathbf{v}. \tag{3.5}$$

Now, take $S = s_\nu\left(b^{-1}D^{-1}A\right)$, where $s_\nu(t)$ is a polynomial of degree ν in $[0,1]$ s.t. $s_\nu(0) = 1$. We use (see [10, 9, 32, 31])

$$s_\nu(t) = (-1)^\nu \frac{1}{2\nu+1}\frac{T_{2\nu+1}\left(\sqrt{t}\right)}{\sqrt{t}}.$$

Here $T_l(t)$ denotes the *Chebyshev polynomial of the first kind* of degree l in $[-1,1]$.

Taking m_T large enough for each $T \in \mathcal{T}_H$, under the assumption of quasi-uniform $\mathcal{T}_h$, and taking ν to be of order $\frac{H}{h}$ the following *"weak approximation property"* holds uniformly, w.r.t. h, H, and k, for all $\mathbf{v} \in \mathbb{R}^n$ (see [10])

$$H^{-1}\|\mathbf{v} - P\left(\overline{P}^T D\overline{P}\right)^{-1}\overline{P}^T D\mathbf{v}\|_G \leq \alpha_1\|\mathbf{v}\|_A. \tag{3.6}$$

Under the same assumption as above and as a consequence of the *energy stability property* of the smoothed interpolation we have the following uniform estimate, w.r.t. h, H, and k, for all $\mathbf{v} \in \mathbb{R}^n$ (see [10])

$$\|\mathbf{v} - P\left(\overline{P}^T D\overline{P}\right)^{-1}\overline{P}^T D\mathbf{v}\|_A \leq \alpha_2\|\mathbf{v}\|_A. \tag{3.7}$$

Remark 3.2. Under the same assumptions as above, except the one for ν, we have that for any $\mathbf{v} \in \mathbb{R}^n$ there exists $\mathbf{v}_c \in \mathbb{R}^{n_c}$ s.t. the following uniform, w.r.t. h, H, and k, "weak approximation property" holds (see [10])

$$H^{-1}\|\mathbf{v} - \overline{P}\mathbf{v}_c\|_G \leq \alpha_0\|\mathbf{v}\|_A.$$

That is, we have the same "weak approximation property" without the smoothing of the prolongation matrix. The smoothing keeps the "weak approximation property" and additionally results in energy stability and, as a corollary, the estimate in energy norm (3.7).

As a consequence of (3.6) and (3.7) we have that for any $\mathbf{v} \in \mathbb{R}^n$ there exists a $\mathbf{v}_c \in \mathbb{R}^{n_c}$ s.t.

$$H^{-2}\|\mathbf{v} - P\mathbf{v}_c\|_G^2 + \|\mathbf{v} - P\mathbf{v}_c\|_A^2 \leq C_a\|\mathbf{v}\|_A^2 \tag{3.8}$$

holds uniformly w.r.t. h, H, and k.

In [10] it is proven that (3.3), (3.4), (3.5), and (3.8) are sufficient for the upper bound K_{TG} in the spectral equivalence estimate $\mathbf{v}^T B_{TG}\mathbf{v} \leq K_{TG}\,\mathbf{v}^T A\mathbf{v}$, for all $\mathbf{v} \in \mathbb{R}^n$, to be bounded from above uniformly w.r.t. h, H, and k. Clearly, $K_{TG} = \frac{1}{1-\rho_{TG}}$, where ρ_{TG} is the asymptotic convergence factor (spectral radius) of the TG iteration.

Remark 3.3. We are not going to use the matrix polynomial above to smooth

the prolongation operator but we are going to use a much simpler smoother. More details are found in Chapter 4.

The choice of M, D and the upper bound b

We now turn to the smoother M and the related D and b. The choice of the smoother is quite relevant, apparently, to the optimal convergence estimate above.

We choose D to be a particular *weighted ℓ_1-smoother*. Namely, $D = \operatorname{diag}(d_i)_{i=1}^n$, where $d_i = \sum_{j=1}^{n} |a_{ij}| \sqrt{\frac{a_{ii}}{a_{jj}}}$. Obviously D is s.p.d., even when A is positive semi-definite which corresponds to the case when no essential boundary conditions are posed (the same holds for D_A). We have (see [10]) the desired spectral equivalence $\mathbf{v}^T D_A \mathbf{v} \leq \mathbf{v}^T D \mathbf{v} \leq \kappa\, \mathbf{v}^T D_A \mathbf{v}$, for all $\mathbf{v}$. We also have (see [10]) $\mathbf{v}^T A \mathbf{v} \leq \mathbf{v}^T D \mathbf{v}$, for all $\mathbf{v}$, which implies that D is A-convergent. It also allows us to take $b = 1$ (instead of $b = \kappa$ when D_A is used). Another corollary is that for the generalized eigenvalue problem

$$A\mathbf{q}_r = \lambda_r D \mathbf{q}_r,\ r = 1, \ldots, n,$$

the largest eigenvalue is at most 1.

The smoother M that we use is a *polynomial smoother*. We consider the following polynomial (see [9, 10]) of degree $3\nu + 1$ in $[0, 1]$

$$p_\nu(t) = \left(1 - T_{2\nu+1}^2\left(\sqrt{t}\right)\right) s_\nu(t).$$

We have $p_\nu(0) = 1$, hence $p_\nu(t) = 1 - t q_{3\nu}(t)$, where $q_{3\nu}(t)$ is an appropriate polynomial of degree 3ν in $[0, 1]$.

Consider the smoother iteration matrix

$$I - M^{-1}A = p_\nu\left(b^{-1}D^{-1}A\right).$$

That is,

$$M^{-1} = \left[I - p_\nu\left(b^{-1}D^{-1}A\right)\right] A^{-1}.$$

Thus,

$$M^{-1} = b^{-1}D^{-1}A\, q_{3\nu}\left(b^{-1}D^{-1}A\right) A^{-1} = b^{-1}\, q_{3\nu}\left(b^{-1}D^{-1}A\right) D^{-1}, \qquad (3.9)$$

where we use twice the fact that for any polynomial $p(t)$, any square matrix C, and any invertible matrix T: $T^{-1}p(C)\,T = p\left(T^{-1}CT\right)$. Using, also, the fact that for any polynomial $p(t)$ and any square matrix C: $[p(C)]^T = p\left(C^T\right)$ we easily get from (3.9) that M is symmetric. (3.9) also shows that no inverses of A are involved when applying M^{-1}.

The smoother defined above has the coercivity property (3.5) (see [9, 10]). The following smoothing property also holds (see [9, 10])

$$\mathbf{v}^T \overline{M} \mathbf{v} \leq \beta \left[\mathbf{v}^T A \mathbf{v} + \frac{b}{(2\nu+1)^2} \|\mathbf{v}\|_D^2\right], \text{ for all } \mathbf{v}.$$

Thus, taking $2\nu + 1$ to be of order $\frac{H}{h}$ provides the desired property (3.4).

As we mentioned above, our choice of D implies $b = 1$. We note that

the smoother described above is A-convergent. It directly follows from the coercivity property (3.5).

Remark 3.4. As we described above, the TG algorithm is convergent with a factor independent of both H and h, thus, independent of how large the coarsening is, i.e., independent of the magnitude of $\frac{H}{h}$. Clearly, the magnitude of the coarsening is compensated by the large polynomial degree of both the interpolation smoother and the smoother in Algorithm 3.1.

3.5.3.2 Solver Adaptation

Here we give a short description of the adaptation procedure we have studied. We first present the main idea. Implementation details will be described in the following Chapter 4.

We recall the notion of stalled convergence introduced in Section 2.2.4. The basic idea reads: we apply a given iteration method to the system $A\mathbf{x} = \mathbf{0}$ and monitor its convergence starting with a random initial iterate $\mathbf{x}_0$. If the convergence is not satisfactory, we take the last iterate $\mathbf{x}_m$, referred to as *prototype* (see [17]), and call it *"algebraically smooth"*. If the method is multigrid V-cycle possible reasons for the stalled convergence (see [31, 32]) is either that the coarse space cannot approximate well some of the components of the prototype, or the coarse solver cannot damp some of the components in the prototype. In the case of TG method when the coarse-grid solver is exact (see Algorithm 3.1), we may ignore the second reason. However, we would want to incorporate the prototype in Range(P) aiming at improvement of the method so that it can handle the unreduced left-over components of $\mathbf{x}_m$.

For our problem with changing PDE coefficient, we would not want to solve the local generalized eigenvalue problems for every single of the many realizations of the coefficient. We extend the idea of adaptive AMG to avoid building the hierarchy from scratch. Assume we are given the global stiffness matrix A computed for a realization k of the coefficient. We have built the interpolation matrix P using the method above and thus a TG mapping (preconditioner) B_{TG} is constructed that uses P, A, $A_c = P^T AP$, and a smoother M. We also have all the components needed for building P; those are: agglomerated elements, corresponding aggregates, and all the local vectors from the eigenvalue problems, their restrictions, and the output of the SVD. Next, we assume that a new global stiffness matrix A' (for a realization k' of the coefficient) is computed (in particular, we may have $A' = A$). We assemble the local stiffness matrices for the new coefficient, i.e., we compute A'_T and their corresponding diagonals D'_T for each $T \in \mathcal{T}_H$. Using the original P, we compute $\widehat{A}_c = P^T A' P$ and we have a new TG mapping $\widehat{B}_{TG}$ that uses the old P, and new A', $\widehat{A}_c$, and M', where M' is the version of the smoother corresponding to A'. We apply the TG method with $\widehat{B}_{TG}$ to $A'\mathbf{x} = \mathbf{0}$ with a random initial guess $\mathbf{x}_0$, i.e., we compute

$$\mathbf{x}_r = \left(I - \widehat{B}_{TG}^{-1} A'\right) \mathbf{x}_{r-1},\ r \geq 1.$$

We monitor the convergence in A'-norm. If the convergence is not satisfactory, we take the last iterate $\mathbf{x}^{bad} = \mathbf{x}_m$. Our purpose is to augment/modify P using $\mathbf{x}^{bad}$ and, thus, produce P' which is expected to make the TG-cycle behave better for the new system. This is done with the adaptive procedure that we describe next.

For each agglomerated element $T \in \mathcal{T}_H$, we restrict $\mathbf{x}^{bad}$ to T, i.e., we take the entries of $\mathbf{x}^{bad}$ restricted to the fine-grid dofs in T, and we form $\mathbf{x}_T^{bad}$. Consider the low eigenvectors from the original generalized eigenvalue problem (3.2) used to construct the original local tentative interpolation matrix, i.e., we consider $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$. We assume that the eigensolver produced $\mathbf{q}_s$ that are D_T-orthonormal. We orthonormalize (see Section 2.4) $\mathbf{x}_T^{bad}$ to the system of vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$, w.r.t. the inner product induced by D_T, and, thus, we get the system of vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}, \widehat{\mathbf{q}}$, where $\widehat{\mathbf{q}}$ may be $\mathbf{0}$. Next, we build $Z_T = [\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}, \widehat{\mathbf{q}}]$, if $\widehat{\mathbf{q}} \neq \mathbf{0}$, or $Z_T = [\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}]$, otherwise. Then, we solve the generalized eigenvalue problem

$$A'_T \mathbf{q}'_r = \lambda'_r D'_T \mathbf{q}'_r,$$

in the subspace spanned by $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}, \widehat{\mathbf{q}}$. That is, we solve the generalized eigenvalue problem

$$\left(Z_T^T A'_T Z_T\right) \overline{\mathbf{q}}_r = \overline{\lambda}_r \left(Z_T^T D'_T Z_T\right) \overline{\mathbf{q}}_r. \tag{3.10}$$

Then, we proceed with building the new tentative interpolation matrix. That is, we take the first m''_T eigenvectors (in the lower part of the spectrum) from (3.10) (it will become clear how we actually do it in Chapter 4); we transform them back to the original space by $\mathbf{q}'_r = Z_T \overline{\mathbf{q}}_r$; we restrict them to the respective aggregate $\mathcal{A}_T \subset T$; and after applying SVD (to filter out possible linear dependence) we get the adapted local tentative interpolation matrix $\overline{P'}_{\mathcal{A}} = \left[\mathbf{p}'_1, \ldots, \mathbf{p}'_{m'''_T}\right]$ ($m'''_T \leq m'_T + 1$). It is clear that $\overline{P'}_{\mathcal{A}}$ is an adaptation of the original one since it is based on the original vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$ (i.e., an already known information is reused) but also incorporates the new vector $\mathbf{x}_T^{bad}$.

After applying the above procedure to all agglomerates $T \in \mathcal{T}_H$ and producing the respective adapted local tentative prolongation operators, i.e., $\overline{P'}_{\mathcal{A}_1}, \ldots, \overline{P'}_{\mathcal{A}_{n_a}}$, we get as a result the adapted (global) tentative interpolation operator

$$\overline{P'} = \begin{bmatrix} \overline{P'}_{\mathcal{A}_1} & 0 & & 0 \\ 0 & \overline{P'}_{\mathcal{A}_2} & & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \ldots & \overline{P'}_{\mathcal{A}_{n_a}} \end{bmatrix}.$$

The final adapted interpolation operator is obtained by smoothing the tentative one. That is, $P' = S'\overline{P'}$, where S' is the version of the smoother corresponding to A'.

At the end, we have the adapted TG operator B'_{TG} that uses P', A', $A'_c =$

$(P')^T A'P'$, and M'.

Remark 3.5. Since $Z_T^T D'_T Z_T$ is s.p.d. it is clear that if the eigensolver produces $\left(Z_T^T D'_T Z_T\right)$-orthonormal $\overline{\mathbf{q}}_r$ vectors when solving (3.10), then the final ones, $\mathbf{q}'_r$, will also be D'_T-orthonormal.

Remark 3.6. If for all $T \in \mathcal{T}_H$ it happens that $\mathbf{q}_{m_T+1}$ is $\mathbf{0}$, i.e., $\mathbf{x}_T^{bad}$ is linearly dependent on $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$, then $\mathbf{x}^{bad}$ is actually not "bad". If it is really "bad", then for at least one T the vector $\mathbf{x}_T^{bad}$ will be linearly independent of $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$.

Remark 3.7. We may apply the above procedure repeatedly until the final adapted TG operator is efficient enough for A', i.e., until a certain convergence factor is achieved.

Remark 3.8. For a further consequent realization k'' of the coefficient and corresponding matrix A'' we may adapt either the original B_{TG} or the intermediately adapted one B'_{TG}.

Remark 3.9. We note, that we solve generalized eigenvalue problems for each agglomerated element, however, unlike building the hierarchy from scratch, here the eigenvalue problems are of much smaller size (we solve them in a subspace) which requires significantly less time.

More specific details are presented in Chapter 4, where, also, experiments and results are shown.

4

The Adaptive Smoothed Aggregation Spectral AMG Method (aSA AMGe): Implementation Details and Results

This chapter is devoted to the method of our main interest, namely the adaptive version of the method from [10] in the case of changing PDE coefficient, described in the previous chapter. Here, we describe our implementation in details and show results from the numerical experiments. We finish with some conclusions.

More specifically, in Section 4.1 we consider the procedure of building the agglomerated elements, the corresponding aggregates and local stiffness matrices. They are necessary prerequisites for producing the interpolation operator and hence for the overall TG algorithm we employ. Section 4.2 contains implementation details related to the interpolation matrix and the smoother we use in the TG algorithm. In Section 4.3 we illustrate the SA-AMGe method with numerical results without involving adaptation. Section 4.4 describes the adaptation procedure and its implementation. In Section 4.5, we present our main numerical results for the adaptive SA-AMGe method. Section 4.6 provides some concluding words including considerations related to improvements and further development of the method.

4.1 Agglomerated Elements, Aggregates, and Local Stiffness Matrices

The procedure of building the interpolation operator (and, thus, the coarse space) relies on the presence of appropriate partitioning of the domain into sets of fine-grid elements – agglomerated elements. It also requires the corresponding aggregates and local stiffness matrices. The purpose of this section is to treat this topic. We review here the definition of agglomerated elements and provide some algorithms to construct them. Next, we consider the related notion of aggregates and we finish the section by inspecting the assembly of the local stiffness matrices.

We begin first with the more general notion of *relation tables*.

4.1.1 Relation Tables

The topology of the mesh can be described by means of incidence relations. Here we consider these relations and a way they can be represented (implemented) in practice. The presentation follows [32, 31].

Suppose we have some numbering of the elements, vertices, edges, faces, etc. of the given fine mesh. It is quite natural to have them in practice, i.e., many mesh generators do provide this information. The edges in 2D have the same role as the faces in 3D, thus we are going to use the word "face" in both cases and our considerations will be applicable in both cases. A typical relation in the mesh is *"element i has vertex j"*. This relation can be represented as a boolean matrix with number of rows equal to the number of elements and number of columns equal to the number of vertices. Thus, the relation *"element i has vertex j"* is expressed by a nonzero entry at position (i, j) of the matrix. These matrices are called *relation tables*. They are sparse and thus can be stored and worked with using the popular CSR format. The relation described above is "element_vertex". We can easily get the relation "vertex_element", representing the relation *"vertex i belongs to element j"*, by transposing "element_vertex". Similarly other relations can be produced, like: "element_face" and the respective "face_element", etc.

We can also use other standard matrix operations on these tables. Namely, we can compute the product

$$\text{"element\_element"} \;=\; \text{"element\_face"} \;\times\; \text{"face\_element"}.$$

It represents the symmetric relation *"element i and element j have a common face"*. That is, it is precisely the *incidence matrix* of the dual graph of the fine mesh (see Section 2.5).

Similarly, we can compute

$$\text{"vertex\_vertex"} \;=\; \text{"vertex\_element"} \times \text{"element\_vertex"}\ ,$$

which represents the symmetric relation *"vertex i and vertex j belong to a common element"*. That is, it is precisely the incidence matrix of the nodal graph of the fine mesh. It also has the same sparsity structure as the global stiffness and mass matrices (we remind that for us dofs and vertices coincide since we use linear finite elements).

There is one more application of the relation tables. They can be used to define a local numbering of the mesh components and, also, mappings between local and global numberings. Consider the relation table "element_vertex" as an example. Taking the i-th row of this table and considering the order in which the vertices of element i appear in the row, we can assign local numbers to these vertices (e.g., in the 2D case when triangles are used the numbers will be from 1 to 3). If we want to find the global number of a vertex belonging to element i that has a local number k, then we look at the i-th row, find the k-th nonzero element, and its (global) column index gives its global number.

Conversely, if we take the global number j of a vertex, then we can look at the j-th row of the table "vertex_element", find all the elements it belongs to, and for every single element i we can look at the i-th row of "element_vertex", and find the local number (based on its order) of the vertex corresponding to the j-th column.

Remark 4.1. Instead of using the order in which the vertices appear in the row it is more practical to use the order in which they appear in the column index array of the CSR representation of the table, which may not coincide with the order they appear in the row. The idea is still the same as above.

Remark 4.2. The finite element package MFEM has a suitable CSR format implementation specially adjusted for representing relation tables, since it has a tool-set for working with meshes. It also generates several relation tables, e.g., it generates "element_face" and "element_element". Many relations are available but not all of them represented in a table form. If we need for example "element_vertex" we can still build it from the information present and we can even keep the ordering, i.e., the local numbering.

4.1.2 Agglomerated Elements

The first thing we need to obtain when we implement the method of our interest, after we have the mesh and the linear system, is the set of *agglomerated elements*. We describe next the procedure we use.

Definition 4.3. We define an *agglomerated element (AE)* simply as a (connected) partition of the dual graph of the fine mesh (see Section 2.5). Actually, we are going to use the words "agglomerated element" and "partition" interchangeably.

Having the "element_element" table we practically have the dual graph and we can use the graph partitioner METIS to generate a partitioning and, thus, obtain the agglomerated elements. The API (Application Programming Interface) of METIS expects as input the adjacency structure of the graph represented in CSR format. That is, it can take the table "element_element" and produce the partitioning. The output is a relation between the elements and the partitions they belong to. It is not represented in a table form. Instead, the output is an array s.t. the value of the cell with index i in the array is the number of the partition to which element i (of the fine mesh) belongs. Thus, a numbering of the agglomerated elements is introduced and also the mapping *"element i belongs to agglomerated element j (the value of the i-th cell of the array)"* is given. That is, effectively, we have the relation "element_AE".

Remark 4.4. Actually, the call to the API of METIS that partitions the dual graph of the fine mesh is part of the MFEM library. In MFEM it is used for partitioning the mesh so that parallel algorithms can be used, i.e., to distribute the load among many processors. A few modifications of the compiler options

allow us to use the code for our purposes. It has the feature to check if some of the resulting partitions turn out to be empty. In that case it splits the largest partitions into two parts so that the desired number of (non-empty) partitions are produced. Although we may consider the desired number of partitions as an upper bound and, thus, we may discard the empty partitions and work with the number of non-empty partitions given by METIS, we decided to use the available procedure in MFEM.

Since the array produced by METIS resembles the table "element_AE" we easily generate the table "AE_element". As a result we have a local numbering of the elements inside an AE. Having "element_vertex" we compute

$$\text{"AE\_vertex"} \;=\; \text{"AE\_element"} \;\times\; \text{"element\_vertex"},$$

which represents the relation *"agglomerated element i has an element which has vertex j"*. Thus, we have a local numbering of the vertices inside an AE. We note that in terms of vertices the agglomerated elements are overlapping.

A sample partitioning of the irregular mesh into 10 agglomerated elements is shown in fig. 4.1. The smallest agglomerated element has 625 elements and the largest has 658 elements which is fairly enough around the mean 640.

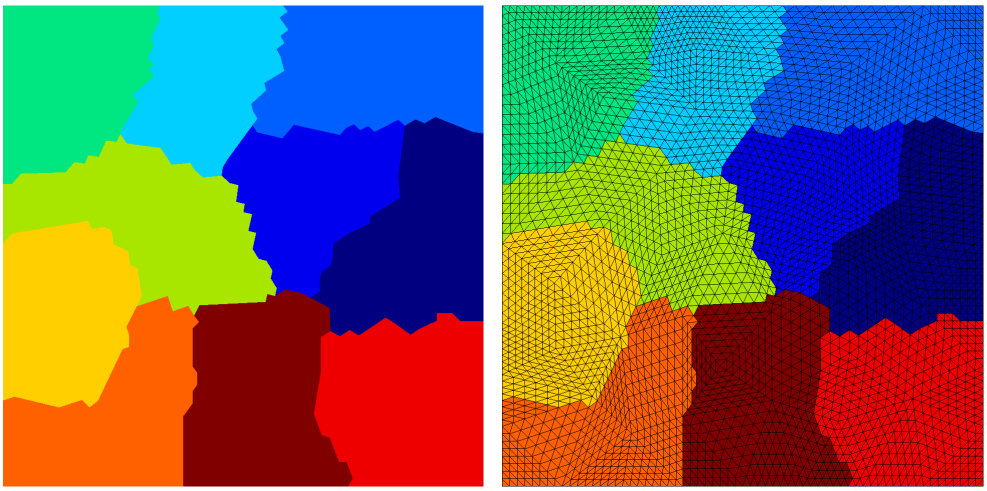
Remark 4.5. The version of METIS we use may return non-connected partitions (see Section 2.5).

4.1.3 Aggregates

The second thing after obtaining the agglomerated elements is to construct the *aggregates*. We present here the procedure we use and, also, make few additional considerations regarding some nonstandard situations that may occur in the implementation and discuss some possible improvements.

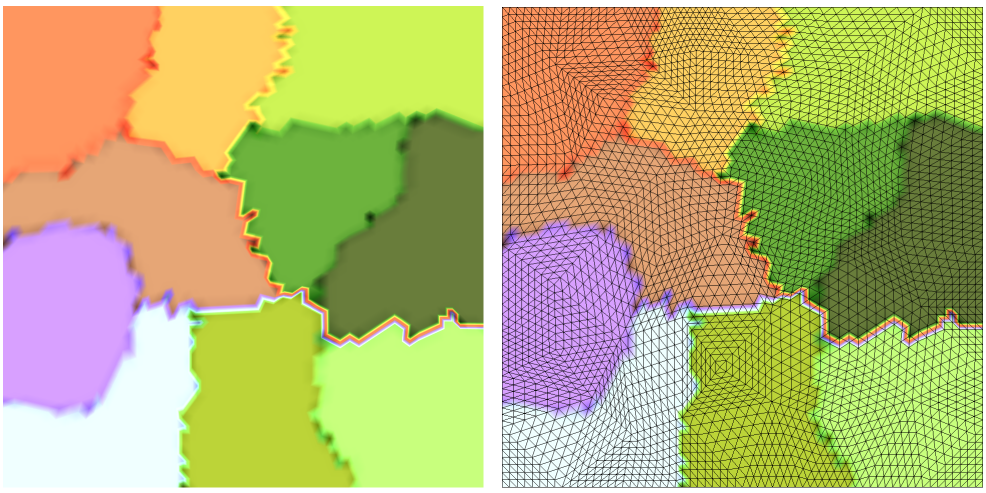
Aggregates are disjoint sets of dofs (vertices, in our case) that cover the whole set of dofs. We want our aggregates to be subordinate to the agglomerated elements. That is, there is *one-to-one correspondence* between the agglomerated elements and the aggregates in a way that for each agglomerated element $T \in \mathcal{T}_H$ there is precisely one aggregate $\mathcal{A}_T$ s.t. it contains only vertices belonging to elements in T. We may consider that we have two types of vertices – those belonging to only one agglomerated element and those lying on the boundary (*interface*) between two or more agglomerated elements. Surely, the vertices lying entirely in T become part of $\mathcal{A}_T$ so we consider them distributed. The vertices lying on an interface may be part of any of the aggregates corresponding to the agglomerated elements having this interface. We have to distribute them somehow.

[10] recommends to assign the interface vertices to the aggregate with the largest value of the PDE coefficient (assuming we have access to the coefficient). In our case, we have many realizations of the coefficient and we want to build the agglomerated elements and respective aggregates once for all. Thus, we should either use some information regarding the distribution of the coefficient



(a) Mesh not displayed **(b)** Mesh displayed

Figure 4.1: 10 AEs on the irregular mesh with 6400 elements and 3321 vertices.



(a) Mesh not displayed **(b)** Mesh displayed

Figure 4.2: 10 aggregates on the irregular mesh with 6400 elements and 3321 vertices.

or we may try to distribute the vertices as uniformly as we can. We choose the second approach. We use a *greedy* heuristics to achieve that. That is, we loop over all the interface vertices, for each vertex we consider all the aggregates that compete to own it and we assign the vertex to an aggregate with a minimal current number of vertices. Thus, we construct the desired aggregates and the relation table "aggregate_vertex" which also provides us a local numbering of the vertices in an aggregate.

As an example, in fig. 4.2 we show 10 aggregates covering the given irregular mesh. These are the aggregates corresponding to the agglomerated elements seen in fig. 4.1. The smallest aggregate has 330 vertices and the largest one has 334 vertices which is fairly close to the mean 332.

Remark 4.6. An aggregate may turn out to be disconnected (in the sense of the nodal graph) even when its corresponding agglomerated element is connected (in the sense of the dual graph).

Empty Aggregates

Some considerations related to the case when the above algorithm returns empty aggregates are in order. At the end, we discuss possible improvements of the aggregation procedure.

The greedy methods have well known disadvantages due to the use of local criteria of choice. It may return a solution that is far from optimal and the result, obviously, depends quite a lot on the order in which the interface vertices are visited. The largest issue is that it may return empty aggregates even in the cases when the number of aggregates is not very close to the number of vertices (naturally, we assume the number of aggregates is less than the number of vertices). Currently, we have not implemented a solution to this issue; we consider it quite unlikely in the case when we use aggressive enough coarsening (large agglomerated elements). The latter is appropriate in our two-level setting. Thus, if it happens to have one or more empty aggregates the current program aborts (i.e., it cannot continue). Still, we are giving some considerations related to this issue and its resolution.

One option we suggest is as follows. If we end up with empty aggregates we cannot simply discard them, since we would also need to discard the corresponding agglomerated elements and redistribute their elements to the neighboring AEs. This is too complicated and confusing so we do not consider it. There is another somewhat easier way which is still not very elegant. We may consider the undirected graph with nodes the aggregates where two aggregates are adjacent if they (used to) compete for vertices (in most cases this will be a *planar graph*). We want to "steal" vertices to add to an empty aggregate from another aggregate s.t. stealing will not make it empty, e.g., it has to have more than one vertex. We simply use a *pathfinding* algorithm in the graph (e.g., *depth-first search* or *breadth-first search*) and find a path between the empty aggregate and an appropriate aggregate we can "steal" from. Then, in

a "domino-like" manner we transfer one or more (if possible) vertices to the empty aggregate. Thus, no aggregate on the way changes its number of vertices except for the empty one and the one we "steal" from.

The above solution is far from elegant. We may look at the problem from another point of view.

Definition 4.7 (see [25])**.** Consider a given set U and a family $\mathcal{F}$ of subsets of U s.t. this family contains at least one subfamily (called *packing*) $\mathcal{C} \subseteq \mathcal{F}$ of disjoint sets which covers the set U. We define the problem of *(maximal) set packing* as the problem of finding a packing $\mathcal{C}$ s.t. it maximizes the cardinality $|\mathcal{C}|$.

In general, this is a NP-complete problem (see [20, 25]). If we identify U with $\mathcal{N}_h$ and $\mathcal{F}$ is the set of all possible aggregates corresponding to the set of agglomerated elements we have, then, with some reservation, we may consider the problem of building the aggregates as an instance of a set packing problem. The NP-completeness directs us to reformulate the problem as another NP-complete problem for which we have a tool that offers a (approximate) solution. Namely, it directs us to reformulate the problem in terms of graph partitioning and use METIS. Actually, graph partitioning is a much better way to look at the problem when we consider the goals that are posed for set packing and graph partitioning. That is, the goal of set packing is to maximize $|\mathcal{C}|$ which for us means nothing since it is always n_a (the number of AEs), while one of the goals of graph partitioning is the partitions to be more or less equal which is what we want. Since METIS is a very sophisticated tool that uses multilevel heuristics which implies a global criterion, compared to the local criterion in the greedy algorithm, we may expect a much more balanced (and thus closer to the optimum) solution of the problem of building the aggregates. Of course, it still might be possible to get empty aggregates (if METIS outputs empty partitions), but, considering the better heuristics, it should be much more unlikely and mainly in case the number of aggregates is too close to the number of vertices. Thus, this is one more option to be exploited to improve the method of building aggregates.

4.1.4 Stiffness Matrices for Local Problems

We now consider the procedure of building the local stiffness matrices corresponding to the agglomerated elements. We present an efficient implementation that uses both the element matrices and the assembled global matrix.

For each $T \in \mathcal{T}_H$ we need to build the corresponding $n_T \times n_T$ local stiffness matrix $A_T = \left[a_{ij}^T\right]_{i,j=1}^{n_T}$, where n_T is the number of dofs in T. In general, A_T is sparse symmetric positive semi-definite. We assume that the global stiffness matrix $A = [a_{ij}]_{i,j=1}^{n}$ has the essential boundary conditions imposed and they are imposed only globally, i.e., the essential boundary conditions are not imposed on the element stiffness matrices of the elements owning dofs that

are on $\Gamma_{\mathfrak{D}}$. This is what MFEM provides. We build A_T as a sparse matrix (i.e., in CSR format) component by component, i.e., we compute the value of each a_{ij}^T (due to the symmetry it equals the value of a_{ji}^T) in a way that we get a local stiffness matrix which has the essential boundary conditions imposed in case the agglomerated element owns dofs that are on $\Gamma_{\mathfrak{D}}$.

Assembling each component a_{ij}^T of A_T from the element stiffness matrices of the elements in T that share the dofs (i.e., the ones that contribute to the value of a_{ij}^T) with local (for T) numbers i and j may be inefficient, since these elements may be too many, especially in 3D. But the majority of the values of the components are already assembled in A and we use those values without assembling them again. The relations "AE_vertex", "vertex_AE", "vertex_element" and the global stiffness matrix A (which also acts as "vertex_vertex") are sufficient for all numberings and mappings between numberings that are used for assembling A_T. The principle of using the entries of A is simple. Namely, if at least one of the dofs i and j is entirely inside T (i.e., not on ∂T) or at least one of them is on $\Gamma_{\mathfrak{D}}$ then we set $a_{ij}^T = a_{pq}$, where p and q are the global numbers corresponding respectively to i and j. Surely, it includes the case of $i = j$. The value of a_{ij}^T is explicitly assembled from element stiffness matrices only in the case when both i and j (including the case of $i = j$) lie on ∂T and none of them is on $\Gamma_{\mathfrak{D}}$.

Remark 4.8. In reality, considering the adjacency structure of the nodal graph presented by A (with imposed essential boundary conditions) the dofs on $\Gamma_{\mathfrak{D}}$ are either connected to the neighboring dofs by zero values or they are not connected to any other dof except themselves (when these zero values are excluded from the CSR representation of A). Thus, a_{ij}^T, when $i \neq j$ and at least one of i and j is on $\Gamma_{\mathfrak{D}}$, is either zero (in the former case) or not considered at all (in the latter case, since i and j are not adjacent in the presented nodal graph) and this explicitly (in the former case) or implicitly (in the latter case) leaves a_{ij}^T equal to zero in the CSR representation of A_T. Having in mind that a_{ii}^T is copied from A when i is on $\Gamma_{\mathfrak{D}}$ we get that the essential boundary conditions are imposed on A_T when T owns dofs on $\Gamma_{\mathfrak{D}}$.

Remark 4.9. The implementation is a bit more general allowing the use of A which does not have the essential boundary conditions imposed and building local stiffness matrices also without the essential boundary conditions being imposed when the agglomerated element owns dofs that are on $\Gamma_{\mathfrak{D}}$.

4.2 Interpolation Matrix and Smoother

In this section we consider first the interpolation matrix P. Next, we treat the smoother M.

4.2.1 Interpolation Matrix

We present here a few details about the interpolation matrix. The general construction is found in Section 3.5.3.1.

As already said, we solve for each $T \in \mathcal{T}_H$ a generalized eigenvalue problem of the form:

$$A_T \mathbf{q}_r = \lambda_r D_T \mathbf{q}_r,\ r = 1, \ldots, n_T. \tag{4.1}$$

Then we select the first $m_T \leq n_T$ eigenvectors corresponding to the lower part of the spectrum. Having $\lambda_{\max} = \lambda_{n_T}$, based on a given tolerance $\theta \in (0, 1]$ (a parameter that we choose) we determine m_T. Namely, we choose m_T s.t. $\lambda_r \leq \theta\lambda_{\max}$ for $r = 1, \ldots, m_T$ and $\lambda_r > \theta\lambda_{\max}$ for $r = (m_T + 1), \ldots, n_T$.

The implementation works with any s.p.d. matrix in place of D_T in (4.1). There are several ways (that we have implemented) to compute the desired eigenvectors. We can compute all n_T eigenvectors and eigenvalues using the LAPACK routine `DSYGV` and thus knowing $\lambda_{\max}$ we take the desired subset of vectors; alternatively, we can compute the maximal eigenvalue using ARPACK and ARPACK++, then using the LAPACK routine `DSYGVX` we compute only the desired eigenvectors. These LAPACK routines return D_T-orthonormal eigenvectors (or orthonormal w.r.t. any s.p.d. matrix in place of D_T) which is a property we desire for the adaptation (see Section 3.5.3.2). LAPACK expects the matrices represented in a format for dense matrices, i.e., we convert the CSR-represented matrices into dense ones.

We have also implement another approach which is the one actually used for our experiments (to follow). Namely, in place of D_T in (4.1) we use the weighted ℓ_1-smoother corresponding to A_T denoted by $\mathscr{D}_T$. That is, we solve

$$A_T \mathbf{q}_r = \lambda_r \mathscr{D}_T \mathbf{q}_r,\ r = 1, \ldots, n_T,$$

and knowing that $\lambda_{\max}$ is at most 1 we take m_T s.t. $\lambda_r \leq \theta$ for $r = 1, \ldots, m_T$ and $\lambda_r > \theta$ for $r = (m_T + 1), \ldots, n_T$. Thus, we can directly apply `DSYGVX`. The estimates in Section 3.5.3.1 still hold when we use $\mathscr{D}_T$ instead of D_T due to their spectral equivalence.

Remark 4.10. If using the method above determines m_T to be 0, we take $m_T = 1$, i.e., we always choose at least one eigenvector.

Remark 4.11. The implementation allows the use of D_T, $\mathscr{D}_T$, or any other s.p.d. matrix.

The rest of the procedure is as described in Section 3.5.3.1. The only detail worth noting is that when restricting the eigenvectors to the aggregate $\mathcal{A}_T$ we also build a simple mapping from the indices of the restricted vectors (which correspond to the dofs in the aggregate) to the global numbers of the dofs which points out to the exact row of the global tentative interpolation matrix $\overline{P}$ where we have to place each row of the local tentative interpolation matrix $\overline{P}_{\mathcal{A}}$. Thus, we easily assemble $\overline{P}$ in CSR format, since each of its rows it has only elements coming from exactly one local tentative interpolation matrix.

As we have already said, instead of smoothing $\overline{P}$ by the polynomial smoother shown in Section 3.5.3.1, we have chosen the simple option $P = \left(I - D^{-1}A\right)\overline{P}$, where D, as before, is the weighted ℓ_1-smoother corresponding to A.

4.2.2 Implementation of the polynomial smoother

This section describes our implementation of the polynomial smoother introduced in Section 3.5.3.1, "The choice of M, D and the upper bound b".

We have, for the given polynomial p_ν (see Section 3.5.3.1)

$$M^{-1} = \left[I - p_\nu\left(D^{-1}A\right)\right] A^{-1}.$$

Since $p_\nu(0) = 1$, we have the factorization

$$p_\nu(t) = \prod_{j=1}^{3\nu+1}\left(1 - \frac{t}{\tau_j}\right),$$

where $\tau_1, \ldots, \tau_{3\nu+1}$ are the roots of $p_\nu(t)$.

We have the following algorithm which implements the actions of M^{-1}.

Algorithm 4.12. *Given an iterate* $\mathbf{x}_0$ *we compute* $\mathbf{x}_{3\nu+1} = \mathbf{x}_0 + M^{-1}(\mathbf{b} - A\mathbf{x}_0)$ *in the following steps:*

$$\text{for } j = 1, \ldots, (3\nu+1) \text{ compute } \mathbf{x}_j \text{ from}$$
$$\mathbf{x}_j = \mathbf{x}_{j-1} + \frac{1}{\tau_j} D^{-1}\left(\mathbf{b} - A\mathbf{x}_{j-1}\right).$$

That is, the algorithm involves $3\nu+1$ sweeps of an iteration of Jacobi type.

To verify Algorithm 4.12, we proceed as follows. Consider the error at step j,

$$\begin{aligned} A^{-1}\mathbf{b} - \mathbf{x}_j &= A^{-1}\mathbf{b} - \mathbf{x}_{j-1} - \frac{1}{\tau_j} D^{-1}\left(\mathbf{b} - A\mathbf{x}_{j-1}\right) \\ &= \left(A^{-1}\mathbf{b} - \mathbf{x}_{j-1}\right) - \frac{1}{\tau_j} D^{-1}A\left(A^{-1}\mathbf{b} - \mathbf{x}_{j-1}\right) \\ &= \left(I - \frac{1}{\tau_j} D^{-1}A\right)\left(A^{-1}\mathbf{b} - \mathbf{x}_{j-1}\right). \end{aligned}$$

Thus by recursion on j, we obtain

$$A^{-1}\mathbf{b} - \mathbf{x}_{3\nu+1} = \underbrace{\prod_{j=1}^{3\nu+1}\left(I - \frac{1}{\tau_j} D^{-1}A\right)}_{p_\nu(D^{-1}A)}\left(A^{-1}\mathbf{b} - \mathbf{x}_0\right).$$

Thus,

$$\begin{aligned} \mathbf{x}_{3\nu+1} &= \left[I - p_\nu\left(D^{-1}A\right)\right] A^{-1}\mathbf{b} + p_\nu\left(D^{-1}A\right)\mathbf{x}_0 \\ &= M^{-1}\mathbf{b} - \left[I - p_\nu\left(D^{-1}A\right)\right] A^{-1}A\mathbf{x}_0 + \mathbf{x}_0 \\ &= \mathbf{x}_0 + M^{-1}\left(\mathbf{b} - A\mathbf{x}_0\right). \end{aligned}$$

The roots of $p_\nu(t)$ are (see [9])

$$\cos^2\left(\frac{j}{2\nu+1}\pi\right), \text{ for } j = 0, \ldots, 2\nu,$$
$$\sin^2\left(\frac{j}{2\nu+1}\pi\right), \text{ for } j = 1, \ldots, \nu,$$

which are precisely the extrema of $T_{2\nu+1}\left(\sqrt{t}\right)$ and the roots of $\frac{T_{2\nu+1}\left(\sqrt{t}\right)}{\sqrt{t}}$.

4.3 Numerical Results without Adaptation

We are now ready to present results from numerical experiments with our implementation of the SA spectral AMGe method. The results here are only for the method without involving adaptation. We first describe the setting of the experiments and then we give the actual results.

4.3.1 Preliminaries

We first outline the setting of the experiments (tolerances used, norms in which we measure convergence and errors, etc.)

Unless we are given the exact solution of $A\mathbf{x} = \mathbf{b}$ or we solve $A\mathbf{x} = \mathbf{0}$, we generally do not have access to the (algebraic) error. Using the fact that the error solves the defect equation, i.e., $A\mathbf{e}_i = \mathbf{r}_i$, we get $\|\mathbf{e}_i\|_A = \|\mathbf{r}_i\|_{A^{-1}}$. We approximate $\|\mathbf{r}_i\|_{A^{-1}}$ by $\|\mathbf{r}_i\|_{B_{TG}^{-1}}$, since B_{TG}^{-1} is a good approximation to A^{-1} (for a well–convergent TG-method). Actually, the norms $\|.\|_{A^{-1}}$ and $\|.\|_{B_{TG}^{-1}}$ are equivalent if B_{TG} is properly chosen. In the computation we use the relation $B_{TG}^{-1}\mathbf{r}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$.

We apply the method to $A\mathbf{e} = \mathbf{0}$ with a random initial iterate aiming to observe the error behavior, since in this case we have access to the error. Consider two given tolerances: the *relative tolerance* $\mathtt{t}_r$ and the *absolute tolerance* $\mathtt{t}_a$. The stopping criteria is: stop after step m, i.e., $\mathbf{e}_m$ is the last iterate, if m is the smallest integer s.t. $\frac{\|\mathbf{e}_m\|_A}{\|\mathbf{e}_0\|_A} \leq \mathtt{t}_r$ or $\|\mathbf{e}_m\|_A \leq \mathtt{t}_a$ hold. We use the notation $\widetilde{\rho}_{TG} = \frac{\|\mathbf{e}_m\|_A}{\|\mathbf{e}_{m-1}\|_A}$ (see Section 2.2.3).

In case we solve $A\mathbf{x} = \mathbf{b}$ the stopping criteria is: stop after step m, i.e., $\mathbf{x}_m$ is the last iterate, if m is the smallest integer s.t. $\frac{\left(B_{TG}^{-1}\mathbf{r}_{m-1},\mathbf{r}_{m-1}\right)}{\left(B_{TG}^{-1}\mathbf{r}_0,\mathbf{r}_0\right)} \leq \mathtt{t}_r$ or $\left(B_{TG}^{-1}\mathbf{r}_{m-1},\mathbf{r}_{m-1}\right) \leq \mathtt{t}_a$ hold, where $(\cdot,\cdot)$ is the Euclidean inner product.

Let (1.1) (with a fixed coefficient k) have an exact solution u. Also, let the solution of the linear system $A\mathbf{v} = \mathbf{b}$ be $\mathbf{v}$, that gives rise to the finite element approximation v_h to the exact solution u. Let $\mathbf{u}$ be the vector with the values of u at the mesh nodes, i.e., $\mathbf{u}$ is the coefficient vector of $\mathcal{I}u$ which is the linear interpolant of u at the nodes of the mesh (we assume that the solution u is smooth enough). Then, $\|\mathbf{u}-\mathbf{v}\|_A = \sqrt{a_k\left(\mathcal{I}u - v_h, \mathcal{I}u - v_h\right)}$ and, thus, it is not the real energy norm of the error (the difference between our approximation and the exact solution) which is $\sqrt{a_k\left(u - v_h, u - v_h\right)}$. Still, we are going to use $\|\mathbf{u}-\mathbf{v}\|_A$ to measure the error and check for the presence of approximation as we decrease h.

For the experiments we use $\mathtt{t}_r = 10^{-12}$, $\mathtt{t}_a = 0$, $\theta = 0.01$, and $\varepsilon = 10^{-8}$, where ε is the threshold used within the SVD (see Section 2.3) and the Gram-Schmidt orthogonalization (see Section 2.4). The coarse solver uses the *Cholesky method* implemented by the LAPACK routine `DPOSV` (which, as expected, requires a dense matrix as input).

4.3.2 Results

We present here some results from experiments with our implementation of the TG SA spectral AMGe method.

Approximation

We show that the numerical solutions have approximation, i.e., the error decreases as we decrease h.

We consider (1.1) with fixed $k = 1$, $g = 0$, $\Gamma_{\mathfrak{D}} = \partial\Omega$, $\Gamma_{\mathfrak{N}} = \emptyset$, and $f = 2x(1-x) + 2y(1-y)$. We use the regular mesh. The r.h.s. f is chosen such that the exact solution is $u = x(1-x)y(1-y)$. We measure $\|\mathbf{u} - \mathbf{v}\|_A$ where $A\mathbf{v} = \mathbf{b}$ is assembled and solved on a sequence of meshes. We use ten (10) AEs, and $\nu = 6$ (i.e., $3\nu+1 = 19$) and a zero initial guess. The well-known estimate of the energy norm of the error is $O(h)$. The results are presented in table 4.1. Figure 4.3 compares the estimate $O(h)$ and the error we compute. As we see the computed error is actually better than the expected $O(h)$ (i.e., some *superconvergence* phenomenon is observed). We can also see that the number of iterations, m, is fairly small and changes very little as we increase the number of fine-grid dofs even though we keep the same ν (note however the increase of the number of coarse dofs).

For the case corresponding to the last row of table 4.1 we show in fig. 4.4 a sample distribution of the eigenvalues coming from the generalized eigenvalue problem for a local stiffness matrix. They are plotted in a logarithmic scale to show their distribution according to their order of magnitude.

Coefficient Jumps

We show experiments illustrating the behavior of the method in the presence of large jumps in the PDE coefficient k.

We consider $g = 0$, $\Gamma_{\mathfrak{D}} = \Gamma_E \cup \Gamma_W$, $\Gamma_{\mathfrak{N}} = \Gamma_N \cup \Gamma_S$ (see Section 1.4.5), $\nu = 6$ (i.e., $3\nu+1 = 19$), 200 AEs and we use the irregular mesh with 102400 elements and 51681 vertices. Figure 4.5 shows a sample checkerboard-like partitioning of the fine-grid elements (which changes slightly when the mesh is coarser or finer). We set $k = 1$ on the gray parts and $k = 10^c$ on the white parts, and vary the contrast c; $c = -12, -9, -6, -3, 0, 3, 6, 9, 12$. We apply the method to $A\mathbf{e} = \mathbf{0}$ with random initial iterates. The results are presented in table 4.2. We can see that the convergence factor is fairly insensitive to the contrast c.

We finish this paragraph by showing in fig. 4.6 a sample distribution of the eigenvalues of a local stiffness matrix when $c = 12$ and when 200 AEs are used.

Smooth Errors and the Coarse Space

We present examples that illustrate the behavior of the algebraically smooth error and discuss the quality of the constructed coarse space based on the parameters of the method most notably the construction of aggregates and the effect of smoothing the tentative interpolant.

h	$\lvert\mathcal{N}_h\rvert$	$\lvert\mathcal{T}_h\rvert$	# coarse dofs	m	$\left(B_{TG}^{-1}\mathbf{r}_m, \mathbf{r}_m\right)$	$e_h = \lVert\mathbf{u} - \mathbf{v}_m\rVert_A$
0.176777	81	128	10	4	1.6145×10^{-20}	0.00190437
0.0883883	289	512	10	4	1.96677×10^{-20}	0.000485122
0.0441942	1089	2048	12	8	6.26752×10^{-18}	0.000121859
0.0220971	4225	8192	39	8	1.41627×10^{-16}	3.05057×10^{-5}
0.0110485	16641	32768	124	8	9.9065×10^{-17}	7.63065×10^{-6}

Table 4.1: The error on a sequence of regular meshes using $k = 1$, $\nu = 6$, and 10 AEs.

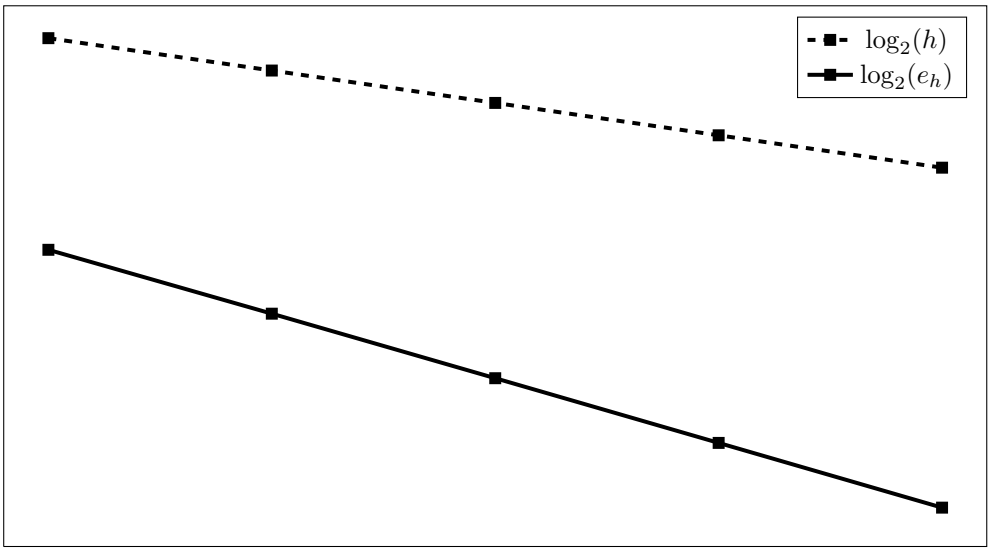


Figure 4.3: $\log_2$ of h and e_h.

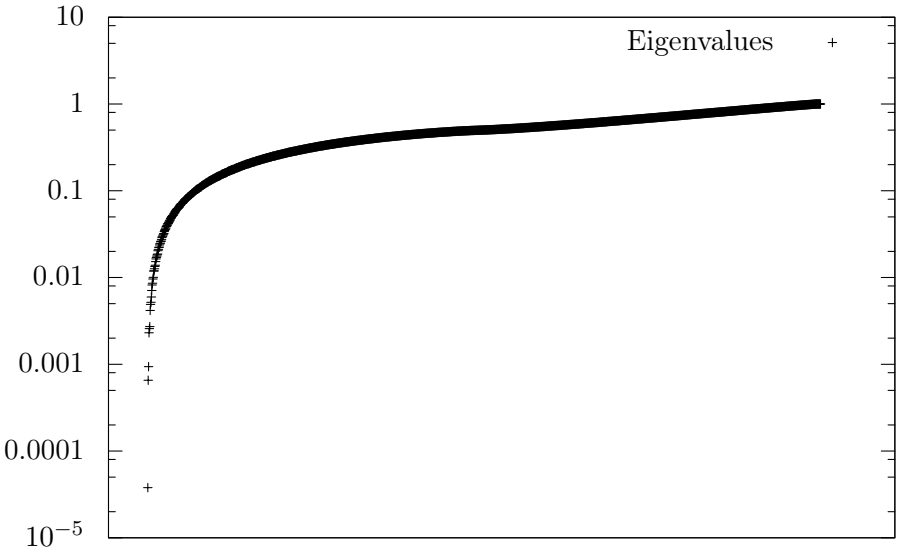


Figure 4.4: The distribution in a logarithmic scale of the eigenvalues of a local stiffness matrix on the regular mesh with 32768 elements and 16641 vertices when $k = 1$.

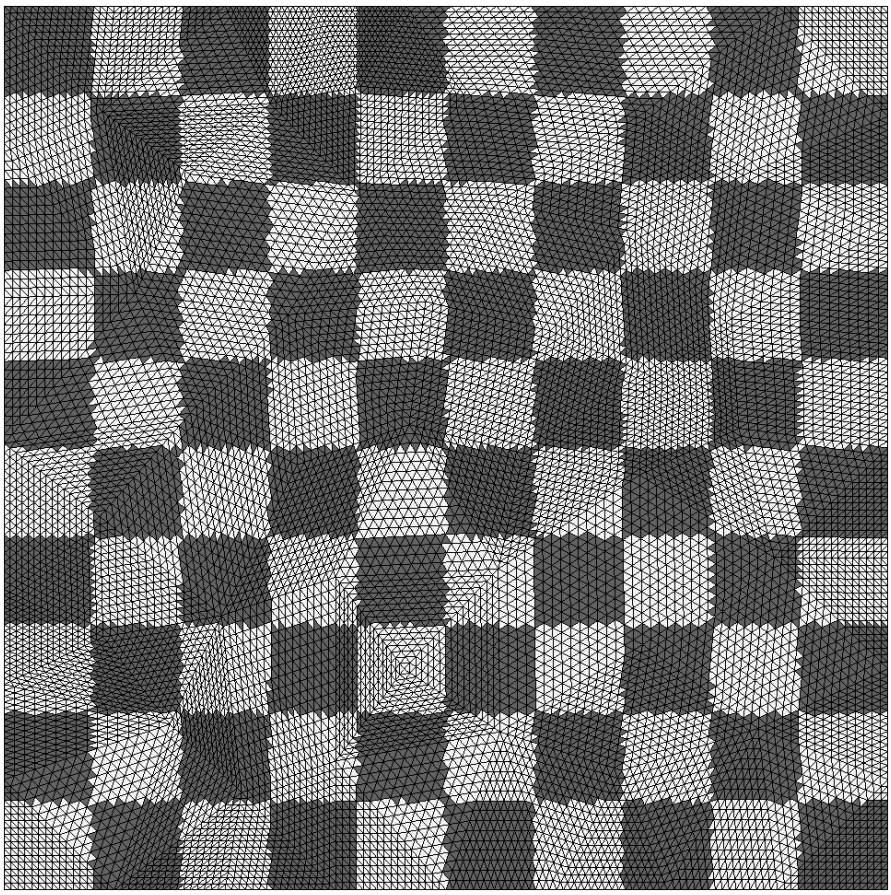


Figure 4.5: The distribution of the values of the coefficient on the irregular mesh with 25600 elements and 13041 vertices.

c	# coarse dofs	m	$\widetilde{\rho}_{TG}$	$\|\mathbf{e}_m\|_A$
-12	697	42	0.618	7.92645×10^{-11}
-9	697	43	0.620	6.42419×10^{-11}
-6	697	43	0.620	7.66067×10^{-11}
-3	696	41	0.616	6.47778×10^{-11}
0	638	27	0.487	1.19774×10^{-10}
3	689	41	0.630	2.31201×10^{-9}
6	689	53	0.725	7.68107×10^{-8}
9	689	60	0.724	2.3428×10^{-6}
12	689	60	0.724	6.83945×10^{-5}

Table 4.2: The convergence and the coefficient jumps on the irregular mesh with 102400 elements and 51681 vertices using 200 AEs.

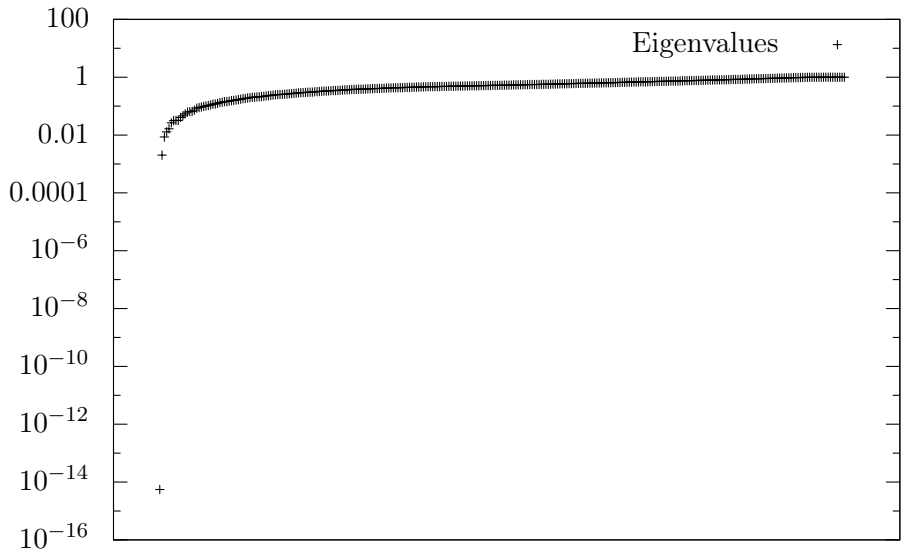
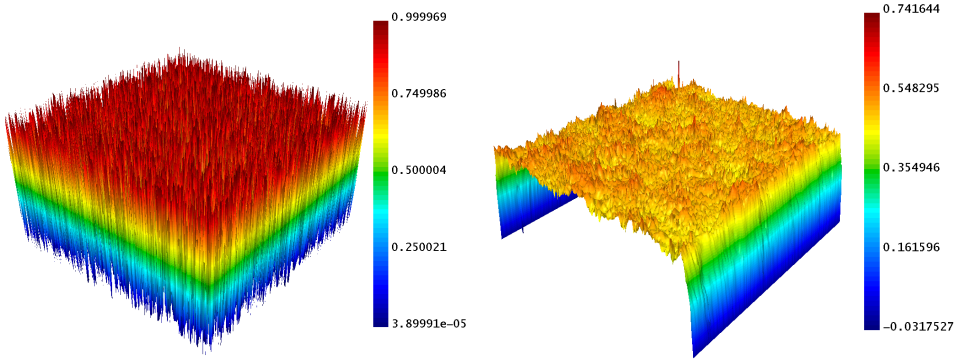
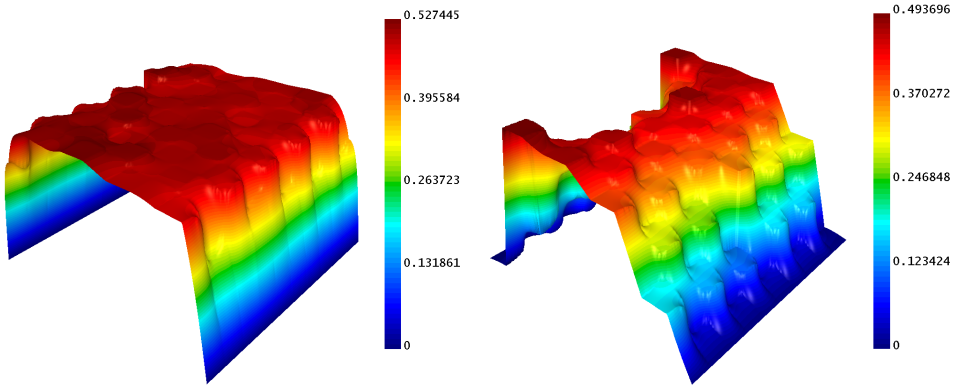


Figure 4.6: The distribution in a logarithmic scale of the eigenvalues of a local stiffness matrix on the irregular mesh with 102400 elements and 51681 vertices using 200 AEs and $c = 12$.



(a) The initial error

(b) The error after 1 sweep of M



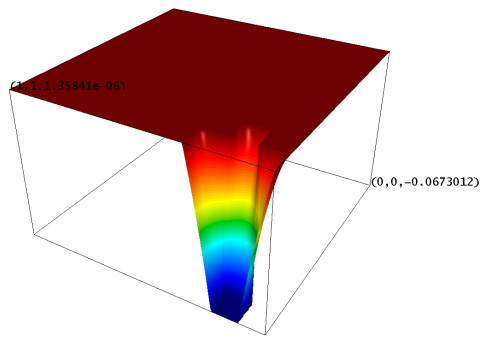
(c) The error after 5 sweeps of M

(d) The error after 200 sweeps of M

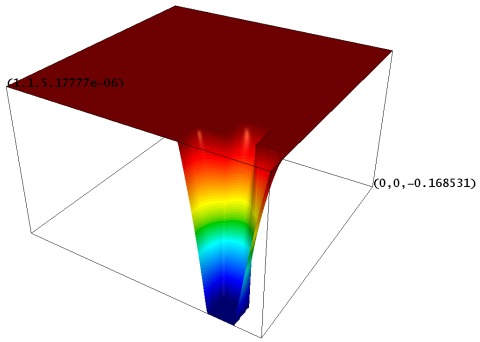
Figure 4.7: The smoothing effect of the smoother M on the error with $\nu = 6$, $c = 6$ and using the irregular mesh with 102400 elements and 51681 vertices.

θ	S	ν	# coarse dofs	$\lvert A_c \rvert$	m	$\widetilde{\rho}_{TG}$	$\lVert \mathbf{e}_m \rVert_A$
0.01	$I - D^{-1}A$	6	774	14320	2840	0.995	8.99665×10^{-8}
0.01	$I - D^{-1}A$	12	774	14320	2729	0.995	9.08379×10^{-8}
0.01	$s_\nu\left(D^{-1}A\right)$	6	774	29448	297	0.949	8.83595×10^{-8}
0.01	$s_\nu\left(D^{-1}A\right)$	12	774	55746	54	0.766	7.66148×10^{-8}
0.03	$I - D^{-1}A$	6	1679	66715	793	0.983	9.07097×10^{-8}
0.03	$I - D^{-1}A$	12	1679	66715	829	0.983	9.09787×10^{-8}
0.03	$s_\nu\left(D^{-1}A\right)$	6	1679	137909	84	0.825	7.89149×10^{-8}
0.03	$s_\nu\left(D^{-1}A\right)$	12	1679	259683	25	0.518	8.67915×10^{-8}
0.05	$I - D^{-1}A$	6	2530	150862	111	0.887	8.17631×10^{-8}
0.05	$I - D^{-1}A$	12	2530	150862	116	0.886	9.00872×10^{-8}
0.05	$s_\nu\left(D^{-1}A\right)$	6	2530	312174	11	0.210	6.45282×10^{-8}
0.05	$s_\nu\left(D^{-1}A\right)$	12	2530	587610	7	0.086	3.60355×10^{-8}
0.09	$I - D^{-1}A$	6	4175	407891	52	0.772	8.72622×10^{-8}
0.09	$I - D^{-1}A$	12	4175	407891	50	0.771	8.35374×10^{-8}
0.09	$s_\nu\left(D^{-1}A\right)$	6	4175	843881	7	0.084	1.21693×10^{-8}
0.09	$s_\nu\left(D^{-1}A\right)$	12	4175	1586143	5	0.021	6.14306×10^{-9}
0.01, 0.09	$I - D^{-1}A$	6	1133	52291	79	0.784	7.35024×10^{-8}
0.01, 0.09	$s_\nu\left(D^{-1}A\right)$	6	1133	100885	11	0.148	5.57028×10^{-8}

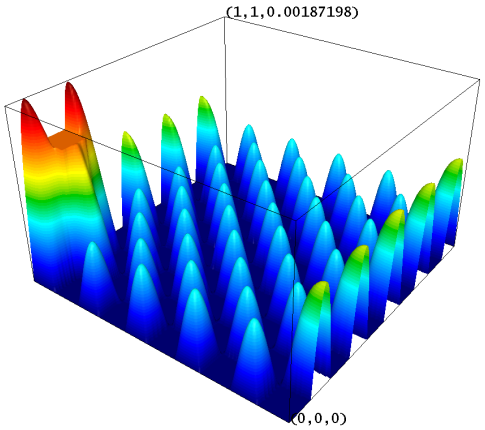
Table 4.3: The convergence on the irregular mesh with 102400 elements and 51681 vertices using 300 AEs, $c = 6$ and varying ν, S, and θ.



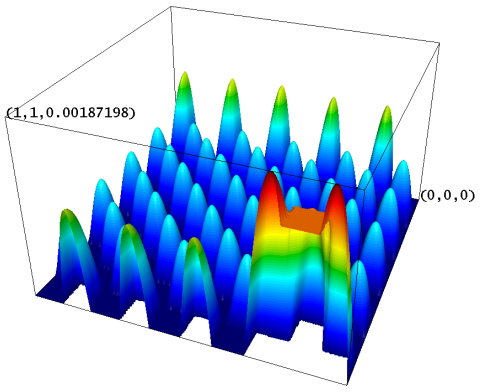
(a) The "bad" vector after 50 iterations using $S = I - D^{-1}A$



(b) The "bad" vector after 10 iterations using $S = s_\nu\left(D^{-1}A\right)$



(c) The solution



(d) The solution

Figure 4.8: "Bad" vectors on the irregular mesh with 102400 elements and 51681 vertices using 300 AEs, $c = 6$, and $\nu = 6$; and the solution of the equation.

S	ν	# coarse dofs	$\lvert A_c\rvert$	m	$\widetilde{\rho}_{TG}$	$\lVert \mathbf{e}_m \rVert_A$
$I - D^{-1}A$	6	689	16473	53	0.725	8.41505×10^{-8}
$I - D^{-1}A$	12	689	16473	46	0.670	7.84846×10^{-8}
$s_\nu\left(D^{-1}A\right)$	6	689	29177	10	0.117	1.18052×10^{-8}
$s_\nu\left(D^{-1}A\right)$	12	689	51995	6	0.026	2.16641×10^{-8}

Table 4.4: The convergence on the irregular mesh with 102400 elements and 51681 vertices using 200 AEs, $c = 6$, $\theta = 0.01$ and varying ν and S.

θ	S	ν	# coarse dofs	$\lvert A_c\rvert$	m	$\widetilde{\rho}_{TG}$	$\lVert \mathbf{e}_m \rVert_A$
0.01	$I - D^{-1}A$	6	584	22638	11776	0.999	9.06992×10^{-8}
0.01	$I - D^{-1}A$	12	584	22638	12688	0.999	9.09531×10^{-8}
0.01	$s_\nu\left(D^{-1}A\right)$	6	584	33314	482	0.965	8.7672×10^{-8}
0.01	$s_\nu\left(D^{-1}A\right)$	12	584	51148	142	0.887	8.56982×10^{-8}
0.0103	$I - D^{-1}A$	6	595	23441	193	0.928	9.03082×10^{-8}
0.0103	$I - D^{-1}A$	12	595	23441	200	0.924	8.77889×10^{-8}
0.0103	$s_\nu\left(D^{-1}A\right)$	6	595	34435	51	0.764	7.11794×10^{-8}
0.0103	$s_\nu\left(D^{-1}A\right)$	12	595	52769	15	0.300	3.24926×10^{-8}

Table 4.5: The convergence on the irregular mesh with 102400 elements and 51681 vertices using 100 AEs, $c = 6$ and varying ν, S, and θ.

S	ν	# coarse dofs	$\lvert A_c\rvert$	m	$\widetilde{\rho}_{TG}$	$\lVert \mathbf{e}_m \rVert_A$
$I - D^{-1}A$	6	580	22514	23	0.489	7.86781×10^{-8}
$I - D^{-1}A$	12	580	22514	21	0.392	4.26719×10^{-8}
$s_\nu\left(D^{-1}A\right)$	6	580	32196	9	0.144	6.30571×10^{-8}
$s_\nu\left(D^{-1}A\right)$	12	580	50822	6	0.019	2.98351×10^{-8}

Table 4.6: The convergence on the irregular mesh with 102400 elements and 51681 vertices using *alternative* 100 AEs, $c = 6$, $\theta = 0.01$ and varying ν and S.

S	ν	# coarse dofs	$\lvert A_c\rvert$	m	$\widetilde{\rho}_{TG}$	$\lVert \mathbf{e}_m \rVert_A$
$I - D^{-1}A$	6	672	15624	4619	0.997	9.08026×10^{-8}
$I - D^{-1}A$	12	672	15624	5006	0.997	9.0812×10^{-8}
$s_\nu\left(D^{-1}A\right)$	6	672	28440	365	0.955	8.93305×10^{-8}
$s_\nu\left(D^{-1}A\right)$	12	672	49320	83	0.822	7.58487×10^{-8}

Table 4.7: The convergence on the irregular mesh with 102400 elements and 51681 vertices using *alternative* 200 AEs, $c = 6$, $\theta = 0.01$ and varying ν and S.

θ	S	ν	# coarse dofs	$\lvert A_c\rvert$	m	$\widetilde{\rho}_{TG}$	$\lVert \mathbf{e}_m \rVert_A$
0.01	$I - D^{-1}A$	6	689	16209	1353	0.989	9.08763×10^{-8}
0.01	$I - D^{-1}A$	12	689	16209	1356	0.989	9.10478×10^{-8}
0.01	$s_\nu\left(D^{-1}A\right)$	6	689	29071	95	0.826	7.76864×10^{-8}
0.01	$s_\nu\left(D^{-1}A\right)$	12	689	51903	25	0.484	8.10407×10^{-8}
0.015	$I - D^{-1}A$	6	896	27322	759	0.978	8.95747×10^{-8}
0.015	$I - D^{-1}A$	12	896	27322	680	0.978	8.99256×10^{-8}
0.015	$s_\nu\left(D^{-1}A\right)$	6	896	48888	54	0.735	6.93061×10^{-8}
0.015	$s_\nu\left(D^{-1}A\right)$	12	896	87226	15	0.347	8.94873×10^{-8}

Table 4.8: The convergence on the irregular mesh with 102400 elements and 51681 vertices using 200 AEs (with *alternative* aggregates), $c = 6$ and varying ν, S, and θ.

$\lvert\mathcal{N}_h\rvert$	$\lvert\mathcal{T}_h\rvert$	# coarse dofs	OC	m	$\widetilde{\rho}_{TG}$	$\lVert\mathbf{e}_m\rVert_A$
3321	6400	227	1.13271	60	0.703	2.11807×10^{-8}
13041	25600	302	1.04099	73	0.743	4.07153×10^{-8}
51681	102400	665	1.04493	50	0.697	6.95167×10^{-8}
205761	409600	1899	1.08678	36	0.617	1.13402×10^{-7}

Table 4.9: The convergence on a sequence of irregular meshes using 200 AEs, $\nu = 6$, and $c = 6$.

Figure 4.7 shows how the error looks after applying the smoother M in the case when $c = 6$. We see that the algebraically smooth error is not uniformly geometrically smooth; it follows the pattern of the jumps of the coefficient.

Consider the test problem with $f = 1$, $g = 0$, $\Gamma_{\mathfrak{D}} = \Gamma_E \cup \Gamma_W$, $\Gamma_{\mathfrak{N}} = \Gamma_N \cup \Gamma_S$, $c = 6$ and use the irregular mesh with 102400 elements and 51681 vertices. Using the notation $P = S\overline{P}$ for the smoothed interpolant, we present the results in tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8, when we vary the solver parameters. In the tables, $|A_c|$ denotes the number of nonzero elements of A_c (for comparison $|A| = 353179$). We observe another quite interesting phenomenon. We explain first the last two rows of table 4.3. When we run the iteration process with $\nu = 6$ for a few iterations, we get the vectors in figs. 4.8a and 4.8b which we call "bad", since the convergence factor is bad then (see Section 2.2.4). In figs. 4.8c and 4.8d we can see the actual solution and we also can see which part of the solution corresponds to the peak in the "bad" vectors. We set $\theta = 0.09$ for each $T \in \mathcal{T}_H$ s.t. the average of the coordinates of the dofs in T is in the set $\{(x, y) : 0 \leq x \leq 0.35,\ 0.75 \leq y \leq 1\} \subset \overline{\Omega}$ and we set $\theta = 0.01$ for the rest, i.e., we increase the tolerance roughly around the peak in the "bad" vectors. Thus, we get the results in the last two rows of table 4.3.

Tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8 show that in almost all cases when we use $S = I - D^{-1}A$ increasing ν (and thus the degree of the polynomial $p_\nu(t)$ used for the smoother M) brings little or no improvement to the convergence factor. Thus, we may conclude that the source of good or bad convergence (in our experiments here) is the quality of the coarse space. Quite naturally, larger coarse space does not necessary mean better coarse space. Tables 4.5 and 4.6 contain results when two different partitionings[1] (both of them with 100 AEs) are used. We can see a serious difference in the results. The situation is the same with tables 4.4 and 4.7. The used alternative partitionings are almost the same as the original ones but still the results are quite different. Actually, if we compare tables 4.4 and 4.8, where the same agglomerated elements are used and the change is only in the corresponding aggregates, then we see that the results are also quite different. Any change to the agglomerated elements results in different aggregates. Particularly, the change corresponding to the results in table 4.8 is as simple as reversing the order in which the interface

[1] All the partitionings that we use here have connected (in the sense of the dual graph of the fine mesh) partitions.

dofs are visited during the construction of the aggregates using the greedy approach (see Section 4.1.3). It is natural for the agglomerated elements and the aggregates to have impact on the coarse space, since they are major building blocks.

As the theory shows (supported also by tables 4.3 to 4.8) both the interpolation smoother S and the eigenvectors of the local stiffness matrices determine the quality of the coarse space. We can also see that the interpolation smoother increases the number of nonzero elements of A_c. This is due to the fact that smoothing the coarse basis leads to enlarged support of the coarse basis functions and thus to decreased sparsity of A_c. We see that in the cases when we use $S = I - D^{-1}A$ (and increasing ν, the number of smoothing steps, (almost) does not improve the convergence) the corresponding results with $S = s_\nu\left(D^{-1}A\right)$ (giving rise to a better smoothed interpolant) show noticeable improvement when ν is increased. Thus, we may conclude that the source is the improved quality of the coarse space due to the better energy stability of the smoothed interpolant. The situation is similar when we increase θ. As already said in Section 3.5.3.1, we should choose m_T, for each $T \in \mathfrak{T}_H$, large enough. This actually means (see [10]) that we should choose m_T s.t. λ_{m_T+1} scales with $\left(\frac{h}{H}\right)^2$. Only the combination of appropriate m_T, ν and interpolation smoother S (as the theory shows and the results in tables 4.3 to 4.8 support) leads to stable and acceptable convergence in all cases (including, stable w.r.t. the particular choice of agglomerated elements and aggregates). Often we get good results without too much tuning of the parameters and one may think that the conditions required by the theory for deducting the optimal spectral estimate are too pessimistic but at the same time in extreme cases, like our examples here, the convergence may be bad and may improve dramatically when appropriate parameters are selected. Thus, the requirements concerning these parameters and posed by the theory are quite sharp if we consider a case that tends to behave like "the worst case". An example of a case when the results are considerably stable w.r.t. the choice of the parameters is when we set $\Gamma_{\mathfrak{D}} = \partial\Omega$ and $\Gamma_{\mathfrak{N}} = \emptyset$ ($\theta = 0.01$, $\nu = 6$, and 300 AEs are used), then we obtain $\widetilde{\rho}_{TG} = 0.779$ with $S = I - D^{-1}A$, and $\widetilde{\rho}_{TG} = 0.231$ when $S = s_\nu\left(D^{-1}A\right)$ (compare with table 4.3). Particularly, in this case the "bad" vectors in figs. 4.8a and 4.8b cease to exist.

We saw that we may end up with some deficiency in the approximation properties of the coarse space depending on the parameters we choose. We also saw that appropriate parameters may easily and dramatically improve the convergence. The adaptation (see Sections 3.5.3.2 and 4.4) is another way to improve the coarse space and thus overcome the deficiency of the method. Currently, we consider the fixed choices $S = I - D^{-1}A$ and $\theta = 0.01$.

Sensitivity to h

Here we investigate the dependence of the convergence factor when the method is applied on a sequence of fine meshes.

We introduce the following notion.

Definition 4.13. The quantity: $\frac{|A_c|+|A|}{|A|}$ is referred to as *operator complexity (OC)*

Consider the test problem with $g = 0$, $\Gamma_{\mathfrak{D}} = \partial\Omega$, $\Gamma_{\mathfrak{N}} = \emptyset$, $c = 6$ (using, as before, the checkerboard-like coefficient), $\nu = 6$ (i.e., $3\nu + 1 = 19$), 200 AEs and use the irregular mesh. We apply the method for $A\mathbf{e} = \mathbf{0}$ with random initial guess. The results are presented in table 4.9. We can see that the convergence factor even improves as h decreases, since we keep $\theta = 0.01$ instead of decreasing it with $\left(\frac{h}{H}\right)^2$ (which results in increased coarse size problem).

4.4 Adaptation

We consider here a few details related to the adaptation. The main idea and procedure are described in Section 3.5.3.2. We start with description of the way we compute the prototype and then we describe the way we build the adapted interpolation matrix. We finish this section by proposing an overall adaptation strategy. We use the same notation as in Section 3.5.3.2.

4.4.1 The Prototype

The first thing we need for the adaptation procedure is the vector $\mathbf{x}^{bad}$. We describe here the way we compute it.

As already described in Section 3.5.3.2, starting with a random initial iterate $\mathbf{x}_0$, we compute

$$\mathbf{x}_r = \left(I - \widehat{B}_{TG}^{-1} A'\right) \mathbf{x}_{r-1},\ r \geq 1.$$

We choose an integer parameter $\mu \geq 1$. We run μ iterations as above; we normalize the initial guess and after each iteration we normalize the iterate in A'-norm (i.e., we make $\|\mathbf{x}_r\|_{A'} = 1$ for $r = 0, \ldots, \mu$). We define $\mathbf{x}^{bad} = \mathbf{x}_\mu$. This is the vector $\mathbf{x}^{bad}$, used in the adaptation procedure as described in Section 3.5.3.2.

Our goal is to define a procedure that will (automatically) adapt the currently available hierarchy (if necessary) so that a target rate of convergence is achieved. For this purpose, we use the following strategy (similar to the one in [8]): we choose an integer parameter μ_0 ($1 \leq \mu_0 \leq \mu$ and typically $\mu_0 \ll \mu$); we run μ iterations with a random A'-normalized initial iterate (i.e., $\|\mathbf{x}_0\|_{A'} = 1$) without further normalizing the consecutive iterates; we monitor (see Section 2.2.3)

$$\rho_{asympt} = \left(\frac{\|\mathbf{x}_\mu\|_{A'}}{\|\mathbf{x}_{\mu-\mu_0}\|_{A'}}\right)^{\frac{1}{\mu_0}}$$

and if $\rho_{asympt} > \mathfrak{t}_{\text{CF}}$ and $\|\mathbf{x}_\mu\|_{A'} > \mathfrak{t}_{\text{error}}$, we let $\mathbf{x}^{bad} = \frac{\mathbf{x}_\mu}{\|\mathbf{x}_\mu\|_{A'}}$ and we adapt the hierarchy, otherwise the current hierarchy is considered efficient enough

and the current unchanged P is used. Here $\mathfrak{t}_{\text{CF}}$ and $\mathfrak{t}_{\text{error}}$ are given parameters (that we choose). They define the desired convergence rate. The complete strategy is described in Section 4.4.3.

Remark 4.14. The implementation uses also other strategies which determine how to compute $\mathbf{x}^{bad}$ (when to stop the iteration process) and when to use it for the adaptation, or completely stop the procedure if a desired rate of convergence is achieved. It also allows defining and using different strategies and criteria that are not currently present. This is one of the most complicated parts of the implementation.

Remark 4.15. We consider the computation of $\mathbf{x}^{bad}$ and the assessment of the convergence rate (and, thus, the necessity of adapting the hierarchy) together because they are naturally related, since when testing the method it exposes the component that the current method cannot handle. Since the smoother is fixed, what is left to improve the method is to incorporate the component $\mathbf{x}^{bad}$ into the adapted coarse space.

4.4.2 The Adapted Interpolation Matrix

After computing $\mathbf{x}^{bad}$ and deciding that adaptation is necessary we have to adapt the current hierarchy. We describe in details our implementation of the construction of the adapted interpolation matrix.

As we have already described in Section 3.5.3.2, for each $T \in \mathcal{T}_H$ we solve

$$\left(Z_T^T A'_T Z_T\right) \overline{\mathbf{q}}_r = \overline{\lambda}_r \left(Z_T^T D'_T Z_T\right) \overline{\mathbf{q}}_r. \tag{4.2}$$

Then for a given tolerance θ (a parameter that we choose), we take the first m''_T eigenvectors, where we select m''_T s.t. $\overline{\lambda}_r \leq \theta \lambda'_{\max}$ for $r = 1, \ldots, m''_T$ and $\overline{\lambda}_r > \theta \lambda'_{\max}$ for $r > m''_T$. Here $\lambda'_{\max}$ stands for the largest eigenvalue of the generalized eigenvalue problem $A'_T \mathbf{q}'_r = \lambda'_r D'_T \mathbf{q}'_r$. In general, we can compute it using ARPACK and ARPACK++, however, since we use $\mathscr{D}'_T$ instead of D'_T, we can use 1 as a sharp upper bound of $\lambda'_{\max}$. To solve (4.2), we use again the same LAPACK routines as described in Section 4.2.1.

Remark 4.16. If using the method above determines m''_T to be 0, we take $m''_T = 1$, that is, we select at least one eigenvector per agglomerate.

Remark 4.17. The implementation allows the use of D'_T, $\mathscr{D}'_T$, or any other given s.p.d. matrix.

The choice of θ

In what follows, we discuss some strategies to choose the tolerance θ.

Consider first the case when $A' = A$. For a given $T \in \mathcal{T}_H$, we have $Z_T^T D_T Z_T = I$, where I is the identity matrix of appropriate order (namely, of order n_T). In the adaptation step, we solve

$$Z_T^T A_T Z_T \overline{\mathbf{q}}_r = \overline{\lambda}_r \overline{\mathbf{q}}_r, \text{ for } r = 1, \ldots, z_T, \tag{4.3}$$

where z_T is the number of columns of Z_T, i.e., it is either m_T or $m_T + 1$. Consider first the case $z_T = m_T$. The vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$ are column vectors of Z_T.

Since $\mathbf{q}_r$ are eigenvectors of $D_T^{-1}A_T$, we have that $Z_T^T A_T Z_T = \operatorname{diag}(\lambda_r)_{r=1}^{m_T}$. That is, we end up with the same space as before (depending on the tolerance θ). Namely, if we use the same θ we get $m_T'' = m_T$ and, clearly, $\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}\} = \operatorname{span}\left\{\mathbf{q}'_1, \ldots, \mathbf{q}'_{m_T''}\right\}$. We conclude that in this case $\operatorname{Range}\left(\overline{P'}_{\mathcal{A}}\right) = \operatorname{Range}\left(\overline{P}_{\mathcal{A}}\right)$.

Consider now the case when we add a vector to the previous local subspace, i.e., we now have $z_T = m_T + 1$. The added vector $\mathbf{x}^{bad}$ gives rise to a vector $\widehat{\mathbf{q}} \in \mathbb{R}^{z_T}$ which is orthonormal, in the Euclidean inner product, to the system of vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$ and $Z_T^T A_T Z_T \widehat{\mathbf{q}} = \widehat{\lambda}\widehat{\mathbf{q}}$, where $\mathbf{q}_r \in \mathbb{R}^{z_T}$ are the unique vectors s.t. $Z_T \mathbf{q}_r = \mathbf{q}_r$, for $r = 1, \ldots, m_T$, and they are, clearly, orthonormal in the Euclidean inner product. Also, $Z_T\widehat{\mathbf{q}}$ is D_T-orthonormal to the system of vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$ and since it is a vector in $\operatorname{Range}(Z_T)$, we conclude that $Z_T\widehat{\mathbf{q}} = \pm\widehat{\mathbf{q}}$. Without loss of generality, we consider $Z_T\widehat{\mathbf{q}} = \widehat{\mathbf{q}}$ and this $\widehat{\mathbf{q}}$ is unique. It is straightforward then to see that $\widehat{\mathbf{q}} = [0, \ldots, 0, 1]^T$. If we show that $\widehat{\lambda} > \theta\lambda_{\max}$ (assuming we use the same θ), then we can conclude again that $m_T'' = m_T$, $\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}\} = \operatorname{span}\left\{\mathbf{q}'_1, \ldots, \mathbf{q}'_{m_T''}\right\}$, and $\operatorname{Range}\left(\overline{P'}_{\mathcal{A}}\right) = \operatorname{Range}\left(\overline{P}_{\mathcal{A}}\right)$. Since $\widehat{\mathbf{q}} \in \operatorname{span}\{\mathbf{q}_{m_T+1}, \ldots, \mathbf{q}_{n_T}\}$, we have

$$\widehat{\mathbf{q}} = \sum_{i=1}^{n_T - m_T} c_i \mathbf{q}_{m_T+i}.$$

The fact that $\widehat{\mathbf{q}}^T D_T \widehat{\mathbf{q}} = 1$ implies that $\sum_{i=1}^{n_T-m_T} c_i^2 = 1$. We have

$$\widehat{\lambda}\widehat{\mathbf{q}} = Z_T^T A_T Z_T \widehat{\mathbf{q}} = Z_T^T A_T \widehat{\mathbf{q}}$$

$$= \sum_{i=1}^{n_T-m_T} c_i Z_T^T A_T \mathbf{q}_{m_T+i} = \sum_{i=1}^{n_T-m_T} c_i \lambda_{m_T+i} Z_T^T D_T \mathbf{q}_{m_T+i}.$$

Since we, obviously, have $Z_T^T D_T \mathbf{q}_{m_T+i} = c_i\,[0, \ldots, 0, 1]^T$, we conclude that $\widehat{\lambda}$ is represented by the following *convex combination*

$$\widehat{\lambda} = \sum_{i=1}^{n_T-m_T} c_i^2 \lambda_{m_T+i}.$$

Therefore, $\widehat{\lambda} \geq \lambda_{m_T+1} > \theta\lambda_{\max}$. Hence, $\widehat{\mathbf{q}}$, and $\mathbf{x}^{bad}$ as well, do not contribute to the coarse space, if we use the old tolerance θ, since its contribution gets cut out.

Remark 4.18. The analysis above holds for general s.p.d. matrix D_T, and in particular applies to $\mathscr{D}_T$ (then we can use 1 instead of $\lambda_{\max}$).

In conclusion, we do not expect any improvement if the same θ is used, since the resulting coarse space does not change. Thus, if a required convergence rate is to be achieved, the only way to gain improvement is increasing θ. Similarly, when $A' \neq A$ if after several steps of adaptation we reach a state s.t. the coarse space remains the same, the only way to gain larger coarse space and hence improvement of the method, is to increase θ. This motivates our choice

to use an increasing function $\varphi(\theta)$ of $\theta \in (0,1)$ s.t. $\varphi(\theta) \leq 1$ for $\theta \in (0,1]$ and $\varphi(1) = 1$. We use such a function when implementing the adaptation strategy (see Section 4.4.3). Our choice is $\varphi(\theta) = (2-\theta)\theta$ but it is not hard-coded and our implementation allows to use any other function specified by the user.

4.4.3 Adaptation Strategy

Our goal is to construct a method that will (automatically) adapt the hierarchy so that a desired convergence rate would be guaranteed. In this section we formulate a somewhat conservative possible strategy for achieving this.

Given a current hierarchy, which might have been built from scratch for A or adapted for A (either from another matrix or from the same A), we want to adapt the current hierarchy so that we can solve a system with a (new) matrix A' with a predetermined convergence rate. If $A' = A$ or the hierarchy was already adapted once for A' (and $A' \neq A$) and we need to adapt it more to achieve the desired convergence rate, then we say that we perform *readaptation* or we *readapt*. If we do readaptation and we obtain as a result $m''_T = m_T$ for all $T \in \mathcal{T}_H$, then we say that no new vectors are introduced. Conversely, if we do readaptation and $m''_T > m_T$ for at least one $T \in \mathcal{T}_H$, then we have that at least one new vector is introduced into the coarse space.

We formulate now the following adaptation strategy:

Algorithm 4.19 (Adaptation strategy)**.** *Given a current hierarchy and a tolerance θ, we do:*

(i) Compute $\mathbf{x}^{bad}$ and check the convergence rate. If we have met a desired criteria, stop the procedure and use the current hierarchy. Otherwise, go to step (ii).

(ii) Adapt the hierarchy based on $\mathbf{x}^{bad}$ and the given θ. Continue with the next step.

(iii) If a new coarse hierarchy has been created or not enough readaptation steps have been performed (in case when no new vectors have been introduced), we go to step (i) to test the new hierarchy or get a new prototype vector. Otherwise, if a maximum allowed readaptation steps were performed without introducing new vectors, increase θ, i.e., $\theta \leftarrow \varphi(\theta)$, and go to step (ii).

We note that once a new vector has been introduced into the coarse space, we reset the counter for readaptation steps to zero. The maximum allowed readaptation steps (in case no new vectors have been introduced) is a global integer parameter that we choose. We denote it as χ. The above strategy is quite conservative w.r.t. θ. That is, it tries to exhaust all possible vectors in all AEs for the current θ, and it only increases θ if the coarse space does not change for all AEs and when the desired rate of convergence has still not been

$\widehat{\theta}$	ϑ	# coarse dofs	$\lvert A_c^a \rvert$	$\widetilde{\rho}_{TG}$
0.01	0	774	14320	0.995
0.0772553	3	1033	25233	0.992
0.148542	4	1063	26711	0.706
0.27502	5	1069	26975	0.697
1	—	1074	27208	0.391

Table 4.10: The convergence when a single adaptation step is applied for the unchanged matrix using 300 AEs, $\nu = 6$, $c = 6$, and $\mu = 10$ on the irregular mesh with 102400 elements and 51681 vertices. The initial coarse space is built using $\theta = 0.01$ and during the adaptation $\theta = \widehat{\theta}$ is used, computed calling ϑ times $\varphi(\theta)$ starting from $\theta = 0.01$.

achieved. We do not claim that this is the best strategy or even a very good one in many cases.

Remark 4.20. When readaptation is being performed $m_T'' = m_T$ does not generally imply that Range $\left(\overline{P'}_{\mathcal{A}}\right)$ and Range $\left(\overline{P}_{\mathcal{A}}\right)$ are the same. Often, when no new vectors are introduced the coarse space is actually changed. Still, we use $m_T'' = m_T$ as a criterion to decide when to increase θ, since the only way to achieve arbitrarily good rate of convergence is to increase θ at some point.

Remark 4.21. Assume we have constructed a hierarchy adapted for A' that satisfies the criteria for the desired convergence rate used in Algorithm 4.19 and let Algorithm 4.19 has exited with $\theta = \widehat{\theta}$. If we want to further adapt this hierarchy for a new A'', we can either start with $\theta = \widehat{\theta}$, or we can start with some globally predefined initial value for θ.

Remark 4.22. Assume we have created a hierarchy corresponding to A that we adapt for a new A'. It is likely to get a smaller coarse space compared to the one for A. Thus, some of the data from the hierarchy for A is lost and cannot be reused in further adaptation steps, e.g., from A' to A'' or even when we do readaptation for A'. Our implementation has the option to "accumulate" all vectors used for previous hierarchy adaptations and thus reuse all the history data. However, in general this becomes too expensive and is unclear if it can lead to a method that is competitive with simply building a hierarchy from scratch.

4.5 Numerical Results with Adaptation

We are now ready to present results from numerical experiments with the described adaptation strategy.

We first present results when adapting for a fixed PDE coefficient (and, respectively, fixed matrix, i.e., when $A' = A$), and next we present various experiments with changing coefficients (and, respectively, changing matrices).

In what follows, we use relative tolerance $\mathtt{t}_r = 10^{-12}$, absolute tolerance $\mathtt{t}_a = 0$, $\varepsilon = 10^{-8}$ (same as in Section 4.3), and, as described above, $\varphi(\theta) = (2 - \theta)\theta$. The coarse solver uses direct solver based on Cholesky factorization.

$\widehat{\theta}$	ϑ	# coarse dofs	$\lvert A_c^a\rvert$	$\widetilde{\rho}_{TG}$
0.01	0	774	14320	0.995
0.0772553	3	1027	25009	0.992
0.148542	4	1060	26576	0.545
0.27502	5	1067	26897	0.552
1	—	1074	27208	0.547

Table 4.11: The convergence when a single adaptation step is applied for the unchanged matrix using 300 AEs, $\nu = 6$, $c = 6$, and $\mu = 30$ on the irregular mesh with 102400 elements and 51681 vertices. The initial coarse space is built using $\theta = 0.01$ and during the adaptation $\theta = \widehat{\theta}$ is used, computed calling ϑ times $\varphi(\theta)$ starting from $\theta = 0.01$.

θ	# coarse dofs	$\lvert A_c\rvert$	$\widetilde{\rho}_{TG}$	$\lvert \widehat{A}_c\rvert$	$\widehat{\rho}_{TG}$
0.01	774	14320	0.995	29448	0.949
0.0772553	3629	307939	0.883	636671	0.159
0.148542	6557	1003851	0.715	2074291	0.069
0.27502	12279	3518105	0.075	7266145	0.002

Table 4.12: The convergence on the irregular mesh with 102400 elements and 51681 vertices using 300 AEs, $\nu = 6$, $c = 6$ and varying θ. The hierarchy is built from scratch and no adaptation is used. $\lvert \widehat{A}_c\rvert$ and $\widehat{\rho}_{TG}$ correspond to the case when the tentative interpolant is smoothed using $s_\nu\left(D^{-1}A\right)$.

The convergence factor is estimated (see Sections 4.3 and 2.2.3) by applying the solver to a system with a zero right-hand side letting $\widetilde{\rho}_{TG} = \frac{\|\mathbf{e}_m\|_A}{\|\mathbf{e}_{m-1}\|_A}$ (where m is the last iteration used).

4.5.1 Adaptation for a Fixed PDE Coefficient

We illustrate the adaptation performance with numerical experiments in the case when $A' = A$. We first show results using a single adaptation step. Next, we experiment with the adaptation strategy described in Section 4.4.3 and we also test an alternative simpler strategy.

Single Adaptation Step

In this paragraph we show results obtained when only one step of adaptation is applied. Also, a comparison is made with building the hierarchy from scratch.

Consider the case corresponding to the results in table 4.3 with contrast $c = 6$, polynomial degree $3\nu + 1$, $\nu = 6$, and spectral tolerance $\theta = 0.01$. We remind that we smooth the tentative interpolant using $I - D^{-1}A$. We build the hierarchy first, then we compute $\mathbf{x}^{bad}$ using $\mu = 10$, normalizing each iterate in A-norm (see Section 4.4.1), and we adapt the hierarchy using the same θ. As expected, the number of coarse dofs, $\widetilde{\rho}_{TG}$, and $\lvert A_c\rvert$ remain unchanged, since the coarse space remains the same (see Section 4.4.2, "The choice of θ"). This and more results are presented in table 4.10, where A_c^a denotes the coarse operator in the adapted coarse space. We next compare the

results in tables 4.10, 4.11 and 4.12. Considering the size of the coarse space and the sparsity of the coarse operator it is easy to see (for the experiments here) the advantage of the adaptation over building the hierarchy from scratch as far as the choice of θ for a fixed ν is concerned.

Remark 4.23. When computing the results in tables 4.10 and 4.11, for each row a separate newly computed $\mathbf{x}^{bad}$ is used.

Applying the Complete Adaptation Strategy

We present results from numerical experiments with the strategy proposed in Section 4.4.3. We also show results with an alternative and somewhat more aggressive (w.r.t. θ) strategy. In what follows we use $\mu = 20$, $\mu_0 = 4$, $\mathfrak{t}_{\text{CF}} = 0.1$, and $\mathfrak{t}_{\text{error}} = 10^{-14}$.

Consider again the case corresponding to the results in table 4.3. Applying the adaptation strategy described in Algorithm 4.19 we get the results shown in table 4.13.

Consider an agglomerated element $T \in \mathcal{T}_H$ and a current local tentative interpolant $\overline{P}_{\mathcal{A}}$ produced using eigenvectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$. When we adapt we obtain the eigenvectors $\mathbf{q}'_1, \ldots, \mathbf{q}'_{m''_T}$ that we use to produce the adapted local tentative interpolant $\overline{P'}_{\mathcal{A}}$ and, also, for the local eigenvalue problems if further adaptation is employed. We implement the option to replace $\mathbf{q}'_1, \ldots, \mathbf{q}'_{m''_T}$ by the old vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$ in the case when $m''_T = m_T$. This effectively results in $\overline{P}_{\mathcal{A}} = \overline{P'}_{\mathcal{A}}$ and if further adaptation is applied, $\mathbf{q}_1, \ldots, \mathbf{q}_{m_T}$ are used for the local eigenvalue problems. That is, the adaptation changes nothing in the hierarchy related to T. Using this option we get the results in table 4.14. The advantage of using this option is that the adaptation process is in a sense more predictable and the results are more stable. Namely, we observe monotone improvement of the convergence rate and the finial results for many executions of the experiments are almost the same, while without this option the convergence rate may jump back and forth with large amplitude and sometimes the final results may differ noticeably when the experiments are executed multiple times. However, we cannot draw a general conclusion whether this option results in a better or worse performance. For the experiments presented here, the performance is almost the same regardless of whether the option to keep the same vectors is used or not.

Introduce the following alternative, somewhat simpler aggressive (w.r.t. θ), strategy.

Algorithm 4.24 (Aggressive adaptation strategy)**.** *Given a current hierarchy and a tolerance θ, we do:*

(i) Compute $\mathbf{x}^{bad}$ and check the convergence rate. If we have met a desired criteria, stop the procedure and use the current hierarchy. Otherwise, go to step (ii).

(ii) Adapt the hierarchy based on $\mathbf{x}^{bad}$ *and the given* θ*. Continue with the next step.*

(iii) Increase θ*, i.e.,* $\theta \leftarrow \varphi(\theta)$*, and go to step (i).*

Applying Algorithm 4.24 we get the results in table 4.15. Instead of applying methods and criteria for increasing θ we can set $\theta = 1$ during the adaptation which results in adding all local restrictions of $\mathbf{x}^{bad}$ to the hierarchy and thus increasing the number of coarse dofs within the number of agglomerated elements on each hierarchy adaptation. Observe the results in table 4.16. The results show the advantage of the aggressive strategy over the conservative one (for the presented experiments here). The aggressive approach (including the use of $\theta = 1$ during the adaptation) is recommended in case the hierarchy built from scratch suffers from severe deficiencies resulting in very slow convergence (which is the case of the experiments presented here).

4.5.2 Adaptation for Changing PDE Coefficients

We present results from experiments with the adaptation in the case when the coefficients of the PDE change, i.e., the matrices of the arising linear systems change. We first illustrate the case when two coefficient realizations are involved and, next, we show results when multiple realizations of the coefficient take place.

We use the regular mesh with 131072 elements and 66049 vertices divided into 200 agglomerated elements. We also set $\nu = 6$, $\chi = 1$, $\mu = 15$, $\mu_0 = 4$, $\mathtt{t}_{\text{error}} = 10^{-14}$, $\theta = 0.01$ (when building the hierarchy from scratch), $f = 1$, $g = 0$ (see (1.1)), $\Gamma_{\mathfrak{D}} = \Gamma_E \cup \Gamma_W$, and $\Gamma_{\mathfrak{N}} = \Gamma_N \cup \Gamma_S$ (see Section 1.4.5).

All coefficients used in this section are generated (simulated) by a tool[1] that implements a Markov Chain Monte Carlo (or MCMC) algorithm and, thus, it outputs coefficients suitable for our experiments (any two consecutive realizations are in some sense "close").

Two Coefficient Realizations

This paragraph is devoted to numerical experiments with the adaptive SA-AMGe when two coefficient realizations are involved.

Denote the two coefficients we use as k' and k''. An illustration of these coefficients is shown in fig. 4.9. Consider the results in Table 4.17 and particularly the last row. It is a typical example of what we want to achieve by the adaptation strategy, namely, the resulting adapted coarse space to have smaller size compared to the original coarse space it has been derived from.

Consider tables 4.18 and 4.19. The former presents results using the original hierarchy (the one built from scratch) for k' which is being adapted to k'', while the latter shows the results when the original hierarchy for k' first gets readapted and then adapted to k''.

We can see the advantage of using adaptation over building the hierar-

[1] Implemented by Christian Ketelsen.

Parameters/Results	Value
θ when building from scratch	0.01
initial θ for the adaptation	0.01
χ	1
$\widetilde{\rho}_{TG}$ when building from scratch	0.995
# coarse dofs when building from scratch	774
OC when building from scratch	1.04055
# readaptation iterations	21
times $\mathbf{x}^{bad}$ computed	17
# $\varphi(\theta)$ calls	4
final θ	0.148542
final $\widetilde{\rho}_{TG}$	0.154
final # coarse dofs	1773
final OC	1.20997
final ρ_{asympt}	0.141077
final $\|\mathbf{x}_\mu\|_{A'}$	5.66873×10^{-21}

Parameters/Results	Value
θ when building from scratch	0.01
initial θ for the adaptation	0.01
χ	4
$\widetilde{\rho}_{TG}$ when building from scratch	0.995
# coarse dofs when building from scratch	774
OC when building from scratch	1.04055
# readaptation iterations	60
times $\mathbf{x}^{bad}$ computed	56
# $\varphi(\theta)$ calls	4
final θ	0.148542
final $\widetilde{\rho}_{TG}$	0.162
final # coarse dofs	1667
final OC	1.18718
final ρ_{asympt}	0.162195
final $\|\mathbf{x}_\mu\|_{A'}$	4.40642×10^{-19}

Parameters/Results	Value
θ when building from scratch	10^{-6}
initial θ for the adaptation	10^{-6}
χ	1
$\widetilde{\rho}_{TG}$ when building from scratch	> 0.999
# coarse dofs when building from scratch	338
OC when building from scratch	1.00766
# readaptation iterations	56
times $\mathbf{x}^{bad}$ computed	39
# $\varphi(\theta)$ calls	17
final θ	0.122845
final $\widetilde{\rho}_{TG}$	0.246
final # coarse dofs	1428
final OC	1.13612
final ρ_{asympt}	0.246326
final $\|\mathbf{x}_\mu\|_{A'}$	1.46065×10^{-15}

Parameters/Results	Value
θ when building from scratch	10^{-6}
initial θ for the adaptation	10^{-6}
χ	4
$\widetilde{\rho}_{TG}$ when building from scratch	> 0.999
# coarse dofs when building from scratch	338
OC when building from scratch	1.00766
# readaptation iterations	149
times $\mathbf{x}^{bad}$ computed	131
# $\varphi(\theta)$ calls	18
final θ	0.2306
final $\widetilde{\rho}_{TG}$	0.176
final # coarse dofs	1975
final OC	1.26025
final ρ_{asympt}	0.176201
final $\|\mathbf{x}_\mu\|_{A'}$	2.58888×10^{-18}

Table 4.13: Readaptation on the irregular mesh with 102400 elements and 51681 vertices using 300 AEs, $\nu = 6$, and $c = 6$. Algorithm 4.19 is applied.

Parameters/Results	Value
θ when building from scratch	0.01
initial θ for the adaptation	0.01
χ	1
$\widetilde{\rho}_{TG}$ when building from scratch	0.995
# coarse dofs when building from scratch	774
OC when building from scratch	1.04055
# readaptation iterations	22
times $\mathbf{x}^{bad}$ computed	18
# $\varphi(\theta)$ calls	4
final θ	0.148542
final $\widetilde{\rho}_{TG}$	0.163
final # coarse dofs	1745
final OC	1.20533
final ρ_{asympt}	0.167335
final $\|\mathbf{x}_\mu\|_{A'}$	6.14103×10^{-20}

Parameters/Results	Value
θ when building from scratch	0.01
initial θ for the adaptation	0.01
χ	4
$\widetilde{\rho}_{TG}$ when building from scratch	0.995
# coarse dofs when building from scratch	774
OC when building from scratch	1.04055
# readaptation iterations	49
times $\mathbf{x}^{bad}$ computed	45
# $\varphi(\theta)$ calls	4
final θ	0.148542
final $\widetilde{\rho}_{TG}$	0.249
final # coarse dofs	1536
final OC	1.15781
final ρ_{asympt}	0.24923
final $\|\mathbf{x}_\mu\|_{A'}$	4.41356×10^{-15}

Parameters/Results	Value
θ when building from scratch	10^{-6}
initial θ for the adaptation	10^{-6}
χ	1
$\widetilde{\rho}_{TG}$ when building from scratch	> 0.999
# coarse dofs when building from scratch	338
OC when building from scratch	1.00766
# readaptation iterations	56
times $\mathbf{x}^{bad}$ computed	38
# $\varphi(\theta)$ calls	18
final θ	0.2306
final $\widetilde{\rho}_{TG}$	0.151
final # coarse dofs	2150
final OC	1.30587
final ρ_{asympt}	0.158094
final $\|\mathbf{x}_\mu\|_{A'}$	3.40575×10^{-20}

Parameters/Results	Value
θ when building from scratch	10^{-6}
initial θ for the adaptation	10^{-6}
χ	4
$\widetilde{\rho}_{TG}$ when building from scratch	> 0.999
# coarse dofs when building from scratch	338
OC when building from scratch	1.00766
# readaptation iterations	118
times $\mathbf{x}^{bad}$ computed	100
# $\varphi(\theta)$ calls	18
final θ	0.2306
final $\widetilde{\rho}_{TG}$	0.159
final # coarse dofs	2104
final OC	1.29327
final ρ_{asympt}	0.158403
final $\|\mathbf{x}_\mu\|_{A'}$	1.95828×10^{-20}

Table 4.14: Readaptation on the irregular mesh with 102400 elements and 51681 vertices using 300 AEs, $\nu = 6$, and $c = 6$. Algorithm 4.19 is applied with the old vectors kept in the case when $m''_T = m_T$.

Parameters/Results	Value
θ when building from scratch	0.01
initial θ for the adaptation	0.01
$\widetilde{\rho}_{TG}$ when building from scratch	0.995
# coarse dofs when building from scratch	774
OC when building from scratch	1.04055
# readaptation iterations	7
times $\mathbf{x}^{bad}$ computed	7
# $\varphi(\theta)$ calls	6
final θ	0.474404
final $\widetilde{\rho}_{TG}$	0.129
final # coarse dofs	1715
final OC	1.19661
final ρ_{asympt}	0.135209
final $\|\mathbf{x}_\mu\|_{A'}$	6.15045×10^{-22}

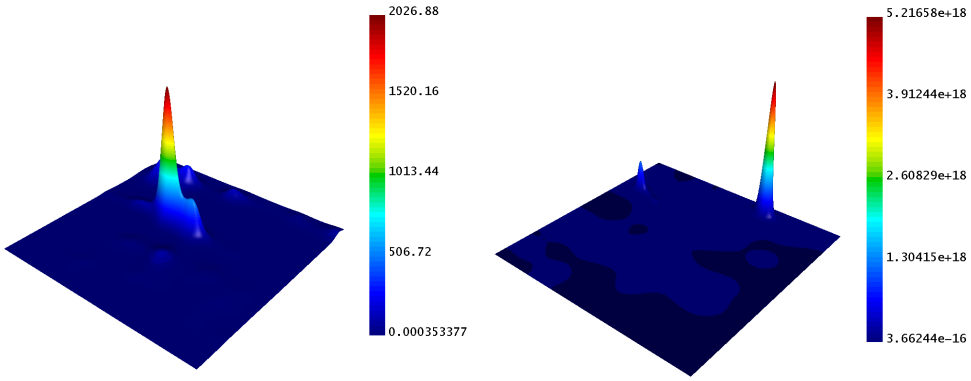
Parameters/Results	Value
θ when building from scratch	10^{-6}
initial θ for the adaptation	10^{-6}
$\widetilde{\rho}_{TG}$ when building from scratch	> 0.999
# coarse dofs when building from scratch	338
OC when building from scratch	1.00766
# readaptation iterations	20
times $\mathbf{x}^{bad}$ computed	20
# $\varphi(\theta)$ calls	19
final θ	0.408023
final $\widetilde{\rho}_{TG}$	0.229
final # coarse dofs	1505
final OC	1.14975
final ρ_{asympt}	0.232115
final $\|\mathbf{x}_\mu\|_{A'}$	3.74469×10^{-17}

Table 4.15: Readaptation on the irregular mesh with 102400 elements and 51681 vertices using 300 AEs, $\nu = 6$, and $c = 6$. Algorithm 4.24 is applied.

Parameters/Results	Value
θ when building from scratch	0.01
initial θ for the adaptation	1
$\widetilde{\rho}_{TG}$ when building from scratch	0.995
# coarse dofs when building from scratch	774
OC when building from scratch	1.04055
# readaptation iterations	3
times $\mathbf{x}^{bad}$ computed	3
final $\widetilde{\rho}_{TG}$	0.229
final # coarse dofs	1374
final OC	1.12525
final ρ_{asympt}	0.22222
final $\|\mathbf{x}_\mu\|_{A'}$	2.86497×10^{-17}

Parameters/Results	Value
θ when building from scratch	10^{-6}
initial θ for the adaptation	1
$\widetilde{\rho}_{TG}$ when building from scratch	> 0.999
# coarse dofs when building from scratch	338
OC when building from scratch	1.00766
# readaptation iterations	4
times $\mathbf{x}^{bad}$ computed	4
final $\widetilde{\rho}_{TG}$	0.185
final # coarse dofs	1238
final OC	1.10035
final ρ_{asympt}	0.186636
final $\|\mathbf{x}_\mu\|_{A'}$	1.98873×10^{-18}

Table 4.16: Readaptation on the irregular mesh with 102400 elements and 51681 vertices using 300 AEs, $\nu = 6$, and $c = 6$. Algorithm 4.24 is applied starting with $\theta = 1$.



(a) The first coefficient — k' **(b)** The second coefficient — k''

Figure 4.9: Two coefficient realizations. (They are piecewise constant on the mesh elements but are displayed as continuous for better appearance.)

Coefficient	# coarse dofs	OC	m	$\widetilde{\rho}_{TG}$
k'	765	1.06035	21	0.403
k''	463	1.02236	16	0.365
k', k''	765	1.06035	243	0.948
$k' \to k''$	400	1.01674	23	0.674

Table 4.17: Results for two coefficient realizations on the regular mesh with 131072 elements and 66049 vertices using 200 AEs and $\nu = 6$. The first two rows correspond to the case when for each coefficient a hierarchy is built from scratch using $\theta = 0.01$. The third row corresponds to the case when the unchanged (not adapted) hierarchy built for k' is applied for k''. The last row corresponds to the case when a single step of adaptation, using $\theta = 0.01$, is applied (using $\mu = 15$ without normalizing the approximations of $\mathbf{x}^{bad}$ on each step).

Parameters/Results	Value
$\mathrm{t}_{\mathrm{error}}$	10^{-14}
t_{CF}	0.4
initial θ for the adaptation	0.01
χ	1
# adaptation iterations	3
times $\mathbf{x}^{bad}$ computed	3
# $\varphi(\theta)$ calls	0
final θ	0.01
final $\widetilde{\rho}_{TG}$	0.394
final # coarse dofs	405
final OC	1.01721
final ρ_{asympt}	0.393988
final $\|\mathbf{x}_\mu\|_{A'}$	2.82649×10^{-10}

Parameters/Results	Value
$\mathrm{t}_{\mathrm{error}}$	10^{-14}
t_{CF}	0.3
initial θ for the adaptation	0.01
χ	1
# adaptation iterations	5
times $\mathbf{x}^{bad}$ computed	4
# $\varphi(\theta)$ calls	1
final θ	0.0199
final $\widetilde{\rho}_{TG}$	0.341
final # coarse dofs	432
final OC	1.01956
final ρ_{asympt}	0.340607
final $\|\mathbf{x}_\mu\|_{A'}$	3.30739×10^{-15}

Table 4.18: Results on the regular mesh with 131072 elements and 66049 vertices using 200 AEs and $\nu = 6$. Algorithm 4.19 is applied for adapting the hierarchy built from scratch for k' to k''.

Parameters/Results	Value
t_{error}	10^{-14}
t_{CF}	0.3
initial θ for the adaptation	0.01
χ	1
# readaptation iterations	7
times $\mathbf{x}^{bad}$ computed	5
# $\varphi(\theta)$ calls	2
final θ	0.039404
final $\widetilde{\rho}_{TG}$	0.201
final # coarse dofs	939
final OC	1.09097
final ρ_{asympt}	0.198408
final $\|\mathbf{x}_\mu\|_{A'}$	1.53652×10^{-14}

(a) Readapting the hierarchy for k'.

Parameters/Results	Value
t_{error}	10^{-14}
t_{CF}	0.3
initial θ for the adaptation	0.01
χ	1
# adaptation iterations	5
times $\mathbf{x}^{bad}$ computed	4
# $\varphi(\theta)$ calls	1
final θ	0.0199
final $\widetilde{\rho}_{TG}$	0.234
final # coarse dofs	429
final OC	1.01927
final ρ_{asympt}	0.225613
final $\|\mathbf{x}_\mu\|_{A'}$	3.10333×10^{-16}

(b) Adapting the final hierarchy for k' to k''.

Table 4.19: Results on the regular mesh with 131072 elements and 66049 vertices using 200 AEs and $\nu = 6$. Algorithm 4.19 is applied first to readapt the hierarchy for k' and then to adapt it to k''.

chy from scratch. Namely, in just a few adaptation iterations we obtain a smaller coarse space, compared to building the hierarchy from scratch, while the convergence rate is almost the same.

Multiple Coefficient Realizations

Since the described method is expected to be applied for many coefficient realizations, we show results when more than two coefficient realizations are used.

The results are for a small number of realizations that are in a way challenging. We generate forty realizations of the coefficient and we use the same set of realizations for the following two experiments. First, we apply the SA-AMGe method without adaptation by building the hierarchy from scratch for each coefficient (see the results in table 4.20). Next, we apply the adaptive SA-AMGe by producing the hierarchy from scratch for the first realization and then successively adapting the current realization to the next one (see the results in table 4.21). $\widetilde{\rho}_{TG}$ is computed by running a fixed number ($m = 50$) of iterations of the method for each coefficient realization.

We can see that we benefit from the adaptation versus building hierarchies from scratch. The source of the advantage is that the consecutive coefficient realizations are "close" to each other. Thus, as the results show, often adaptation is not necessary and the old hierarchy can be directly reused, and in case adaptation is necessary, a few iterations of the adaptation strategy are usually enough. The advantage of adaptation over building hierarchies from scratch is even larger when the agglomerated elements are large thus resulting in large

Parameters/Results	Value
θ	0.01
$\widetilde{\rho}_{TG}$ (minimal/maximal/mean)	0.361/0.773/0.583
# coarse dofs (minimal/maximal/mean)	466/805/556.875
OC (minimal/maximal/mean)	1.02222/1.06651/1.03272925

Table 4.20: Results for 40 coefficient realizations on the regular mesh with 131072 elements and 66049 vertices using 200 AEs and $\nu = 6$. The hierarchy is built from scratch for each coefficient realization. Arithmetic mean is used.

Parameters/Results	Value
$\mathrm{t}_{\mathrm{error}}$	10^{-14}
t_{CF}	0.5
initial θ for all realizations	0.01
χ	1
times (re)adaptation needed	19
# (re)adaptation iterations (minimal/maximal/mean/total)	1/11/4.263/81
times $\mathbf{x}^{bad}$ computed (minimal/maximal/mean/total)	2/10/4.421/105
$\widetilde{\rho}_{TG}$ (minimal/maximal/mean)	0.266/0.759/0.489
# coarse dofs (minimal/maximal/mean)	361/805/524.925
OC (minimal/maximal/mean)	1.01325/1.06651/1.02978675

Table 4.21: Results for 40 coefficient realizations on the regular mesh with 131072 elements and 66049 vertices using 200 AEs and $\nu = 6$. Algorithm 4.19 is applied. Arithmetic mean is used. For "# (re)adaptation iterations" and "times $\mathbf{x}^{bad}$ computed" minimal, maximal, and mean are computed among all calls to Algorithm 4.19 whenever adaptation was necessary.

local eigenvalue problems. Solving these problems is slow and so is building the hierarchy from scratch whereas the adaptation is still fast since we solve each eigenvalue problem in substantially smaller subspace. In general, adaptation is recommended when the consequent matrices of the systems being solved are in some sense "close" (as it turns out in the MCMC simulations), otherwise it may turn out that building hierarchies from scratch may be the preferable way.

4.6 Conclusions

At the end we draw some conclusions and outline directions for possible further development and improvements.

We have described an adaptive SA-AMGe method that can be used for solving many linear systems arising from finite element discretizations of second-order elliptic PDEs with random coefficients. The adaptation strategy that we have employed showed clearly its advantage over building two-level solvers from scratch when the PDE coefficient changes from one state k' to a next one, k''. The once build two-level hierarchy can be reused for subsequent realizations of the coefficient and, if necessary, it can be quickly adapted so that a desired rate of convergence is achieved. The method demonstrates a good potential and its applications are not limited to Monte Carlo simulation

of diffusion equations with stochastic coefficients. For example, the strategy can be applied for solving discretizations of a sequence of linear PDEs obtained by Picard linearization of (deterministic) nonlinear PDEs.

A natural candidate for further development is working out and implementing a multilevel extension of the two-level method described in this thesis. Although the method allows for aggressive coarsening that results in small coarse spaces, a multilevel extension of the method will lead to even better efficiency due to the better complexity of the resulting solvers.

There is also a potential in constructing improved Monte Carlo simulations utilizing the hierarchy of coarse spaces, if information between the hierarchies for the solver and the actual Monte Carlo method is "exchanged" and reused.

Finally, there is a potential of improvements in our software implementation. We have pointed out on some weaknesses that need further attention, such as the issue with potentially empty aggregates (see Section 4.1.3, "Empty Aggregates"). A natural and desirable major future development is implementing a parallel version of the method (both, two- and multilevel versions).

References

[1] `http://code.google.com/p/mfem/`. 2, 3

[2] `http://code.google.com/p/glvis/`. 2

[3] `http://glaros.dtc.umn.edu/gkhome/views/metis/`. 2, 17

[4] `http://www.caam.rice.edu/software/ARPACK/`. 2

[5] O. Axelsson. *Iterative Solution Methods.* Cambridge University Press, 1996. 11

[6] D. Braess. *Finite elements: theory, fast solvers, and applications in elasticity theory.* Cambridge University Press, 2007. 6

[7] 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, 2001. 10.1137/S1064827598344303. 27

[8] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. Adaptive smoothed aggregation (αSA) multigrid. *SIAM Rev.*, 47(2):317–346, 2005. 10.1137/050626272. 54

[9] Marian Brezina, Petr Vaněk, and Panayot S. Vassilevski. An improved convergence analysis of smoothed aggregation algebraic multigrid. *Numerical Linear Algebra with Applications*, 2011. 10.1002/nla.775. 30, 31, 44

[10] Marian Brezina and Panayot S. Vassilevski. *Smoothed Aggregation Spectral Element Agglomeration AMG: SA-ρAMGe.* Lawrence Livermore National Laboratory Technical Report LLNL-PROC-490083. June 29, 2011. 28, 29, 30, 31, 35, 38, 53

[11] W Briggs and S McCormick. Introduction. In Stephen F McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in applied mathematics*, pages 1–30. SIAM, 1987. 23

[12] W.L. Briggs, V.E. Henson, and S.F. McCormick. *A multigrid tutorial.* Society for Industrial and Applied Mathematics, 2000. 11, 19, 23

[13] T. Chartier, R. D. Falgout, V. E. Henson, J. Jones, T. Manteuffel, S. Mccormick, J. Ruge, and P. S. Vassilevski. Spectral AMGe (ρAMGe). *SIAM J. Sci. Comput.*, 25(1):1–26, 2003. 27

[14] Timothy Chartier, Robert Falgout, Van Emden Henson, Jim Jones, Tom Manteuffel, John Ruge, Steve McCormick, and Panayot Vassilevski. Spectral element agglomerate AMGe. In Olof B. Widlund and David E. Keyes, editors, *Domain Decomposition Methods in Science and Engineering XVI*, volume 55 of *Lecture Notes in Computational Science and Engineering*, pages 513–521. Springer Berlin Heidelberg, 2007. 10.1007/978-3-540-34469-8_64. 27

[15] K. Cliffe, M. Giles, R. Scheichl, and A. Teckentrup. Multilevel monte carlo methods and applications to elliptic PDEs with random coefficients. *Computing and Visualization in Science*, 14:3–15, 2011. 10.1007/s00791-011-0160-x. 1

[16] Y. Efendiev and T.Y. Hou. *Multiscale finite element methods: theory and applications.* Surveys and tutorials in the applied mathematical sciences. Springer, 2009. 2

[17] R.D. Falgout. An introduction to algebraic multigrid computing. *Computing in Science & Engineering*, 8(6):24–33, nov.-dec. 2006. 10.1109/MCSE.2006.105. 32

[18] G.H. Golub and C.F.V. Loan. *Matrix computations.* Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, 1996. 14

[19] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method.* Dover books on mathematics. Dover Publications, 2009. 5, 6, 10

[20] Richard M. Karp. Reducibility among combinatorial problems. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Gio-

vanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 219–241. Springer Berlin Heidelberg, 2010. 10.1007/978-3-540-68279-0_8. 41

[21] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1998. 16

[22] J W Ruge and K Stüben. Algebraic multigrid. In Stephen F McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in applied mathematics*, pages 73–130. SIAM, 1987. 24, 25

[23] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003. 7, 11

[24] Y. Shapira. *Solving PDEs in C++: numerical methods in a unified object-oriented approach*. Computational science and engineering. Society for Industrial and Applied Mathematics, 2006. 7

[25] Steven S. Skiena. Set and string problems. In *The Algorithm Design Manual*, pages 620–656. Springer London, 2008. 10.1007/978-1-84800-070-4_18. 41

[26] K. Stüben. *Algebraic multigrid (AMG): an introduction with applications; updated version of GMD report No 53, March 1999*. GMD-Report. GMD-Forschungszentrum Informationstechnik, 1999. 24, 25, 26

[27] Klaus Stüben and Ulrich Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, volume 960 of *Lecture Notes in Mathematics*, pages 1–176. Springer Berlin / Heidelberg, 1982. 10.1007/BFb0069928. 23

[28] U. Trottenberg, C.W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001. 23, 24, 25, 26

[29] P. Vaněk, J. Mandel, and M. Brezina. *Algebraic multigrid on unstructured meshes*. UCD/CCM report. University of Colorado at Denver, Center for Computational Mathematics, 1994. 25, 26, 27

[30] Petr Vaněk, Marian Brezina, and Jan Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numerische Mathematik*, 88:559–579, 2001. 10.1007/s211-001-8015-y. 27

[31] Panayot S. Vassilevski. *Lecture Notes on Multigrid Methods*. Lawrence Livermore National Laboratory Technical Report LLNL-TR-439511. July 1, 2010. 12, 22, 26, 27, 30, 32, 36

[32] P.S. Vassilevski. *Multilevel Block Factorization Preconditioners: matrix-based analysis and algorithms for solving finite element equations*. Springer, 2008. 6, 12, 22, 24, 26, 27, 29, 30, 32, 36