KRYLOV SUBSPACE METHODS FOR TOPOLOGY OPTIMIZATION ON ADAPTIVE MESHES

BY

SHUN WANG

B.Eng., Tsinghua University, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

Abstract

Topology optimization is a powerful tool for global and multiscale design of structures, microstructures, and materials. The computational bottleneck of topology optimization is the solution of a large number of extremely ill-conditioned linear systems arising in the finite element analysis. Adaptive mesh refinement (AMR) is one efficient way to reduce the computational cost. We propose a new AMR scheme for topology optimization that results in more robust and efficient solutions.

For large sparse symmetric linear systems arising in topology optimization, Krylov subspace methods are required. The convergence rate of a Krylov subspace method for a symmetric linear system depends on the spectrum of the system matrix. We address the ill-conditioning in the linear systems in three ways, namely rescaling, recycling, and preconditioning.

First, we show that a proper rescaling of the linear systems reduces the huge condition numbers that typically occur in topology optimization to roughly those arising for a problem with homogeneous density.

Second, the changes in the linear system from one optimization step to the next are relatively small. Therefore, recycling a subspace of the Krylov subspace and using it to solve the next system can improve the convergence rate significantly. We propose a minimum residual method with recycling (RMINRES) that preserves the short-term recurrence and reduces the cost of recycle space selection by exploiting the symmetry. Numerical results show that this method significantly reduces the total number of iterations over all linear systems and the overall computational cost (compared with the MINRES method which

is optimal for a single symmetric system). We also investigate the recycling method for adaptive meshes.

Third, we propose a multilevel sparse approximate inverse (MSPAI) preconditioner for adaptive mesh refinement. It significantly improves the conditioning of the linear systems by approximating the global modes with multilevel techniques, while remaining cheap to update and apply, especially when the mesh changes. For convection-diffusion problems, it achieves a level-independent convergence rate. We then make a few changes in the MSPAI preconditioner for topology optimization problems. With these extensions, the MSPAI preconditioner achieves a nearly level-independent convergence rate. Although for small to moderate size problems the incomplete Cholesky preconditioner is faster in time, the multilevel sparse approximate inverse preconditioner will be faster for (sufficiently) large problems. This is important as we are more interested in scalable methods.

In memory of my mom

Acknowledgments

I would like to express my gratitude to all those who have given me support and help to complete this thesis. Without them, this work would not have been a possibility.

First and foremost, I am deeply indebted to my advisor, Eric de Sturler. His wide knowledge, insightful advice and many good ideas have been of great value to me. He has given me enormous support and guidance, not only on my research but also on my professional development. It has been a wonderful experience working with him. I would also like to thank my co-advisor, Glaucio Paulino, for his guidance and supervision along my work on topology optimization. I have benefited greatly from the discussions with him and his group.

I also thank my other committee members, Stephen Bond and Michael Heath for the constructive comments and discussion on my research. In addition, I would thank Stephen for kindly being my *de jure* advisor since Eric moved to Blacksburg, and for providing invaluable comments and suggestions on my thesis. I would thank Michael for being the leader of the Numerical Analysis (NA) group, for his support, and for leading the CS591 seminar for all our fellow students.

In addition to my committee members, I thank David Alber and Kapil Ahuja for reviewing the manuscript and giving me invaluable feedback.

I have benefited greatly from the interaction with my fellow students from both the NA group and Professor Paulino's Computational Mechanics group. I would like to thank, especially, David Alber, Zhen Cheng, Bill Cochran, Eric Cyr, Rebecca Hartman-Baker, Chau Le, Vanessa Lopez, Hanna Neradt, Kyoungsoo Park, Mike Parks, Chris Siefert, Bin Shen,

Fernando Stump, Cam Talischi and Zhengyu Zhang. To my friends, I offer my sincere thanks too. In addition to the people above, Jia Guo, Chao Huang, Kevina Lam, Vincent Li, Chunke Su, Bin Yu, Jing Zhang, and many others have been great companions during my life in Urbana-Champaign.

I have used a number of numerical libraries to develop the new methods and algorithms described in this thesis. This would have been much harder without the significant help from Hong Zhang and Mat Knepley from the `PETSc` development team in Argonne National Laboratory, and from Roy Stogner, John Peterson, and Benjamin Kirk from the `libMesh` development team in the University of Texas at Austin. I offer special thanks to them.

I am also grateful for the financial support from Eric de Sturler through the IBEAM project funded by the National Aeronautics and Space Administration (grant number: NASA NCC 5-615), and from Eric de Sturler and David Ceperley through the Material Computation Center funded by the National Science Foundation (grant number: DMR-0325939).

Finally, I am greatly indebted to my wonderful parents, Wang Fuqiang and Pan Jianjun. Their love, dedication, support and encouragement have accompanied me throughout my life. No words could describe my infinite thankfulness to them.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

AMR	Adaptive Mesh Refinement [14, 13]
BiCGStab	Stabilized Bi-Conjugate Gradient method [71]
CAMD	Continuous Approximation of Material Distribution [46]
CG	Conjugate Gradient method [37]
GCRO	Generalized Conjugate Residual method with inner Orthogonalization [28]
GCRODR	Generalized Conjugate Residual method with inner Orthogonalization and Deflated Restarting [51]
GCROT	Generalized Conjugate Residual method with inner Orthogonalization and optimal Truncation [29]
GMRES	Generalized Minimum Residual method [59]
FE, FEA	Finite Element, Finite Element Analysis [3]
FGM	Functionally Graded Material [52, 53]
IC, ILU	Incomplete Cholesky/LU decomposition [47, 33]
MINRES	Minimum Residual method [50]
MMA	Method of Moving Asymptotes [69]
MSPAI	Multilevel Sparse Approximate Inverse preconditioner [76]
OC	Optimality Criteria [9, 7]
PDE	Partial Differential Equation [48]
RMINRES	Recycling Minimum Residual method [77]
SIMP	Solid Isotropic Material with Penalization [6]
SPAI	Sparse Approximate Inverse preconditioner [35, 24, 22]

List of Notations

$\boldsymbol{A}, \boldsymbol{K}, \boldsymbol{\Lambda}$	Bold upper-case Latin and Greek letters represent matrices or tensors
$\boldsymbol{x}, \boldsymbol{u}$	Bold lower-case Latin letters represent vectors or vector functions
k, s, θ, λ	Lower-case Latin and Greek letters represent scalars or scalar functions
$\mathcal{K}$	Krylov subspace
$\mathcal{P}_m$	Set of polynomials of degree m
$\|\cdot\|$	2-norm
$\|\cdot\|_F$	Frobenius norm
$\Lambda(\boldsymbol{A})$	Set of eigenvalues of matrix $\boldsymbol{A}$
$\kappa(\cdot), \mathrm{cond}(\cdot)$	condition number of a matrix
$\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{R}^n$, $\mathbb{R}^{m\times n}$	Set of real numbers, set of positive real numbers, set of real n vectors, set of real $m \times n$ matrices
$\mathrm{range}\,(\boldsymbol{U})$	Range of matrix $\boldsymbol{U}$, subspace expanded by the columns of $\boldsymbol{U}$
$\boldsymbol{A}_{:,j}$	The jth column of matrix $\boldsymbol{A}$

Chapter 1

Introduction

Structural design optimization includes size, shape and topology optimizations. In size optimization we specify a truss structure, and the goal is to find the optimal size for each member as demonstrated in Figure 1.1(a). This is a finite-dimensional optimization problem. In shape optimization we specify the topology of the design, and the goal is to find the optimal shape of the holes and materials as demonstrated in Figure 1.1(b). This is an infinite-dimensional optimization. Both size and shape optimization require some level of prior knowledge about the nature of the design problem to determine a proper solution space that includes the optimal solution.

On the other hand, in topology optimization we specify only an allowed physical domain, and the goal is to find the optimal material distribution in this physical domain. That is, the optimization algorithm determines which parts of the space should be material and which parts should be empty. It is an infinite-dimensional optimization in the continuum setting. All possible structures, or material distributions to be more precise, within the specified domain are considered. To define a topology optimization problem, we specify only the configuration of the problem, i.e., the design domain, the boundary conditions and the loading. We do not need any prior knowledge about the final design. Therefore, in general, topology optimization is more powerful than size and shape optimization.

Topology optimization has been used for various structural design problems. For example, Airbus designed the wing box ribs of the A380 airplane using topology optimization [43] and saved 500 kilogram weight (40% of the original wing ribs) in each wing. Altair Engineering has designed bus and motorcycle frames using topology

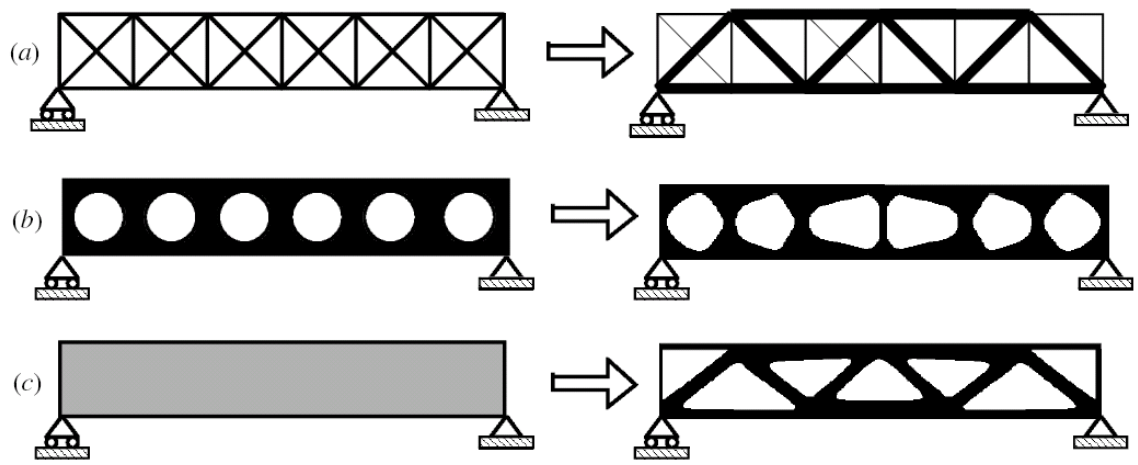


Figure 1.1: Three types of structural optimization. (a) size optimization; (b) shape optimization; (c) topology optimization. The initial setups are shown on the left and the optimal solutions are shown on the right. (Figure 1 in [62] by courtesy of Prof. Ole Sigmund)

optimization [1]. Furthermore, because of its flexibility, topology optimization has been used for design problems other than structural design. For example, it has been used to design Microelectromechanical Systems (MEMS) [57]. It has also been used to design microstructural materials with extreme material properties, e.g., materials with negative Poisson's ratio [44] and materials with negative thermal expansion coefficient [65].

To make topology optimization a truly effective tool in the design of large structures and complex materials, we must use large three-dimensional models. In the literature, most work on topology optimization for continuum structures has emphasized developing new formulations and applications, designing suitable elements, and studying existence and uniqueness issues. The computational aspect of large-scale topology optimization, specifically the high cost of solving many large and ill-conditioned linear systems, has not received enough attention. Therefore, it is the main focus of this thesis.

A topology optimization algorithm lead to an evolving process of the density distribution. In the process, the structure evolves as void and solid appear. In order to achieve accurate and smooth results, a fine mesh representation is required for solid regions, especially at

the surface of solid regions. However it is not necessary to use fine mesh representation for void regions. Therefore, adaptive mesh refinement (AMR) [14, 13] is useful technique for reducing the computational cost while achieving the same level of accuracy. Limited amount of research has been conducted on applying AMR to topology optimization [67, 25]. In many cases, the mesh refinement strategies proposed [67, 25] lead to suboptimal designs. In this thesis, we propose a more robust and efficient mesh adaptation strategy for topology optimization. It provides more freedom for the design to move to its global optimum, and it allows for mesh derefinement which further reduces the computational cost.

The main computational cost in topology optimization comes from the finite element analysis, which involves solving a long sequence of linear systems of the form:

$$\boldsymbol{K}^{(i)}\boldsymbol{u}^{(i)} = \boldsymbol{f}. \tag{1.1}$$

Here, $\boldsymbol{K}^{(i)}$ is the stiffness matrix, which is a function of the design variables at the ith optimization step, $\boldsymbol{f}$ is the load vector, and $\boldsymbol{u}^{(i)}$ is the displacement vector. Currently, direct solvers are most commonly used because of the very large condition numbers arising in topology optimization. However, direct solvers cannot effectively handle large 3D problems, because they scale poorly in terms of the storage requirements and the computational cost. This makes them prohibitively expensive for large systems. On the other hand, iterative solvers have low storage requirements and the computational cost per iteration is often linear in the problem size. Therefore, as long as the convergence rate is reasonably fast, iterative solvers can solve very large problems efficiently.

Iterative solvers offer additional advantages compared with direct solvers. First, we do not need to solve the finite element problem very accurately in the early phase of the topology optimization process. Second, iterative solvers are relatively easy to parallelize [17, 73, 40], which is important for very large problems. Third, iterative solvers can use solutions from previous systems as starting guesses, leading to smaller initial residuals and reduced solving

time. Last, for a sequence of linear systems that change slowly, we can reduce the total number of iterations by recycling subspaces of earlier search spaces [51, 39].

In topology optimization, the change in the design variables becomes small after the first few optimization steps. Therefore, the change in the system matrix $\boldsymbol{K}$ from one optimization step to the next is also small, and the Krylov subspace recycling methods introduced in [51] are likely to be effective. Some topology optimization problems lead to a nonlinear system in each optimization step [31]. If we use a Newton or quasi-Newton method for the nonlinear system, the (approximate) Jacobians often change sufficiently slowly so that we can further exploit recycling, for example, see [39]. In most structural problems, the matrices are symmetric but not necessarily positive definite. For example, in vibration problems symmetric indefinite matrices arise [63]. For such matrices, MINRES (minimum residual method) [50] is the method of choice. Therefore, we study Krylov subspace recycling with the MINRES method. Figure 1.2 demonstrates the basic idea of the recycling MINRES method. We solve the first system by building the Krylov subspace $\mathcal{K}^{(1)}$. We obtain the most important subspace $\mathcal{R}^{(1)}$ out of $\mathcal{K}^{(1)}$ for recycling. For the following systems, we build the Krylov subspace $\mathcal{K}^{(j)}$ such that it is orthogonal to the recycle space $\mathcal{R}^{(j-1)}$ from the previous system. We obtain the solution of the linear system and an updated recycle space $\mathcal{R}^{(j)}$ from the union of $\mathcal{K}^{(j)}$ and $\mathcal{R}^{(j-1)}$. If we judiciously choose the recycle space $\mathcal{R}^{(j)}$ such that it contains the most "important" information, we would obtain the solution of the linear system in fewer iterations. We study the selection of the recycle space in this thesis. We make recycling more efficient by exploiting symmetry and short-term recurrences.

For symmetric systems, the ratio between the largest and smallest eigenvalues governs the worst-case upper bound on the convergence rate of a Krylov subspace method. In topology optimization, this ratio can be extremely large because of the wide range of magnitudes of the element densities. This ill-conditioning also affects the accuracy in the solution of the linear system. We address this ill-conditioning by using a diagonal rescaling before applying a more general preconditioner like an incomplete factorization. Our analysis and

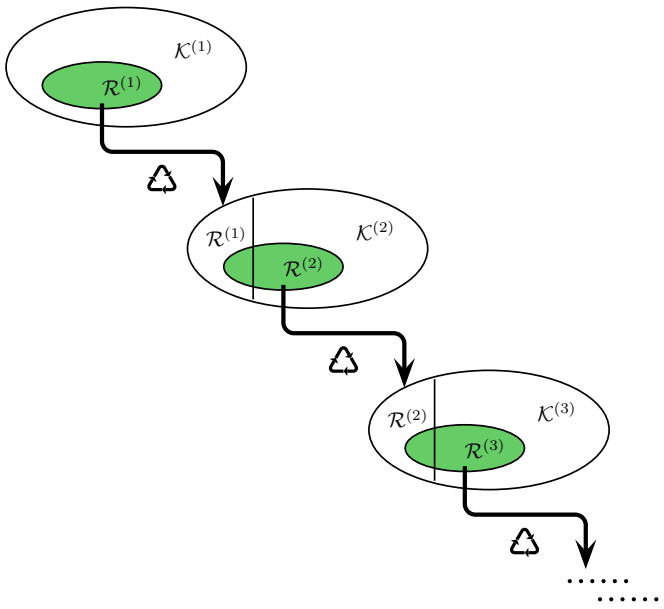


Figure 1.2: Demonstration of recycling MINRES. $\mathcal{K}$ denotes the Krylov subspace, and $\mathcal{R}$ denotes the recycle space.

demonstrations with 1D problems show that a simple diagonal rescaling reduces the huge condition numbers arising from topology optimization problems to a magnitude of roughly the condition numbers arising from problems with homogeneous densities.

In addition to the rescaling technique, a more general preconditioner is required to further improve the convergence rate. For large-scale simulations on adaptive meshes, a desired preconditioner should be easy to adapt to the changes in the mesh. For this reason, we are interested in sparse approximate inverse (SPAI) preconditioners [35, 11, 22, 24, 23, 12]. First, the columns of a SPAI preconditioner are independent and explicitly stored. Therefore, the construction and the matrix-vector multiplications of a SPAI preconditioner are easy for parallelization. Second, the weak data interdependencies allow us to limit the update of a SPAI preconditioner locally to places where the mesh changes or the coefficients of the system change drastically. However, the local support property of SPAI preconditioners makes them ineffective to capture global modes. In many applications, we need a very large sparsity pattern for SPAI to get a good convergence rate. Such a large sparsity pattern

leads to higher computational cost for constructing and applying the preconditioner. This problem can be remedied by multilevel techniques at a low cost. We propose a new method of using multilevel techniques in SPAI. Exploiting the hierarchical structure of the adaptive mesh, our method significantly improves the convergence rate of an iterative solver and yet remains relatively cheap.

The remainder of this thesis is organized as follows. In Chapter 2, we give a brief introduction to topology optimization. Then, we propose a new mesh adaptation strategy for topology optimization that gives more robust and efficient solutions compared to the strategies in the existing literature [67, 25]. In Chapter 3, we analyze the ill-conditioning introduced by the wide range of element densities in topology optimization, and show that a simple diagonal rescaling can largely cure this artificial ill-conditioning. In Chapter 4, we discuss subspace recycling for Krylov subspace methods. In particular, we introduce a variant of the MINRES method that includes subspace recycling for symmetric systems arising in topology optimization. We also discuss the adaptation of our recycling MINRES method for AMR. In Chapter 5, we introduce a new multilevel sparse approximate inverse preconditioner for AMR. We first consider convection-diffusion problems, and then move on to its application and adaptation to topology optimization problems. In Chapter 6, we provide an overview of the major contributions in this thesis and suggest some future work.

Readers interested in Krylov subspace methods are referred to the book *Iterative methods for sparse linear systems* by Yousef Saad [58], and the book *Iterative Krylov Methods for Large Linear Systems* by Henk A. van der Vorst [72]. Readers interested in topology optimization and its applications are referred to the book *Topology Optimization: Theory, Methods and Applications* by Martin P. Bendsøe and Ole Sigmund [9], which covers a wide range of topics and lists a large number of reference papers.

Chapter 2

Topology Optimization with Adaptive Mesh Refinement

In this chapter, we first introduce some background on topology optimization, including the modeling and solution scheme. As the solution of a topology optimization problem evolves, holes and solid regions appear. It is often efficient to use adaptive mesh refinement (AMR). We discuss some related work in the literature and propose a more robust and efficient mesh adaptation strategy.

2.1 Topology Optimization

Topology optimization is a relatively new branch of structural optimization. Unlike shape and size optimization, topology optimization constructs a structure by material points, similar to the process of constructing a picture from pixels. In the continuum setting, the feasible design space includes all configurations (i.e., shape, size and connectivity) in a given domain. Therefore, it is generally more powerful than traditional shape and size optimization.

A desired result of topology optimization consists of either material points or void points. However, it is mathematically difficult to work with integer (discrete) variables, so this condition is typically relaxed. By allowing intermediate material density between 0 (void) and 1 (solid), functions become continuous and differentiable. To steer the solution back to discrete 0/1 values, we penalize intermediate densities by making them uneconomical in terms of stiffness per volume.

At an early stage, the homogenization method [7] was used to derive the stiffness for

intermediate densities based on certain configurations of microstructures. The solutions for the problems discussed here, however, are not supposed to contain microstructures. Thus, the stiffness for intermediate material densities based on the homogenization approach only serves as a means of penalization. Later on, the Solid Isotropic Material with Penalization (SIMP) approach was proposed [8] as a simpler way to interpolate the stiffness of intermediate density. The SIMP material model was originally thought of as a fictitious material model, but later on it was proved that there exist materials with the stiffness derived by the SIMP model if the penalization parameter is greater than or equal to three.

In the following sections, we briefly review the mathematical definition of topology optimization problems in the continuum and finite element discretization settings. In the finite element setting, we discuss two approaches, namely the element-based approach and the node-based approach with Continuous Approximation of Material Distribution (CAMD) [46]. The CAMD approach is further extended to model functionally graded materials (FGMs) with the so-called FGM-SIMP model [56].

2.1.1 Topology Optimization in Continuum Setting

In the continuum setting, the design variable we optimize is the material density field in a specified design domain. In a desired solution, the material density at any point should be either 0 or 1. For the relaxed problem, the material density can take intermediate values between 0 and 1. The problem is posed as the minimization of an objective function, such as the mean compliance, subject to a volume constraint. The compliance is a function of the displacement, which is implicitly defined by the density-dependent equilibrium equation. The problem statement for minimization of compliance subject to volume constraint is as follows:

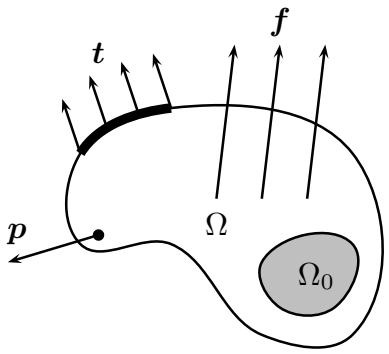


Figure 2.1: Topology optimization problem configuration. Ω is the design domain; Ω_0 is the domain with fixed boundary conditions; $\boldsymbol{f}$ is a body force; $\boldsymbol{t}$ is a surface traction; $\boldsymbol{p}$ is a point force.

$$\min_{\rho\in[0,1]} a(\boldsymbol{u},\boldsymbol{u}) \tag{2.1}$$

$$\text{s.t.} \quad \begin{cases} \boldsymbol{u} \in \{\boldsymbol{w} \in W_2^1(\Omega) \mid \boldsymbol{w} = \boldsymbol{u}_0 \text{ on } \Omega_0\}, \\ a(\boldsymbol{u},\boldsymbol{v}) = l(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \{\boldsymbol{w} \in W_2^1(\Omega) \mid \boldsymbol{w} = 0 \text{ on } \Omega_0\}, \\ \int_\Omega \rho d\Omega \le V_0, \end{cases}$$

where Ω is the whole domain in which we solve the design problem, Ω_0 is the domain with fixed boundary conditions, $W_1^2(\Omega)$ is the Sobolev function space defined on domain Ω [3, p. 115], V_0 is the volume constraint, and

$$a(\boldsymbol{u},\boldsymbol{v}) = \int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{v}) \cdot (\boldsymbol{C}(\rho)\boldsymbol{\varepsilon}(\boldsymbol{u}))d\Omega, \tag{2.2}$$

$$l(\boldsymbol{u}) = \int_\Omega \boldsymbol{f}\cdot\boldsymbol{u}d\Omega + \int_{\Gamma_t} \boldsymbol{t}\cdot\boldsymbol{u}d\Gamma + \boldsymbol{p}\cdot\boldsymbol{u}. \tag{2.3}$$

In mechanics, $\frac{1}{2}a(\boldsymbol{u},\boldsymbol{u})$ is the internal strain energy, $a(\boldsymbol{u},\boldsymbol{u})$ is the compliance of the structure, and $l(\boldsymbol{u})$ is the external energy from a body force $\boldsymbol{f}$, a surface traction $\boldsymbol{t}$, and point forces $\boldsymbol{p}$ (see Figure 2.1).

An analytical solution in the continuum setting is generally not possible, and the finite

element method is often used to discretize both the displacement field and material density field. However, the interpolation for the displacements and densities can be independent [54].

2.1.2 Finite Element Discretization

Element-based Approach

To solve the problem numerically, we discretize the problem using finite elements. We use a lower order interpolation for the density field than for the displacement field. The most common approach is to use (bi-,tri-)linear interpolation for the displacement field and piecewise constant density throughout each element. The compliance minimization problem after finite element discretization can be defined as

$$\min_{\rho_e \in [\rho_o, 1], \forall e} \boldsymbol{f}^T \boldsymbol{u} \tag{2.4}$$
$$\text{s.t.} \quad \begin{cases} \boldsymbol{K}(\rho)\boldsymbol{u} = \boldsymbol{f} & \text{for } \boldsymbol{x} \in \Omega \setminus \Omega_0, \\ \boldsymbol{u} = \boldsymbol{u}_0 & \text{for } \boldsymbol{x} \in \Omega_0, \\ \sum_e \rho_e V_e \leq V_0, \end{cases}$$

where the stiffness matrix $\boldsymbol{K}$ is a function of the density vector (discretized density field), and V_e is the volume of element e. To avoid singularity of the stiffness matrix, we enforce a small positive lower bound ρ_o on the element density, typically 10^{-3}.

We use the SIMP method to make the undesirable intermediate densities between 0 (or ρ_o) and 1 unfavorable. In this case, the elasticity tensor is defined as a function of the element density

$$\boldsymbol{E}_e = \rho_e^p \boldsymbol{E}_0, \tag{2.5}$$

where p is the penalization parameter. With a parameter $p > 1$, an intermediate density for an element is made unfavorable due to its relatively low contribution to the stiffness

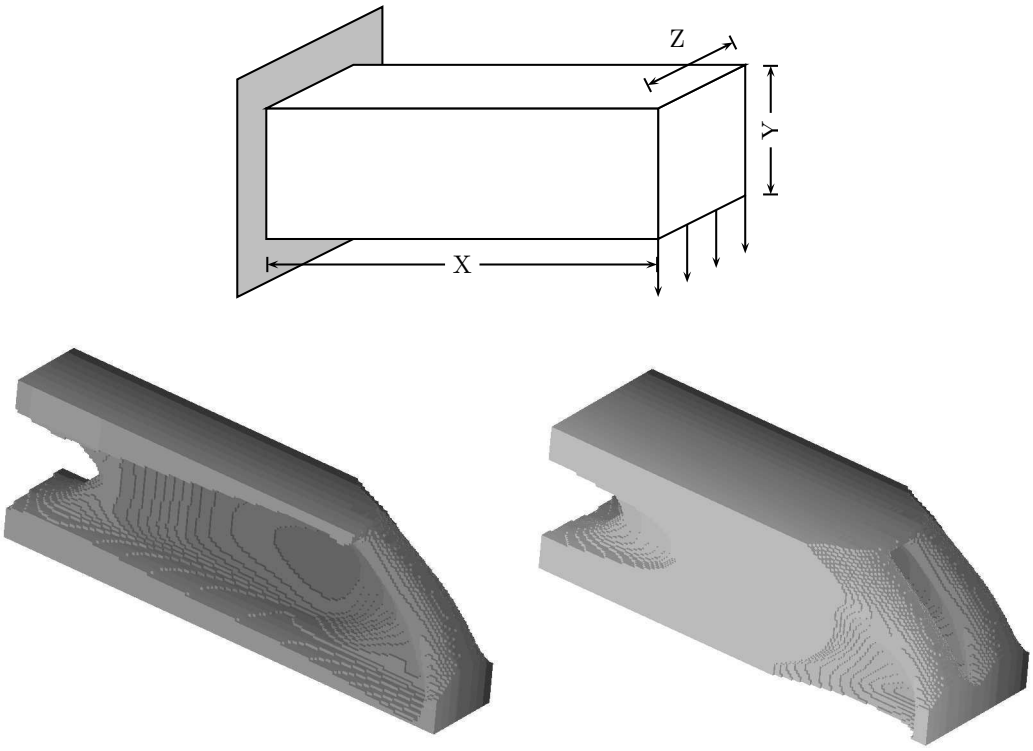


Figure 2.2: 3D beam design on a $3 \times 1 \times 1$ domain with the element-based approach.

compared to its material cost. A common choice for p is three, which results in intermediate material properties that satisfy the Hashin–Strikman bound for any real composite materials.

In Figure 2.2, we give the result of a 3D beam problem using the element-based approach. Exploiting the symmetry in the z direction, we solve the problem on only half of the domain with a $180 \times 60 \times 30$ trilinear element (B8) mesh discretization, which leads to 324,000 design variables and over 1 million degrees of freedom in the finite element simulation.

Node-based Approach

For smoother and more realistic results, we can model the design variable by representing the material density at mesh nodes and interpolating the density inside each element. This node-based approach is called Continuous Approximation of Material Distribution (CAMD) [46]. The mesh for discretization of the density field usually coincides with that for discretization

of the displacement field. However, the two meshes are not necessarily the same [54]. Higher order polynomial shape functions can be used to improve the accuracy of the displacement field. However, only first order shape functions are used for the density field because higher order shape functions may result in negative material density.

With this discretization, the compliance minimization problem is defined as

$$\min_{\rho_i \in [\rho_0, 1], \forall i} \boldsymbol{f}^T \boldsymbol{u} \tag{2.6}$$
$$\text{s.t.} \quad \begin{cases} \boldsymbol{K}(\rho)\boldsymbol{u} = \boldsymbol{f} & \text{for } \boldsymbol{x} \in \Omega \setminus \Omega_0, \\ \boldsymbol{u} = \boldsymbol{u}_0 & \text{for } \boldsymbol{x} \in \Omega_0, \\ \int_\Omega \rho dV \leq V_0, \end{cases}$$

The elasticity tensor is still a function of the density field, but now it varies inside each element.

$$\boldsymbol{E}(\boldsymbol{x}) = (\rho(\boldsymbol{x}))^p \boldsymbol{E}_0, \tag{2.7}$$
$$\rho(\boldsymbol{x}) = \sum_i N_i(\boldsymbol{x}) \rho_i. \tag{2.8}$$

where ρ_i represents the density at node i. Therefore, we need numerical integration, such as Gaussian quadrature, to compute the stiffness matrix and the volume cost.

We solve the same 3D beam problem as in Figure 2.2 with the CAMD approach. With a coarser mesh $84 \times 28 \times 14$, we achieve a design of similar resolution, as shown in Figure 2.3.

Functionally Graded Material Domain

If we design with functionally graded material, the material properties vary in space [52, 53]. In particular, the elasticity tensor is a variable with respect to the position as well as the density [56, 30]:

$$\boldsymbol{E}(\boldsymbol{x}) = \rho(\boldsymbol{x}) \boldsymbol{E}_0(\boldsymbol{x}). \tag{2.9}$$

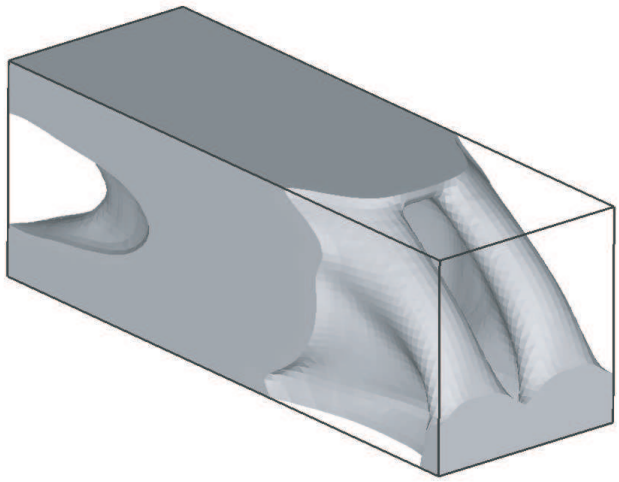


Figure 2.3: 3D beam design with the CAMD approach.

For a simple exponentially graded material in 3D, the elasticity tensor using the FGM-SIMP model is given by

$$\boldsymbol{E}(\boldsymbol{x}) = \rho(\boldsymbol{x})\boldsymbol{E}_0 e^{\alpha x+\beta y+\gamma z}. \tag{2.10}$$

To capture the gradient of FGM properties inside each element, we usually use the CAMD approach. The parameters $1/\alpha$, $1/\beta$ and $1/\gamma$ denote the length scales of nonhomogeneity in the x, y and z directions, respectively.

As an example, we solve a 3D cantilever beam problem for both homogeneous material and functionally graded material. Both solutions are obtained using the CAMD approach on a $210 \times 70 \times 70$ B8 mesh without exploiting symmetry. The results are shown in Figure 2.4. While the design for the homogeneous material shows symmetry in both y and z directions, the design for the functionally graded material is symmetric only in the z direction and has more material at the bottom of the domain where the material is softer.

2.2 Finite Element Solution Scheme

The general scheme for topology optimization is illustrated in Figure 2.5. First, we set up the geometry, the finite element (FE) mesh, the loading and boundary conditions, and we

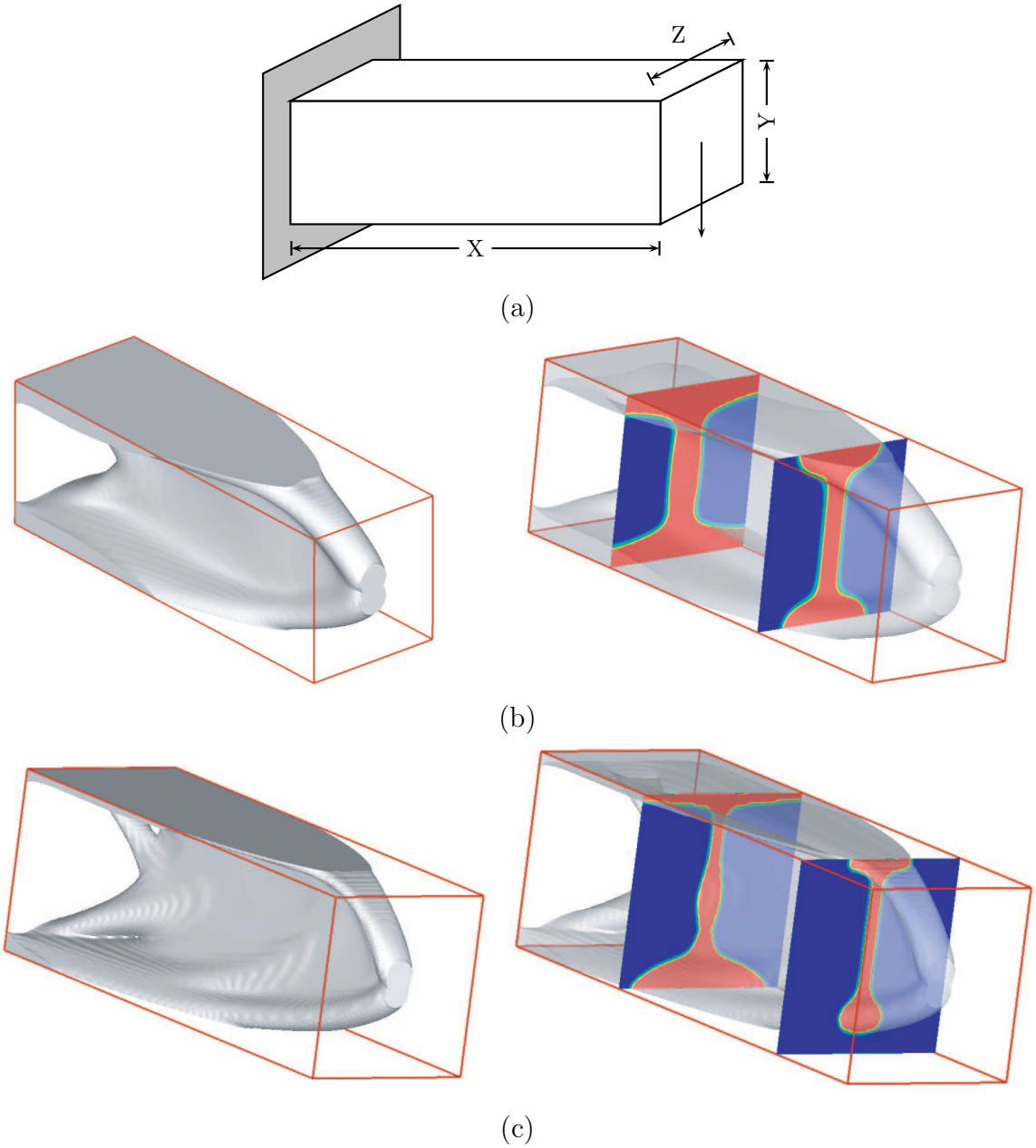


Figure 2.4: A 3D cantilever beam on a $3 \times 1 \times 1$ domain. (a) problem configuration; (b) design result for homogeneous material; (c) design result for exponentially graded material (the material is only graded in the y direction and is softer on the bottom than on the top).

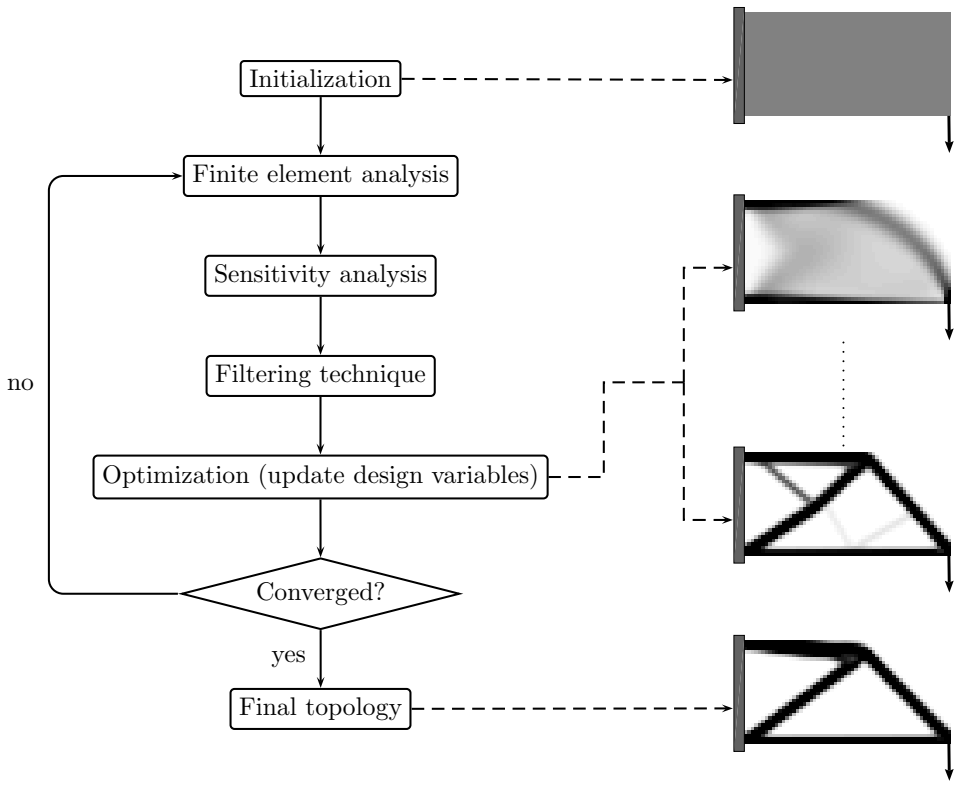


Figure 2.5: The general scheme of topology optimization.

initialize the density distribution ρ. Then, we start the optimization loop. In the loop, we assemble and solve the equilibrium equations $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$ in (2.4) and (2.6) using the FE discretization and a linear solver. Next, in the sensitivity analysis, we compute the derivatives of the objective function with respective to the design variables, e.g., $\partial c/\partial \rho_e$ for (2.4) or $\partial c/\partial \rho_i$ for (2.6). Thereafter, we can apply an optional low-pass filter to remedy the checkerboard problem [60, 61, 64], which can be also addressed by an alternative minimum length scale approach [36]. In the next step, we compute an update of the design variable. There are various optimization algorithms applicable to topology optimization. For instance, Optimality Criteria (OC) is a simple approach based on a set of intuitive criteria [9, 7], while the Method of Moving Asymptotes (MMA) is a mathematical programming algorithm that is more robust and well-established [69]. After updating the design variables using a chosen optimization algorithm, we check the convergence of the design.

In general, the optimization takes many steps to converge. The computationally most expensive part of this loop is the finite element analysis (FEA), which must be carried out many times. In this thesis, we propose faster solvers and preconditioners for solving the sequence of evolving linear systems arising from the FEA.

2.3 Adaptive Mesh Refinement for Topology Optimization

In the field of topology optimization, problems are solved most commonly on fixed uniform meshes with a large number of elements in order to achieve accurate design results. However, as holes and solid regions appear in the design, it is more efficient to represent the holes with fewer large elements and the solid regions, especially the material surface, with more fine elements. Since the shape and position of the holes and solids initially unknown, the most economical mesh representation for the design is unknown *a priori*. Therefore, adaptive mesh refinement (AMR) is very suitable for topology optimization. The purpose of AMR for topology optimization is to get the design that would be obtained on a uniformly fine mesh, but at a much lower computational cost by reducing the total number of elements and having fine elements only where necessary.

Limited amount of research has been conducted into applying AMR to topology optimization [67, 25]. In [67], Stainko chooses to refine only the elements at the interface between solid and void. In [25], Costa and Alves choose to refine the void elements on the solid/void interface and all the material elements. However, their approaches share some similarities. First, both approaches use only refinement, but no derefinement. Second, both approaches solve the design problems on a fixed mesh until convergence before carrying out mesh refinement. When the mesh is refined to the specified finest level, it is unchanged for the remainder of the optimization. This works well in terms of refining the design. However, for some design problems, a converged solution on a coarser mesh may not be the

optimal solution on a finer mesh. Mesh refinement based only on the coarser level solution may erroneously confine the solution on the finest mesh to a smooth version of the coarse level solution. Therefore, the approaches proposed in [67, 25] sometimes lead to suboptimal designs, and thus a more robust refinement strategy is required. Furthermore, for designs with thin structures, the starting coarsest mesh must be fine enough to give a reasonable result. In many cases, derefinement would save more computational effort when holes appear. Here, we propose a more economical and more robust mesh adaptation scheme for topology optimization.

2.3.1 Mesh Adaptation for Topology Optimization

We represent the adaptive mesh in a hierarchical fashion. Following the refinement of an element, the new finer elements are stored as the children of the original element in a tree structure. This makes mesh refinement easier and derefinement possible. We only need to add and remove tree nodes for refinement and derefinement, respectively.

The convergence criterion for topology optimization is often that the maximum change in the design variables for the last optimization step is smaller than a certain tolerance, which we usually set as 0.01. We adapt the mesh when the following conditions are satisfied:

1. the relative change in the compliance is smaller than a threshold;

2. a given number of optimization steps have occurred since the last mesh (de)refinement.

The first condition is satisfied when the solution is almost trapped in a local minimum, which may be caused by the undesirable mesh. Then, we adapt the mesh to allow the design to change further. The second condition is to avoid mesh (de)refinement from happening too frequently and limits the cost of mesh adaptation. The mesh is refined once at least every ten optimization steps. Using these conditions, we can start with a fairly coarse mesh, and we may carry out mesh (de)refinement before the design converges on any mesh if necessary.

To avoid confinement of the design solution by the mesh, in addition to multiple mesh (de)refinements on the same level, we also construct a layer of fine elements outside the material surface in order to provide some freedom for the material to be redistributed locally. We adapt our mesh according to the following procedure.

1. Mark all the elements for refinement or derefinement based on the following criteria:

   - If element e is solid, i.e., $\rho_e \in [\rho_s, 1]$ with a chosen density threshold ρ_s, we mark it for refinement.
   - If element e is void, i.e., $\rho_e \in [\rho_\epsilon, \rho_s]$, but there are solid elements within a given distance r_{amr}, we mark element e for refinement; otherwise, we mark element e for derefinement (see Figure 2.6).

2. Check the mesh compatibility and make the following adjustments through two sweeps of all elements:

   - In the first sweep, we unmark the elements that are marked for derefinement, if they have sibling element not marked for derefinement.
   - In the second sweep, we adjust the mark where level two or higher edge incompatibility happens, and allow only level one incompatibility, see Figure 2.7 and refer to Section 2.3.2.

The above refinement criteria result in a layer of fine elements on the void side of the solid/void interface, which allows the material to be redistributed locally. If a material boundary has moved to the fine/coarse element interface, another mesh (de)refinement process takes place and makes another layer of fine elements around the current material surface to allow further local redistribution of the material. Where the material have been removed, some of the fine elements are derefined in order to keep the optimization efficient.

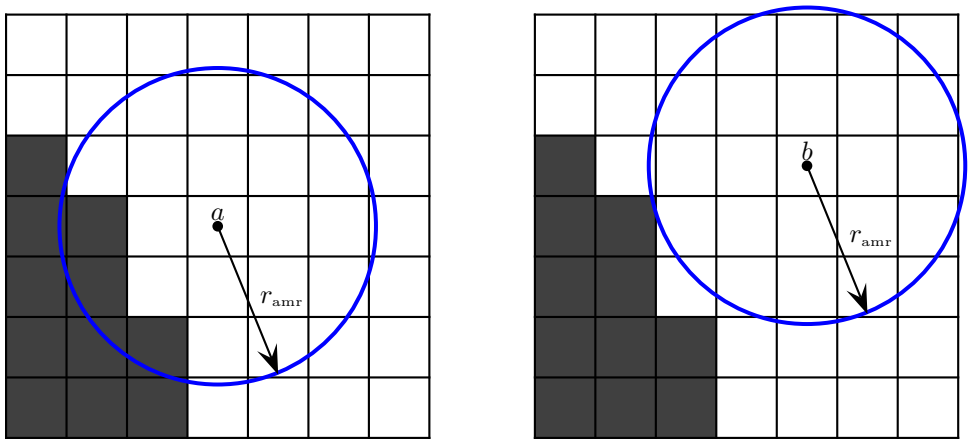


Figure 2.6: Refinement criteria for void element. Element a is marked for refinement for it has solid elements within distance r_{amr}; element b is marked for derefinement.

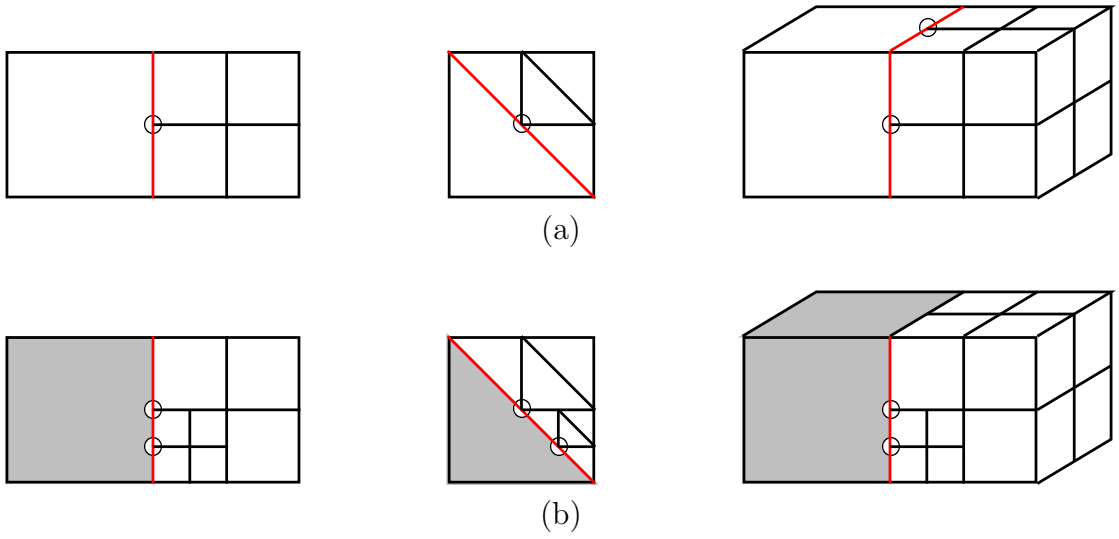


Figure 2.7: Mesh incompatibility with examples of quad, triangle and hex elements. (a) level one edge incompatibility marked by red edges and circled nodes; (b) level two edge incompatibility marked by red edges and circled nodes. We allow level one edge incompatibility (see Section 2.3.2), but we avoid level two or higher incompatibility by refining the gray coarse elements, or not derefining their children elements if these gray elements result from a potential derefinement.

2.3.2 Implementation with `libMesh`

For the implementation of adaptive mesh refinement, we use the `libMesh` library [41] developed at the University of Texas at Austin and the Technische Universität Hamburg-Harburg. The `libMesh` library consists of a C++ framework for numerical simulations of partial differential equations on serial and parallel platforms. It supports 1D, 2D and 3D finite element and finite volume simulations on adaptive meshes. It uses `PETSc` [5, 4] for the solution of linear systems on both serial and parallel platforms.

We have developed 2D and 3D topology optimization algorithms on top of `libMesh`. Currently, we use element-based design variables, the SIMP method for material interpolation [8], the OC method for optimization [9, 7], and Sigmund's filter technique [60, 61, 64]. There is a small modification we make in Sigmund's filter for a nonuniform mesh. The filter takes a distance and density weighted average on the sensitivities of all elements in a certain radius as

$$\widehat{\frac{\partial c}{\partial \rho_e}} = \frac{1}{\rho_e \sum_d H_{de}} \sum_d \rho_d H_{de} \frac{\partial c}{\partial \rho_d}, \tag{2.11}$$

where $\partial c/\partial \rho_e$ is the sensitivity of the compliance with respect to the density of element e, and H_{de} is a distance weight defined as

$$H_{de} = \max\{r_{\min} - \mathrm{dist}(d, e), 0\}. \tag{2.12}$$

The parameter $r_{\min}$ is a given radius for the filter, and $\mathrm{dist}(d, e)$ is the distance between the centers of elements d and e. For a nonuniform mesh, we consider different element sizes by adding element volume as part of the weight in the filter [67] as

$$\widehat{\frac{\partial c}{\partial \rho_e}} = \frac{1}{\rho_e \sum_d H_{de} V_d} \sum_d \rho_d H_{de} V_d \frac{\partial c}{\partial \rho_d}. \tag{2.13}$$

The filter radius $r_{\min}$ is often a physical size independent on the mesh representation. Notice that the filter will be effectively deactivated if its size is smaller than that of the smallest

element, i.e., no element has any neighbors within distance $r_{\min}$. This plays a role in the mesh refinement strategy because we have to start with a relatively fine mesh for the filter to work properly.

Due to the hierarchical data structure of `libMesh`, we cannot avoid mesh incompatibility completely. However, we do avoid level two or higher edge incompatibility, see Figure 2.7. The remaining level one mesh incompatibility results in hanging nodes, e.g., the circled nodes in Figure 2.7. We handle those by enforcing constraints in our stiffness matrix. We can divide the degrees of freedom (DOFs) into two groups. Group one consists of all the unconstrained DOFs, and group two consists of the constrained DOFs on the hanging nodes. The constrained DOFs can be computed by linear interpolation from unconstrained DOFs. If we define vector $\tilde{\boldsymbol{u}}$ on the unconstrained DOFs, then

$$\boldsymbol{u} = \begin{pmatrix} \tilde{\boldsymbol{u}} \\ \boldsymbol{P}\tilde{\boldsymbol{u}} \end{pmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{P} & \boldsymbol{0} \end{bmatrix} \begin{pmatrix} \tilde{\boldsymbol{u}} \\ \boldsymbol{0} \end{pmatrix} \tag{2.14}$$

is the mapping of $\tilde{\boldsymbol{u}}$ on all the DOFs, where $\boldsymbol{P}$ is the interpolation matrix. We solve the original linear system $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$ on all the DOFs by solving a constrained system

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{P}^T \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \boldsymbol{K} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{P} & \boldsymbol{0} \end{bmatrix} \hat{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{P}^T \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \boldsymbol{f}. \tag{2.15}$$

Since `libMesh` does not drop the constrained DOFs in the linear system, the constrained system in (2.15) is singular when there is any hanging node. This singularity is fine for Krylov subspace methods as long as the right hand side is consistent, but it may cause problems for preconditioners. To avoid this singularity, we set the diagonal entries of the

system that correspond to the constrained DOFs as 1 and solve

$$\left(\begin{bmatrix} \boldsymbol{I} & \boldsymbol{P}^T \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \boldsymbol{K} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{P} & \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \right) \hat{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{P}^T \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \boldsymbol{f}. \tag{2.16}$$

In the end, we recover the constrained DOFs by applying the interpolation matrix as

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{P} & \boldsymbol{0} \end{bmatrix} \hat{\boldsymbol{u}}. \tag{2.17}$$

2.3.3 Results and Discussion

In order to demonstrate the improvement our new AMR scheme makes, we solve a design problem on both a fixed uniform mesh and an adaptive mesh, and with both our AMR scheme and the approaches in [67, 25].

One approach to measuring the difference between two designs is to take the 1-norm of the difference between the two density vectors, if they are defined on the same uniform mesh. If the two designs are represented on two different meshes, uniform or not, it is more appropriate to measure their relative difference with

$$D(\rho^{(1)}, \rho^{(2)}) = \frac{\int_\Omega |\rho^{(1)} - \rho^{(2)}| d\Omega}{\int_\Omega \rho^{(1)} d\Omega}. \tag{2.18}$$

This can be done by refining the meshes for both designs to a same fine mesh, and then evaluating the 1-norm of the difference between the designs.

As our first experiment, we solve the 2D beam problem shown in Figure 2.8(a) on a uniformly fine mesh. Figure 2.8(b) shows an intermediate result, and Figure 2.8(c) the converged design. The truss at the lower-right corner has risen up noticeably from the intermediate result to the final one.

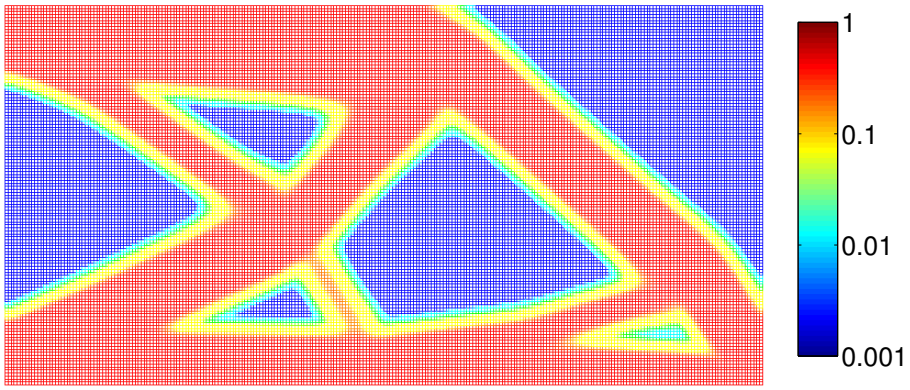
Next, we solve the same problem following the strategy mentioned in [67, 25]. We start

with a relatively coarse mesh (64×32), and obtain the converge optimization solution shown in Figure 2.9(a). Then, we refine the mesh according to this coarse level result and get the refined mesh shown in Figure 2.9(b). Next, we solve the optimization problem on this locally refined mesh until convergence, and then refine the mesh and solve again. Finally, we obtain the result on the finest mesh shown in Figure 2.9(c). The same truss at the lower-right corner has been stuck at the finest elements, making this solution different than the solution we obtain on the uniformly fine mesh. This confinement is artificially imposed by the undesired mesh obtained based only on the coarse level solution. Moreover, we have to start with a relatively fine mesh, such as the one in Figure 2.9(a), because a coarser initial mesh would lead to an unreasonable solution due to the deactivation of the filter. In this case, mesh adaptation with only refinement and no derefinement leaves fairly fine elements at the void regions on the final mesh, which could otherwise be derefined for further improvement on efficiency.

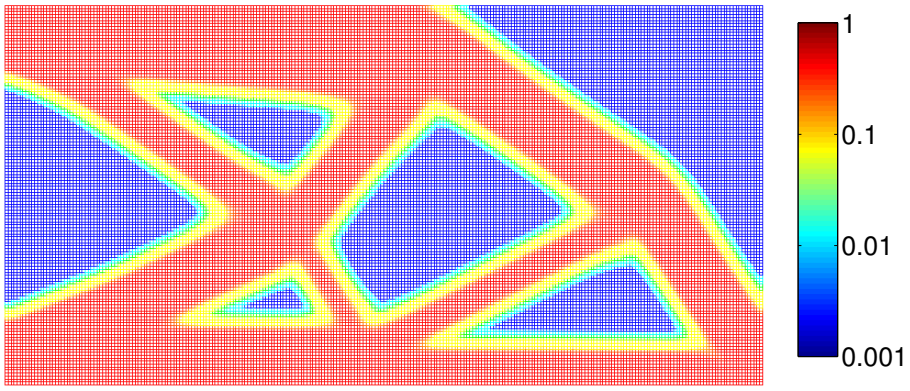
Now, we solve the same problem, starting with the same coarse mesh as shown in Figure 2.9(a), but following the strategy proposed in Section 2.3.1. We allow multiple mesh adaptations on any level and a layer of fine elements on the void side of the solid/void interface. This leads to the final result shown in Figure 2.10(c), with two intermediate results shown in Figures 2.10(a) and (b). The mesh changes from the intermediate results to the final one when the material distribution changes. The element size on the finest parts of the meshes in Figure 2.10 is the same as that of the uniform mesh shown in Figure 2.8. Compared to the solution shown in Figure 2.9, the solution obtained with our mesh adaptation strategy is much closer to the solution obtained on the uniform mesh. Under the measurement given in (2.18), the relative difference between the designs in Figure 2.9(c) and Figure 2.8(b) is 0.196, while the relative difference between the designs in Figure 2.10(c) and Figure 2.8(b) is only 1.68×10^{-3}. Furthermore, with derefinement, we have coarser elements at the void regions compared to the final mesh shown in Figure 2.9(c).

The next problem is a three-dimensional cantilever beam as shown in Figure 2.11 with the

(a)

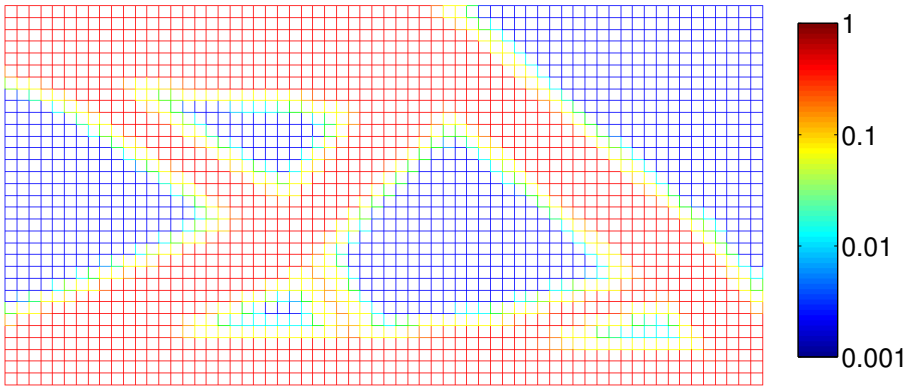


(b)

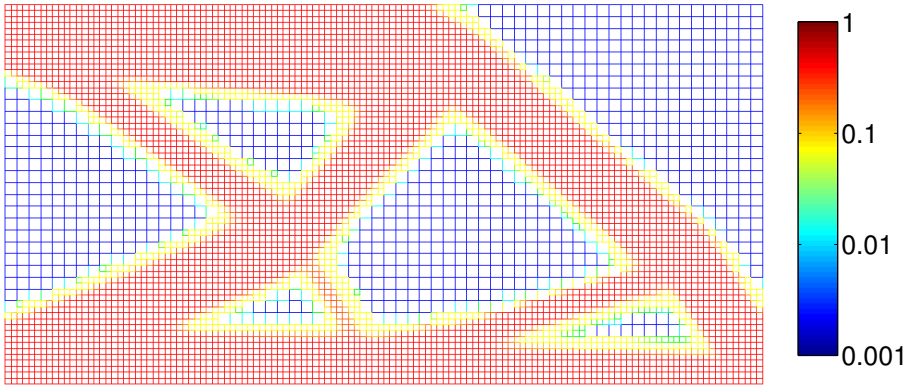


(c)

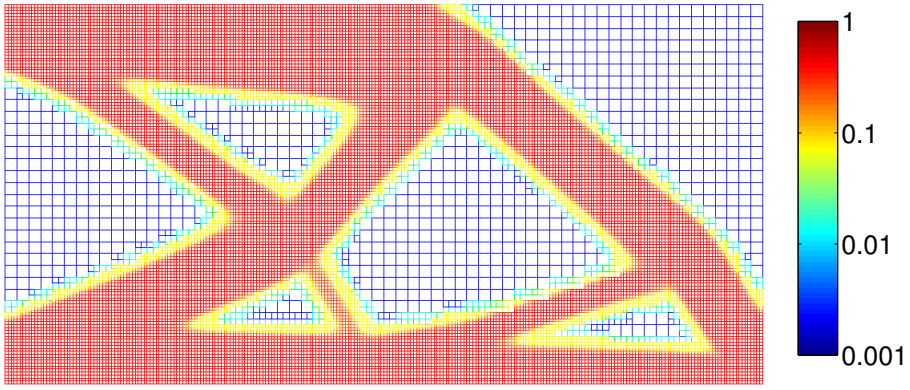
Figure 2.8: Topology optimization on a 256×128 *uniform mesh*. (a) problem configuration (volume constraint V_0 is 50% of the domain volume); (b) an intermediate result; (c) final converged result.



(a)

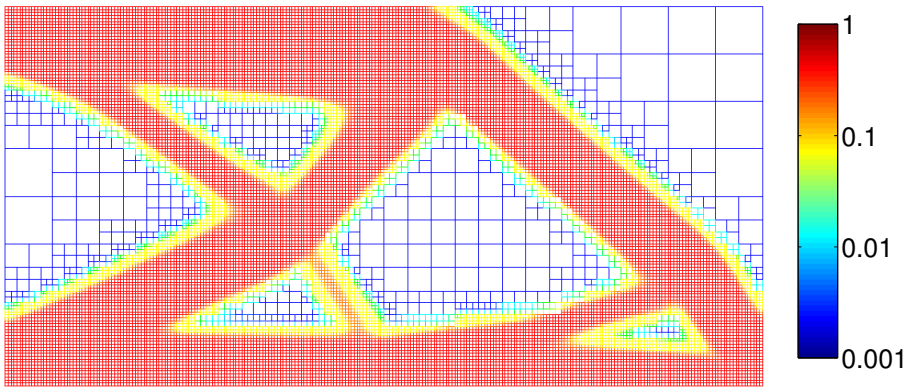


(b)

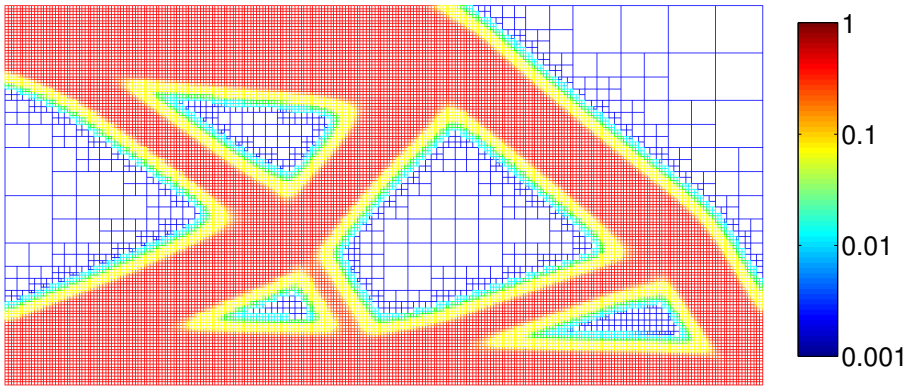


(c)

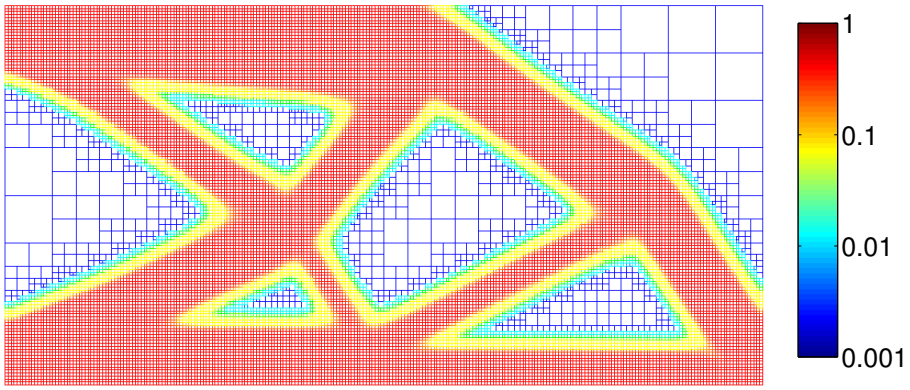
Figure 2.9: Topology optimization on *an adaptive mesh with single mesh refinement on each level*. (a) converged result on the coarsest mesh with 2048 elements; (b) converged result on the intermediate mesh with 5675 elements; (c) converged result on the final mesh with 20216 elements.



(a)



(b)



(c)

Figure 2.10: Topology optimization on *an adaptive mesh with multiple dynamic mesh refinement and derefinement on each level*. (a)–(b) intermediate results; (c) final converged result on a nonuniform mesh with 23099 elements, whose finest resolution is the same as the uniform mesh in Figure 2.8.

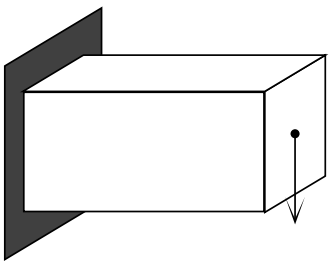
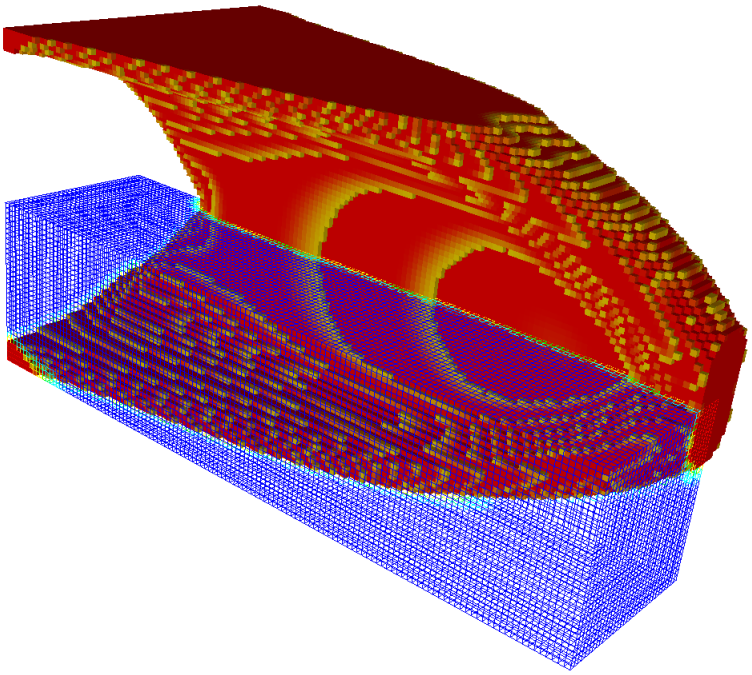


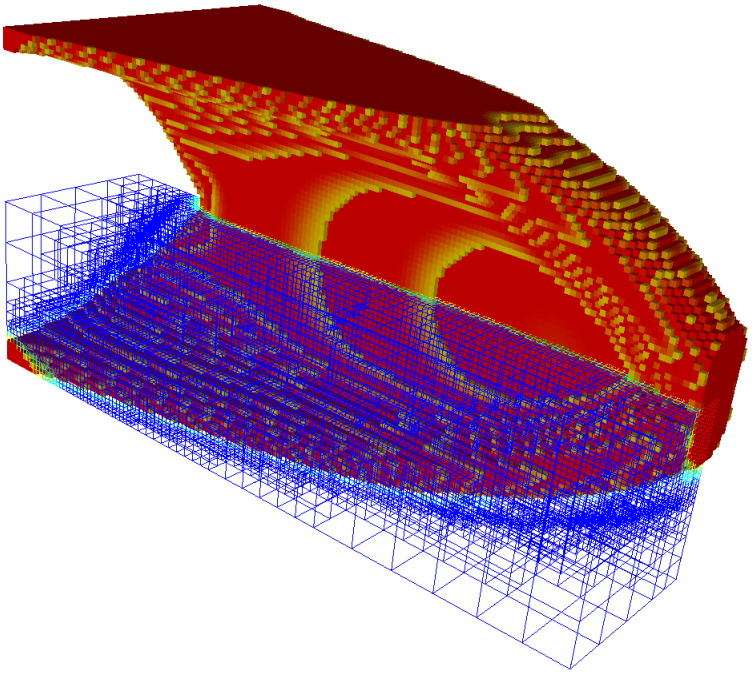
Figure 2.11: 3D cantilever beam example with domain scale 2:1:1.

volume constraint as 25%. Exploiting the symmetry, we use only a quarter of the domain. We solve it on a fixed uniform mesh with $128 \times 32 \times 32$ B8 elements, and on an adaptive mesh starting with $64 \times 16 \times 16$ B8 elements and following our (de)refinement strategy. The final results are shown in Figure 2.12. The relative difference between these two designs is only 9.09×10^{-4}. We use the RMINRES solver proposed in Chapter 4 to solve the FE systems. The system on the adaptive mesh is less than half the size of the one on the fixed uniform mesh, and is even smaller at the start of the optimization process. The number of RMINRES iterations for the adaptive mesh is slightly smaller than for the fixed uniform mesh, because the FE systems on the adaptive mesh are less ill-conditioned. Therefore, the adaptive mesh reduces the solving time significantly (see Figure 2.13).

Now, we present a more complex 3D example shown in Figure 2.14. In a cross-shaped domain, we want to find the optimal design subject to the fixed boundary on the front and back ends at the bottom and two loads on the left and right side at the bottom. The maximum volume allowed is 20% of the domain volume. We solve this problem both on a uniform mesh and on an adaptive mesh. The results are shown in Figures 2.15 and 2.16, respectively. The uniform mesh consists of 40960 B8 elements, while the final adaptive mesh consists of only 19736 B8 elements. Moreover, the optimization requires over 200 steps to converge on the uniform mesh, but only 106 steps on the adaptive mesh. The adaptive mesh refinement saves about 70% computational time in total. Nonetheless, the relative L_1 difference between the two design is 0.0258.



(a)



(b)

Figure 2.12: Final solutions of the 3D cantilever problem in Figure 2.11 obtained on only quarter of the domain indicated by the mesh. (a) final solution on a fixed uniform mesh with $128 \times 32 \times 32$ elements; (c) final solution on an adaptive mesh with 57173 elements, whose finest resolution is the same as that of the fixed uniform mesh.

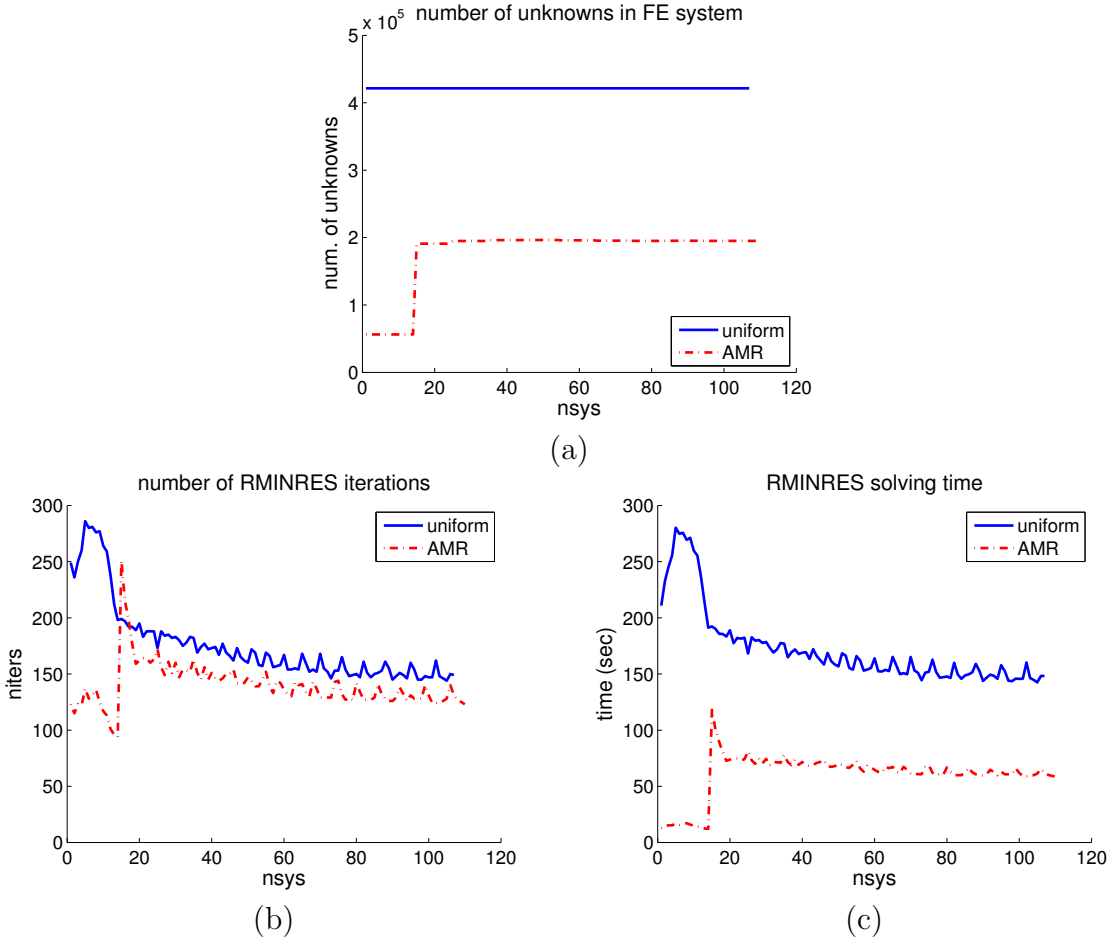


Figure 2.13: Comparison between the solutions on the fixed uniform mesh and the adaptive mesh. (a) number of unknowns in the FE systems; (b) number of RMINRES(200,10) iterations (see Chapter 4) for each step of topology optimization; (c) solving time of the FE systems with RMINRES(200,10).

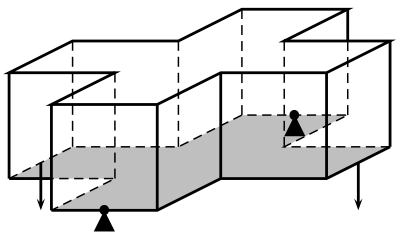


Figure 2.14: A 3D compliance minimization problem in a cross-shaped domain with the front and back ends at the bottom fixed and the left and right ends at the bottom pulled down. The volume constraint V_0 is 20% of the domain volume.

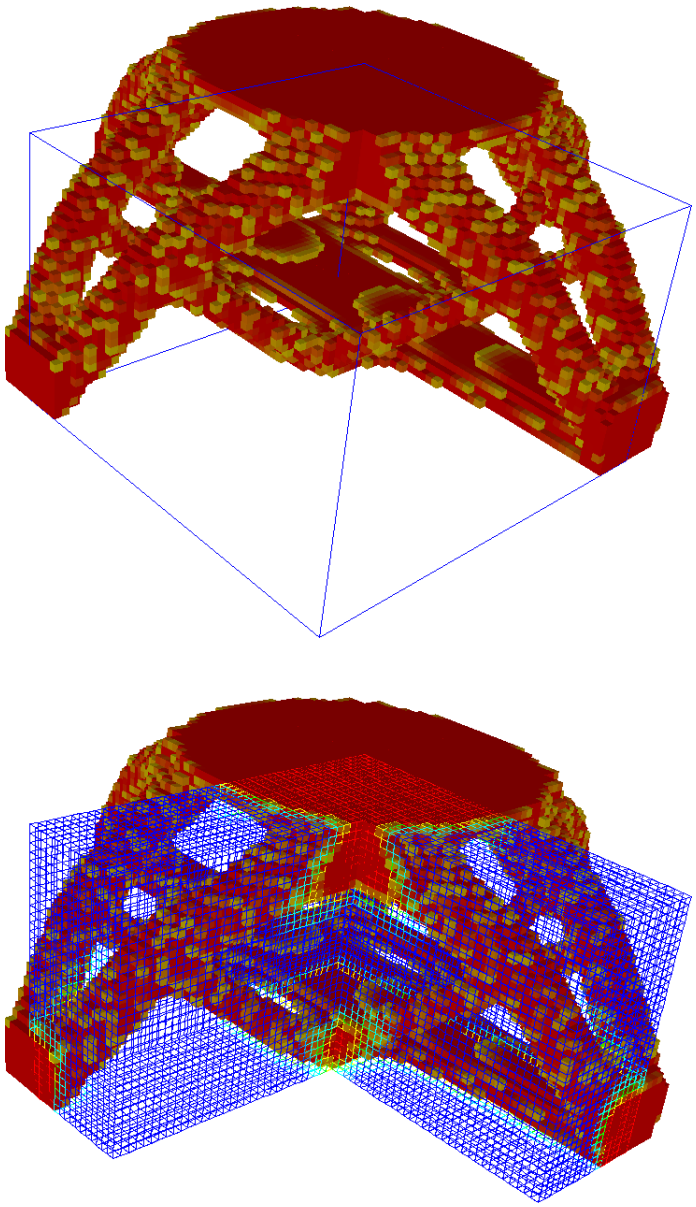


Figure 2.15: The optimization solution of the problem shown in Figure 2.14 on a finite element mesh with 40960 B8 elements of uniform size.

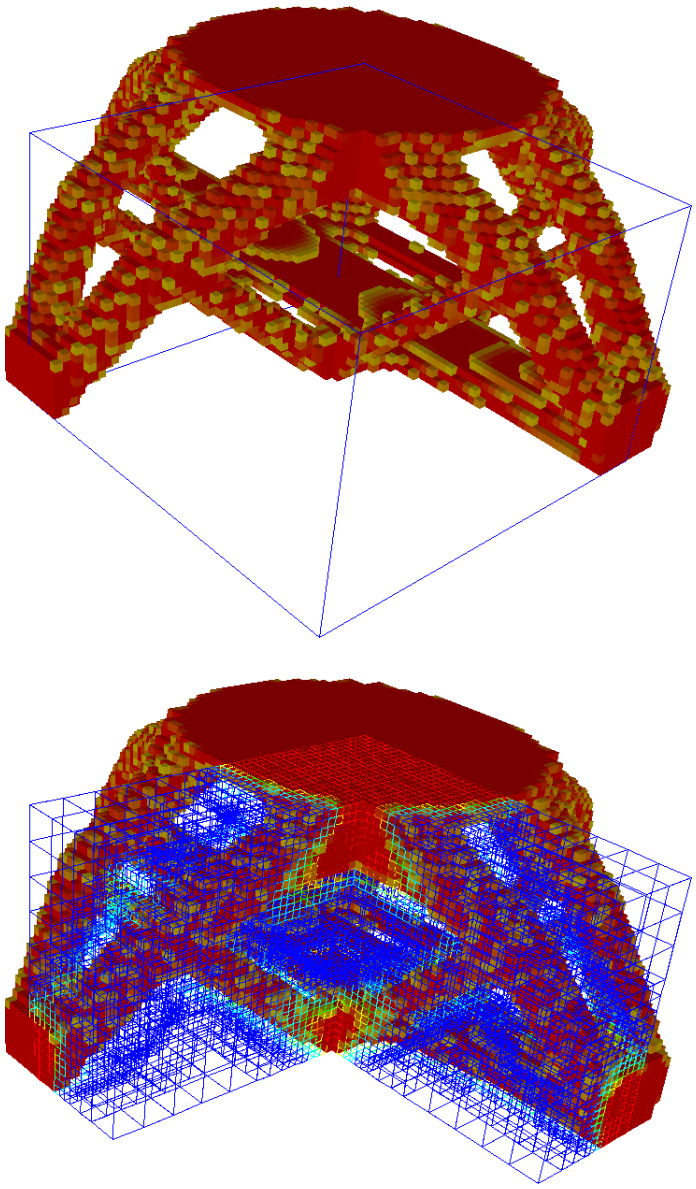


Figure 2.16: The optimization solution of the problem shown in Figure 2.14 on an adaptively refined mesh. The final mesh consists of 19736 B8 elements.

Chapter 3

Preconditioning for Topology Optimization

Most topology optimization applications, structural designs in particular, require finding the solutions of symmetric linear systems. The convergence rates of Krylov subspace methods for a symmetric matrix depend only on the spectrum of the matrix. In fact, the ratio between the absolute largest and smallest eigenvalue governs a worst-case upper bound on the convergence rate. In large-scale finite element simulations in physics and engineering, the linear systems tend to be ill-conditioned. In topology optimization, this problem is exacerbated by the wide range of magnitudes of the element densities.

Ill-conditioning creates two problems for numerical simulation. First, ill-conditioning may seriously affect the accuracy of the computed solution. Second, the convergence of iterative methods is poor for ill-conditioned problems. The second problem is generally addressed by proper preconditioning. In principle, preconditioning does not alleviate the potential accuracy problem, because a preconditioner that is effective for an ill-conditioned matrix must be fairly ill-conditioned itself. This leads to two multiplications by ill-conditioned matrices in each iteration (or three for two-sided preconditioning), which may, in turn, lead to serious accumulation of numerical errors. In certain cases, however, the accuracy problem can be relieved by properly rescaling the linear system. We show that this is the case for topology optimization. This leads to a preprocessing (rescaling) step and a preconditioning step (or two preconditioning steps, depending on one's point of view).

In Section 3.1, we discuss the preprocessing and preconditioning for topology optimization. The preconditioning for adaptive mesh refinement is further discussed in Chapter 5. In Section 3.2, we illustrate the idea of rescaling from a mechanical point of view for a 1D

problem. Borrvall and Petersson [17] suggest, without further discussion, that the condition number of the stiffness matrix can be as large as the ratio of maximum to minimum density. We show that this ratio provides only a lower bound on the condition number and that the actual condition number typically is much larger. The actual conditioning is a combination of this ratio and the conditioning of a corresponding problem with homogeneous density.

3.1 Scaling Issue in Topology Optimization

The following analysis addresses the ill-conditioning in the stiffness matrices. The two-norm condition number of a matrix $\boldsymbol{K}$ can be defined as

$$\kappa(\boldsymbol{K}) = \frac{\max_{\|\boldsymbol{u}\|=1} \|\boldsymbol{K}\boldsymbol{u}\|}{\min_{\|\boldsymbol{u}\|=1} \|\boldsymbol{K}\boldsymbol{u}\|}. \tag{3.1}$$

Since

$$\min_{\|\boldsymbol{u}\|=1} \|\boldsymbol{K}\boldsymbol{u}\| \leq \|\boldsymbol{K}\boldsymbol{e}_\ell\| = \|\boldsymbol{k}_\ell\| \leq \max_{\|\boldsymbol{u}\|=1} \|\boldsymbol{K}\boldsymbol{u}\|, \qquad \text{for any } \ell = 1, \ldots, n, \tag{3.2}$$

where $\boldsymbol{k}_\ell$ is the ℓth column of $\boldsymbol{K}$ and $\boldsymbol{e}_\ell$ is the Cartesian basis vector with the ℓth coefficient equal to 1, we have

$$\kappa(\boldsymbol{K}) \geq \frac{\|\boldsymbol{k}_{\ell_1}\|}{\|\boldsymbol{k}_{\ell_2}\|}, \qquad \text{for any } \ell_1, \ell_2 = 1, \ldots, n. \tag{3.3}$$

In a compliance minimization problem with the element-based approach discussed in Section 2.1.2, a column of the global stiffness matrix is given by

$$\boldsymbol{k}_\ell = \sum_{e \in N_\ell} \rho_e^p \boldsymbol{L}_e^T \boldsymbol{K}_0 \boldsymbol{L}_e \boldsymbol{e}_\ell, \tag{3.4}$$

where $\boldsymbol{K}_0$ is the unit element stiffness matrix, $\boldsymbol{L}_e$ is the local-to-global transformation matrix,

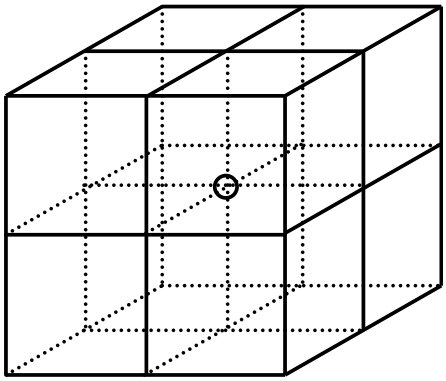


Figure 3.1: N_ℓ: the set of elements associated with the ℓth d.o.f. indicated by the circle in the middle.

and N_ℓ is the set of elements that are associated with the ℓth DOF. These usually form a $2 \times 2 \times 2$ block in the 3D mesh (see Figure 3.1). If the blocks associated with d.o.f. ℓ_1 and ℓ_2 are solid and void respectively, namely $\rho_e = 1$ for $e \in N_{\ell_1}$ and $\rho_e = \rho_o$ for $e \in N_{\ell_2}$, we have

$$\boldsymbol{k}_{\ell_1} = \sum_{e \in N_\ell} \boldsymbol{L}_e^T \boldsymbol{K}_0 \boldsymbol{L}_e \boldsymbol{e}_{\ell_1}, \tag{3.5}$$

$$\boldsymbol{k}_{\ell_2} = \rho_o^p \sum_{e \in N_\ell} \boldsymbol{L}_e^T \boldsymbol{K}_0 \boldsymbol{L}_e \boldsymbol{e}_{\ell_2}. \tag{3.6}$$

Then, assuming that the elements are uniform and isotropic, we have

$$\kappa(\boldsymbol{K}) \geq \frac{\|\boldsymbol{k}_{\ell_1}\|}{\|\boldsymbol{k}_{\ell_2}\|} = \frac{1}{\rho_o^p}. \tag{3.7}$$

For $\rho_o = 10^{-3}$ and $p = 3$, which are commonly used in topology optimization, the condition number of the stiffness matrix will be greater than 10^9 when solid and void areas begin to appear in the design domain. This bound holds for node-base methods as well, if we have $2 \times 2 \times 2$ blocks with all solid nodes and all void nodes.

Note that this analysis provides only a lower bound on the condition number, and that structures from homogeneous material can also have large condition numbers. However, the analysis suggests that, to a significant degree, the ill-conditioning comes from the poor

scaling of the material densities over the design domain. We can understand this intuitively as follows. A change in an algebraic degree of freedom, say the Cartesian basis vector $\boldsymbol{e}_j$, associated with a nodal basis function in a region with very small density corresponds to a displacement that requires a very small amount of energy ($\boldsymbol{e}_j^T \boldsymbol{K}(\boldsymbol{\rho}) \boldsymbol{e}_j$ small). However, that same change in an algebraic degree of freedom, $\boldsymbol{e}_i$, associated with a nodal basis function in a region with large density corresponds to a displacement of the same magnitude that requires a large amount of energy ($\boldsymbol{e}_i^T \boldsymbol{K}(\boldsymbol{\rho}) \boldsymbol{e}_i$ large). Since for symmetric $\boldsymbol{K}$

$$\kappa(\boldsymbol{K}(\rho)) \geq \frac{\boldsymbol{e}_i^T \boldsymbol{K}(\boldsymbol{\rho}) \boldsymbol{e}_i}{\boldsymbol{e}_j^T \boldsymbol{K}(\boldsymbol{\rho}) \boldsymbol{e}_j}, \tag{3.8}$$

this shows that the system is inherently ill-conditioned. Therefore, we expect that we can reduce the ill-conditioning due to the large variation in density by scaling the linear system such that changes of equal magnitude in algebraic degrees of freedom yield equal changes in energy ($\boldsymbol{e}_i^T \boldsymbol{K}(\boldsymbol{\rho}) \boldsymbol{e}_i = \boldsymbol{e}_j^T \boldsymbol{K}(\boldsymbol{\rho}) \boldsymbol{e}_j$ for all i and j). Since this is the case for a problem with homogeneous density, we expect that this scaling reduces the condition number of the stiffness matrix to roughly that for a similar problem with homogeneous density. Indeed, in general we obtain a condition number that is slightly better than that for a problem with constant density. Alternatively, in light of (3.7), we may want to scale the linear system such that all columns have equal norms. In the next section, we discuss the effects of rescaling for a simple 1D problem with heterogeneous density.

We propose to rescale the stiffness matrices $\boldsymbol{K}$ by multiplying with a diagonal matrix on both sides (for symmetry),

$$\widetilde{\boldsymbol{K}} = \boldsymbol{D}^{-1/2} \boldsymbol{K} \boldsymbol{D}^{-1/2}, \tag{3.9}$$

where the entries of the diagonal matrix $\boldsymbol{D}$ are either the diagonal coefficients of $\boldsymbol{K}$ or the absolute column sums of $\boldsymbol{K}$, i.e., $d_i = \|\boldsymbol{k}_i\|_1$.

To obtain rapid convergence for iterative methods, it is important to further reduce the condition number after rescaling by more general preconditioning techniques. For example,

we can apply an incomplete Cholesky decomposition [47] to the rescaled stiffness matrix:

$$\widetilde{\boldsymbol{K}} = \boldsymbol{D}^{-1/2}\boldsymbol{K}\boldsymbol{D}^{-1/2} \approx \boldsymbol{L}\boldsymbol{L}^{T}. \tag{3.10}$$

We solve the preconditioned system

$$\boldsymbol{L}^{-1}\widetilde{\boldsymbol{K}}\boldsymbol{L}^{-T}\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{f}} \tag{3.11}$$

for $\tilde{\boldsymbol{u}}$, where $\tilde{\boldsymbol{f}} = \boldsymbol{L}^{-1}\boldsymbol{D}^{-1/2}\boldsymbol{f}$. Then, we compute

$$\boldsymbol{u} = \boldsymbol{D}^{-1/2}\boldsymbol{L}^{-T}\tilde{\boldsymbol{u}} \tag{3.12}$$

to obtain the solution of the original system $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$.

We note that diagonal scaling does not decrease the relative accuracy of the matrix coefficients, and hence such scaling leads to a real improvement in the worst case numerical error in the computed solution. The second type of preconditioning, e.g., the incomplete Cholesky decomposition, improves the rate of convergence, but does not typically affect the accuracy of the computed solutions. Since this type of preconditioners may fail or become very poor for a very ill-conditioned matrix, we always explicitly rescale the stiffness matrix before computing the more general preconditioner.

We examine the conditioning of the linear systems arising in the topology optimization problem shown in Figure 2.2 on a $18 \times 6 \times 3$ mesh. Figure 3.2 shows the condition numbers of four matrices at each optimization step, namely the original stiffness matrix, the rescaled matrix, the original matrix preconditioned by incomplete Cholesky, the rescaled matrix preconditioned by incomplete Cholesky. The condition numbers of the original stiffness matrices quickly rise to about 10^{11} after only a few optimization steps. However, the condition numbers of the rescaled matrices remain at about the same level as those at the beginning (approximately 10^5). We note that incomplete Cholesky takes care of the poor

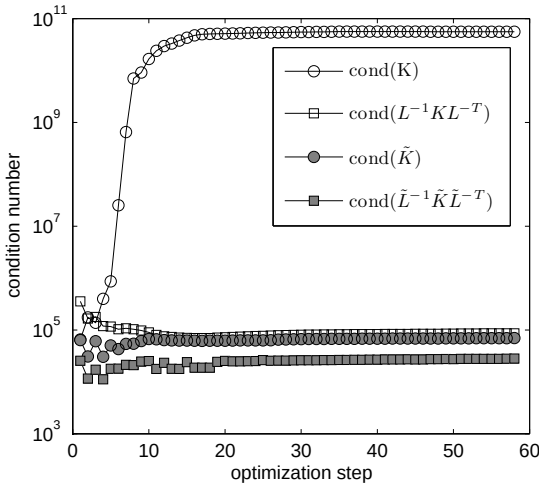


Figure 3.2: Condition numbers of the unpreconditioned and preconditioned systems with and without rescaling for the model problem in Figure 2.2 on a $18 \times 6 \times 3$ mesh with the node-based approach.

scaling to some degree. However, as we mentioned above, it will not improve the accuracy of the solution. And computing the incomplete Cholesky factor after rescaling does improve the conditioning over the incomplete Cholesky preconditioned system without rescaling.

3.2 1D Rescaling Analysis

In this section, we use an idealized 1D elasticity problem with piecewise constant modulus of elasticity to explain the idea of rescaling. Consider the following problem.

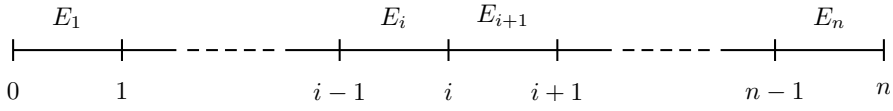


Figure 3.3: Piecewise constant modulus of elasticity E_i.

Find $u(x)$ with boundary conditions $u(0) = 0$ and $u(1) = 1$, such that

$$a(u, v) \equiv \int_0^1 E(x) u_x v_x dx = 0, \quad \text{with} \quad E(x) \geq E_0 > 0, \tag{3.13}$$

for all v with $v(0) = v(1) = 0$. Furthermore, following the typical case of topology optimization, we assume that E is piecewise constant (see Figure 3.3) and varies over a large range of values.

For simplicity, we discretize the problem using piecewise linear nodal basis functions and a mesh with uniform elements. This yields the following linear system:

$$\begin{bmatrix} E_1+E_2 & -E_2 & & & \\ -E_2 & E_2+E_3 & -E_3 & & \\ & \ddots & \ddots & \ddots & \\ & & -E_{n-1} & E_{n-1}+E_n \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ E_n \end{pmatrix}. \tag{3.14}$$

We can write this system of equations as follows (note $(Eu_x)_x = 0 \Leftrightarrow Eu_x = \text{constant}$):

$$E_i(u_i - u_{i-1}) - E_{i+1}(u_{i+1} - u_i) \;=\; 0, \quad \text{for } i = 1, \ldots, n-1,$$

where we have used $u_0 = 0$ and $u_n = 1$. Introducing the difference matrix

$$\boldsymbol{D}_1 = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}$$

and the diagonal matrix $\boldsymbol{\Omega} = \text{diag}(E_1, E_2, \ldots, E_{n-1})$, we can write (3.14) as

$$\left(\boldsymbol{D}_1^T \boldsymbol{\Omega} \boldsymbol{D}_1 + E_n \boldsymbol{e}_{n-1} \boldsymbol{e}_{n-1}^T\right) \boldsymbol{u} = E_n \boldsymbol{e}_{n-1}. \tag{3.15}$$

For a problem with constant modulus of elasticity, E, this equation gives

$$E\left(\boldsymbol{D}_1^T \boldsymbol{D}_1 + \boldsymbol{e}_{n-1} \boldsymbol{e}_{n-1}^T\right) \boldsymbol{u} = E \boldsymbol{e}_{n-1}, \tag{3.16}$$

where $\boldsymbol{D}_1^T\boldsymbol{D}_1 + \boldsymbol{e}_{n-1}\boldsymbol{e}_{n-1}^T$ is the well-known tridiagonal matrix with coefficients $[-1\ 2\ -1]$.

Next, we want to demonstrate two things. The comparison of (3.15) with (3.16) shows that the extreme ill-conditioning in (3.15) must arise from the scaling introduced by $\boldsymbol{\Omega}$. First, we demonstrate that this leads to a condition number (bound) that is roughly the product of the condition number of the constant elasticity problem and the condition number of $\boldsymbol{\Omega}$. Second, we show that following a proper rescaling the condition number is commensurate with the condition number for the constant elasticity case, if the solution is properly defined. We note that for general choices of $\boldsymbol{\Omega}$ there may be no diagonal scaling that reduces the condition number. In 1D, for example, if we have two non-adjacent 'holes', the displacement for material in between the holes is not properly defined (as the modulus of elasticity goes to zero), since there is no connection to any point with a fixed displacement. In higher dimensions this is rarely a problem, as the topology optimization algorithm leads to energetically favorable solutions lacking such anomalies.

Below, we need the following well-known result for symmetric positive definite matrices $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ and $\alpha, \beta \in \mathbb{R}^+$ [3, pp. 338-9]. Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be such that for all $\boldsymbol{u} \neq 0$

$$\alpha \leq \frac{\boldsymbol{u}^T\boldsymbol{A}\boldsymbol{u}}{\boldsymbol{u}^T\boldsymbol{B}\boldsymbol{u}} \quad \leq \quad \beta. \tag{3.17}$$

Then

$$\kappa(\boldsymbol{B}^{-1/2}\boldsymbol{A}\boldsymbol{B}^{-1/2}) \quad \leq \quad \frac{\beta}{\alpha}, \tag{3.18}$$

where κ denotes the condition number.

Using (3.17–3.18), we can bound the condition number of the matrix in (3.15) as follows:

$$\kappa(\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1 + E_n\boldsymbol{e}_{n-1}\boldsymbol{e}_{n-1}^T) \quad = \quad \frac{\max_{\|\boldsymbol{u}\|=1}\boldsymbol{u}^T(\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1 + E_n\boldsymbol{e}_{n-1}\boldsymbol{e}_{n-1}^T)\boldsymbol{u}}{\min_{\|\boldsymbol{u}\|=1}\boldsymbol{u}^T(\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1 + E_n\boldsymbol{e}_{n-1}\boldsymbol{e}_{n-1}^T)\boldsymbol{u}}. \tag{3.19}$$

Let $\boldsymbol{y} = \boldsymbol{D}_1\boldsymbol{u}$ and hence $\boldsymbol{u} = \boldsymbol{D}_1^{-1}\boldsymbol{y}$. Then

$$\begin{aligned} \boldsymbol{u}^T\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1\boldsymbol{u} + E_n\boldsymbol{u}^T\boldsymbol{e}_{n-1}\boldsymbol{e}_{n-1}^T\boldsymbol{u} &= (\boldsymbol{D}_1\boldsymbol{u})^T\boldsymbol{\Omega}(\boldsymbol{D}_1\boldsymbol{u}) + E_n u_{n-1}^2 \\ &= \frac{\boldsymbol{y}^T\boldsymbol{\Omega}\boldsymbol{y} + E_n(y_1 + \ldots + y_{n-1})^2}{(\boldsymbol{D}_1^{-1}\boldsymbol{y})^T(\boldsymbol{D}_1^{-1}\boldsymbol{y})}. \end{aligned} \tag{3.20}$$

Let $E_{\min} = \min_i(E_i)$, $E_{\max} = \max_i(E_i)$, and let $\lambda_{\min}$ be the smallest eigenvalue of the matrix $\boldsymbol{D}_1\boldsymbol{D}_1^T$ and $\lambda_{\max}$ its largest eigenvalue. Furthermore, note that $\boldsymbol{D}_1\boldsymbol{D}_1^T$ and $\boldsymbol{D}_1^T\boldsymbol{D}_1$ have the same eigenvalues. Since $\boldsymbol{y}$ appears quadratically in both the numerator and the denominator, we can assume $\boldsymbol{y}$ to be normalized. Then, (3.20) gives

$$E_{\min}\lambda_{\min} \leq \frac{\boldsymbol{y}^T\boldsymbol{\Omega}\boldsymbol{y} + E_n(y_1 + \ldots + y_{n-1})^2}{(\boldsymbol{D}_1^{-1}\boldsymbol{y})^T(\boldsymbol{D}_1^{-1}\boldsymbol{y})} \leq nE_{\max}\lambda_{\max}, \tag{3.21}$$

which finally leads to

$$\kappa(\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1 + E_n\boldsymbol{e}_{n-1}\boldsymbol{e}_{n-1}^T) \leq n\frac{E_{\max}}{E_{\min}}\frac{\lambda_{\max}}{\lambda_{\min}} = n\kappa(\boldsymbol{\Omega})\kappa(\boldsymbol{D}_1^T\boldsymbol{D}_1). \tag{3.22}$$

Next, we show that scaling a problem (without non-adjacent 'holes') reduces the condition number of the linear system to roughly that of a problem with constant elasticity. Let

$$\boldsymbol{S} = \mathrm{diag}(E_1 + E_2, E_2 + E_3, \ldots, E_{n-1} + E_n). \tag{3.23}$$

We have

$$\begin{aligned} &\frac{\boldsymbol{u}^T(\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1 + E_n e_{n-1}e_{n-1}^T)\boldsymbol{u}}{\boldsymbol{u}^T\boldsymbol{S}\boldsymbol{u}} = \\ &\frac{E_1u_1^2 + E_2(u_1 - u_2)^2 + \ldots + E_{n-1}(u_{n-2} - u_{n-1})^2 + E_nu_{n-1}^2}{E_1u_1^2 + E_2(u_1^2 + u_2^2) + \ldots + E_{n-1}(u_{n-2}^2 + u_{n-1}^2) + E_nu_{n-1}^2} = \\ &\frac{(E_1 + E_2)u_1^2 + \ldots + (E_{n-1} + E_n)u_{n-1}^2 - 2(E_2u_1u_2 + \ldots + E_{n-1}u_{n-2}u_{n-1})}{(E_1 + E_2)u_1^2 + \ldots + (E_{n-1} + E_n)u_{n-1}^2} = \\ &1 - \frac{2(E_2u_1u_2 + \ldots + E_{n-1}u_{n-2}u_{n-1})}{(E_1 + E_2)u_1^2 + \ldots + (E_{n-1} + E_n)u_{n-1}^2} = \end{aligned}$$

$$1 - \frac{2(E_2 u_1 u_2 + \ldots + E_{n-1} u_{n-2} u_{n-1})}{E_1 u_1^2 + E_2(u_1^2 + u_2^2) + \ldots + E_{n-1}(u_{n-2}^2 + u_{n-1}^2) + E_n u_{n-1}^2}. \tag{3.24}$$

It is easy to see that the maximum of (3.24) is bounded by two. The condition number therefore depends primarily on the minimum of (3.24). We consider three examples.

The first example examines the case of constant modulus of elasticity. The second example demonstrates that the case of a bar with variable modulus (solid bar with a 'hole') leads to approximately the same condition number after scaling as the case of a bar with constant modulus (homogeneous). The third example shows that for the hypothetical case of a 1D bar with two non-adjacent 'holes', scaling cannot remove the actual singularity.

Example 1 – Constant Modulus

For a constant modulus of elasticity, (3.24) leads to

$$\begin{aligned} & \frac{\boldsymbol{u}^T(E\boldsymbol{D}_1^T\boldsymbol{D}_1 + Ee_{n-1}e_{n-1}^T)\boldsymbol{u}}{\boldsymbol{u}^T\boldsymbol{S}\boldsymbol{u}} \\ = \; & 1 - \frac{2u_1u_2 + \ldots + 2u_{n-2}u_{n-1}}{u_1^2 + (u_1^2 + u_2^2) + \ldots + (u_{n-2}^2 + u_{n-1}^2) + u_{n-1}^2}. \end{aligned} \tag{3.25}$$

The minimum for (3.25) is obtained for $u_i = \sin(\pi i h)$, which gives $u_{i-1}u_i \approx u_i^2$ and minimizes the influence of the terms Eu_1^2 and Eu_{n-1}^2. This leads to a condition number for the preconditioned system of $O(h^{-2})$.

Example 2 – Variable Modulus

Now consider a problem with a 'hole' at the end of the bar; $E_i = 1$ for $i = 1, \ldots, n-5$, where $n \gg 5$, and $E_i = \varepsilon$, $0 < \varepsilon \ll 1$ for the remaining elements. The minimum for (3.24) is obtained for a vector u such that $|u_i| = O(1)$ for $i = 1, \ldots, n-5$ and $|u_i| = O(\varepsilon)$ for the remaining elements. After substituting for the E_i in (3.24) and dropping the u_i terms that

are $O(\varepsilon)$, we need to minimize the following expression:

$$1 - \frac{2u_1u_2 + \ldots + 2u_{n-6}u_{n-5}}{u_1^2 + (u_1^2 + u_2^2) + \ldots + (u_{n-6}^2 + u_{n-5}^2) + \varepsilon u_{n-5}^2}. \tag{3.26}$$

Comparing (3.26) to (3.25), we see that this minimization problem is essentially the same as the constant modulus example (with a few terms of small magnitude dropped). Therefore, the resulting condition number is about the same.

Example 3: Hypothetical Case

Finally, consider a hypothetical problem of a 1D bar with two non-adjacent 'holes'. Let $n = 5$, and let $E_1 = E_3 = E_5 = 1$ and $E_2 = E_4 = \varepsilon$. Now taking $u_1 = u_4 = 0$ and $u_2 = u_3 = 1$ in (3.24) gives

$$\begin{aligned} \min_{\boldsymbol{u}\neq 0} \frac{\boldsymbol{u}^T(\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1 + E_n e_{n-1}e_{n-1}^T)\boldsymbol{u}}{\boldsymbol{u}^T\boldsymbol{S}\boldsymbol{u}} &\leq \\ 1 - \frac{2E_3u_2u_3}{E_2u_2^2 + E_3(u_2^2 + u_3^2) + E_4u_3^2} &= \\ 1 - \frac{2}{\varepsilon + 2 + \varepsilon} &= \frac{\varepsilon}{1+\varepsilon}. \end{aligned} \tag{3.27}$$

(3.27) can be made arbitrarily small, and the condition number $\kappa(\boldsymbol{D}_1^T\boldsymbol{\Omega}\boldsymbol{D}_1 + E_n e_{n-1}e_{n-1}^T)$ can be made arbitrarily large.

Chapter 4

Recycling Krylov Subspace Methods

In topology optimization and many other numerical algorithms, e.g., Newton's method for nonlinear problems, we need to solve a sequence of linear systems that evolve slowly from one to the next. With Krylov subspace methods, we can collect a subspace of the Krylov subspace as we iterate and use it to accelerate the solution of the next system [51, 39]. This is the idea behind Krylov subspace recycling.

In most topology optimization problems the system matrices are symmetric. In most cases, they are also positive definite. However, in some applications, e.g., topology design with dynamic vibrations, they can be indefinite [63]. So, the minimum residual method (MINRES) is the most suitable iterative solver for topology optimization. It keeps a short-term recurrence by exploiting the symmetry, and therefore is very efficient. On the other hand, recycling methods for general matrices, like GCRODR [51], are less efficient for symmetric systems because of their long Arnoldi recurrence. We adapt the MINRES method to include recycling and to keep the short-term recurrence, and we make the selection of recycle space much cheaper by further exploiting symmetry in the underlying generalized eigenvalue problem.

In this chapter, we first introduce the motivation of Krylov subspace recycling in Section 4.1. In Section 4.2, we discuss GCRODR, the recycling method for general systems [51]. We address two issues for Krylov subspace recycling, namely which subspace to recycle and how to use the recycle space in solving the next system. In Section 4.3, we adapt the recycling idea to the MINRES method for symmetric systems. We refer to our recycling MINRES method as RMINRES. The key issue is to select the desired subspace for recycling while

maintaining the efficiency of MINRES. We also discuss the choice of subspace from where the recycle space is selected and the simplification of subspace selection formulae. Both of them make the recycle space selection much cheaper. In Section 4.5, we present numerical results of the recycling MINRES method for topology optimization problems, and we discuss the impact on performance that the RMINRES parameters have. In Section 4.6, we discuss Krylov subspace recycling for adaptive meshes.

4.1 Motivation of Recycling

Consider a general linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and an initial guess $\boldsymbol{x}_0$. The Generalized Minimum Residual method (GMRES) [59] builds the Krylov subspace, $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) = \text{span}\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \cdots, \boldsymbol{A}^{m-1}\boldsymbol{r}_0\}$, where $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, and computes the optimal solution over that subspace. We use the Arnoldi recurrence [2, 58] to obtain an orthonormal basis of the Krylov subspace, see Algorithm 4.1.

Algorithm 4.1: Arnoldi Recurrence

$\boldsymbol{v}_1 \leftarrow \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$;
for $i \geq 1$ **do**
  $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{A}\boldsymbol{v}_i$;
  for $k = 1, \cdots, i$ **do**
    $h_{k,i} \leftarrow \boldsymbol{v}_i^T \boldsymbol{v}_{i+1}$;
    $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_{i+1} - h_{k,i}\boldsymbol{v}_k$;
  end
  $h_{i,i+1} \leftarrow \|\boldsymbol{v}_{i+1}\|$;
  $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_{i+1}/h_{i,i+1}$;
end

This can be written in matrix form as

$$\boldsymbol{A}\boldsymbol{V}_m = \boldsymbol{V}_{m+1}\underline{\boldsymbol{H}}_m, \tag{4.1}$$

where the columns of $\boldsymbol{V}_m$ are $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$; the columns of $\boldsymbol{V}_{m+1}$ are $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{m+1}$; and $\underline{\boldsymbol{H}}_m$ is

an $(m+1) \times m$ upper Hessenberg matrix with coefficients $\{h_{ij}\}$. At step m, GMRES finds the solution

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{\varepsilon}_m \tag{4.2}$$

in the affine subspace $\boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ such that it minimizes the residual norm.

Since all vectors in $\boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ can be written as $\boldsymbol{x}_0 + q_m(\boldsymbol{A})\boldsymbol{r}_0$, where $q_m \in \mathcal{P}_{m-1}$ and $\mathcal{P}_{m-1}$ is the set of polynomials of degree $m-1$, the residual norm of GMRES at step m would be

$$\begin{aligned} \|\boldsymbol{r}_m\| = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_m\| &= \min_{q_m \in \mathcal{P}_{m-1}} \|(\boldsymbol{I} - \boldsymbol{A}q_m(\boldsymbol{A}))\boldsymbol{r}_0\| \\ &= \min_{p_m \in \mathcal{P}_m^{(0)}} \|p_m(\boldsymbol{A})\boldsymbol{r}_0\|, \end{aligned} \tag{4.3}$$

where $\mathcal{P}_m^{(0)} = \{p_m \in \mathcal{P}_m | p_m(0) = 1\}$.

If $\boldsymbol{A}$ is diagonalizable, we have $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^{-1}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix whose coefficients are the eigenvalues of $\boldsymbol{A}$. Following (4.3), we have

$$\begin{aligned} \|\boldsymbol{r}_m\| &= \min_{p_m \in \mathcal{P}_m^{(0)}} \|\boldsymbol{X}p_m(\boldsymbol{\Lambda})\boldsymbol{X}^{-1}\boldsymbol{r}_0\| \\ &\leq \min_{p_m \in \mathcal{P}_m^{(0)}} \|\boldsymbol{X}\|\|\boldsymbol{X}^{-1}\|\|p_m(\boldsymbol{\Lambda})\|\|\boldsymbol{r}\| \end{aligned} \tag{4.4}$$

Therefore, the convergence rate of GMRES or other Krylov methods that minimize 2-norm of the residual is bounded by [58, p. 195]

$$\frac{\|\boldsymbol{r}_m\|}{\|\boldsymbol{r}_0\|} \leq \text{cond}(\boldsymbol{X}) \min_{p_m \in \mathcal{P}_m^{(0)}} \max_{\lambda \in \Lambda(\boldsymbol{A})} |p_m(\lambda)|, \tag{4.5}$$

where $\Lambda(\boldsymbol{A})$ is the set of eigenvalues of $\boldsymbol{A}$. If $\boldsymbol{X}$ is not too ill-conditioned, which is true for most problems addressed in this thesis, this bound mainly depends on the spectrum of the matrix. For symmetric $\boldsymbol{A}$, the similarity transformation matrix $\boldsymbol{X}$ is orthogonal, which

means $\mathrm{cond}(\boldsymbol{X}) = 1$. In this case, the convergence rate solely depends on the spectrum of $\boldsymbol{A}$. If we could consider only an appropriate subset of $\Lambda(\boldsymbol{A})$, then

$$\min_{p_m \in \mathcal{P}_m^{(0)}} \max_{\lambda \in \Lambda(\boldsymbol{A})/S} |p_m(\lambda)| \tag{4.6}$$

could be significantly smaller than the bound in (4.5). In that case, the rate of convergence would be greatly improved. This can be achieved by including the corresponding (approximate) invariant subspace in the search space.

Given the normalization condition, $p_m(0) = 1$, it is often effective to remove the eigenvalues close to the origin. Depending on the problem, sometimes it is helpful to remove outermost eigenvalues. For example, if there are only a few outermost eigenvalues separated from the rest. For symmetric indefinite systems with only a few negative eigenvalues, it would be extremely useful to remove the negative eigenvalues.

When solving a sequence of linear systems that change slowly from one to the next, we often expect that the eigenvectors and eigenvalues of two consecutive systems are similar. Therefore, an invariant subspace of one system approximates that of the next one, and thus can be included into the search space of the next system to remove the corresponding eigenvalues. This is the key of recycling approximate invariant subspaces. In general, this assumption holds for systems resulting from PDEs. Especially when we consider the smallest eigenvalues, the corresponding invariant subspace consists mostly of the smoothest modes, which change least when the system changes smoothly.

4.2 Krylov Subspace Recycling

Now, we introduce the Krylov subspace recycling method for general systems [51]. We address two issues, namely how to obtain an appropriate invariant subspace for recycling and how to use such subspace in solving the next system.

For any subspace $\mathcal{S} \subseteq \mathbb{R}^n$, we define $\boldsymbol{y} \in \mathcal{S}$ as a Ritz vector of $\boldsymbol{A}$ with Ritz value θ with respect to $\mathcal{S}$ [68, p. 282] if

$$\boldsymbol{A}\boldsymbol{y} - \theta\boldsymbol{y} \perp \boldsymbol{w}, \qquad \forall \boldsymbol{w} \in \mathcal{S}. \tag{4.7}$$

We also define $\tilde{\boldsymbol{y}} \in \mathcal{S}$ as a harmonic Ritz vector of $\boldsymbol{A}$ with harmonic Ritz value $\tilde{\theta}$ with respect to $\mathcal{S}$ [68, p. 292] if

$$\boldsymbol{A}\tilde{\boldsymbol{y}} - \tilde{\theta}\tilde{\boldsymbol{y}} \perp \boldsymbol{A}\boldsymbol{w}, \qquad \forall \boldsymbol{w} \in \mathcal{S}. \tag{4.8}$$

Let the columns of matrix $\boldsymbol{W}$ form an orthonormal basis for subspace $\mathcal{S}$. We represent $\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}$ on this basis, $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{z}$ and $\tilde{\boldsymbol{y}} = \boldsymbol{W}\tilde{\boldsymbol{z}}$. Then, (4.7) and (4.8) lead to the following generalized eigenvalue problems respectively:

$$\boldsymbol{W}^T\boldsymbol{A}\boldsymbol{W}\boldsymbol{z} = \theta\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{z} = \theta\boldsymbol{z}, \tag{4.9}$$

$$\boldsymbol{W}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{W}\tilde{\boldsymbol{z}} = \tilde{\theta}\boldsymbol{W}^T\boldsymbol{A}^T\boldsymbol{W}\tilde{\boldsymbol{z}}. \tag{4.10}$$

While Ritz values tend to approximate the outermost eigenvalues of $\boldsymbol{A}$ in magnitude, harmonic Ritz values tend to approximate the eigenvalues closest to the origin [49]. Moreover, the Ritz vectors and the harmonic Ritz vectors approximate the corresponding eigenvectors. Therefore, as discussed in Section 4.1, including the harmonic Ritz vectors in the Krylov subspace would remove the corresponding eigenvalues as in (4.6), and hence improve the convergence rate of a Krylov subspace method.

When we build a Krylov subspace as we solve a linear system iteratively, we can compute the harmonic Ritz vectors of $\boldsymbol{A}$ with respect to the Krylov subspace $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ by solving a generalized eigenvalue problem as in (4.10). Then, we get an approximate invariant subspace corresponding to the smallest harmonic Ritz values as a recycle space.

When solving one system from a sequence, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, we use the recycle space from the solution of the previous system as follows. We represent the basis for the recycle space by

the columns of a matrix $\boldsymbol{U}$, such that

$$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{U}, \qquad \boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}. \tag{4.11}$$

This can always be done with a QR decomposition [51]. In addition, we adapt the Arnoldi process to make each new Arnoldi vector $\boldsymbol{v}$ orthogonal to range $(\boldsymbol{C})$. This leads to the recurrence in Algorithm 4.2.

Algorithm 4.2: MODIFIED ARNOLDI RECURRENCE

$\boldsymbol{v}_1 \leftarrow \boldsymbol{r}_0 - \boldsymbol{C}(\boldsymbol{C}^T\boldsymbol{r}_0)$;
$\boldsymbol{v}_1 \leftarrow \boldsymbol{v}_1/\|\boldsymbol{v}_1\|$;
for $i \geq 1$ **do**
    $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{A}\boldsymbol{v}_i$;
    $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_{i+1} - \boldsymbol{C}(\boldsymbol{C}^T\boldsymbol{v}_{i+1})$;
    for $k = 1, \cdots, i$ **do**
        $h_{k,i} \leftarrow \boldsymbol{v}_i^T\boldsymbol{v}_{i+1}$;
        $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_{i+1} - h_{k,i}\boldsymbol{v}_k$;
    end
    $h_{i,i+1} \leftarrow \|\boldsymbol{v}_{i+1}\|$;
    $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{v}_{i+1}/h_{i,i+1}$;
end

This can be written in matrix form as

$$\begin{aligned} (\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{A}\boldsymbol{V}_m &= \boldsymbol{V}_{m+1}\underline{\boldsymbol{H}}_m \quad \Longleftrightarrow \\ \boldsymbol{A}\boldsymbol{V}_m &= \boldsymbol{C}\boldsymbol{C}^T\boldsymbol{A}\boldsymbol{V}_m + \boldsymbol{V}_{m+1}\underline{\boldsymbol{H}}_m, \end{aligned} \tag{4.12}$$

where $\underline{\boldsymbol{H}}_m$ is still an $(m+1) \times m$ upper Hessenberg matrix. Next, we compute the vector $\boldsymbol{\varepsilon}_m = \boldsymbol{U}\boldsymbol{z}_m + \boldsymbol{V}_m\boldsymbol{y}_m$, such that $\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{\varepsilon}_m$ minimizes $\|\boldsymbol{r}_m\|$.

$$\|\boldsymbol{r}_m\| = \left\| \boldsymbol{r}_0 - \boldsymbol{A}\left[\boldsymbol{U}\ \boldsymbol{V}_m\right] \begin{pmatrix} \boldsymbol{z}_m \\ \boldsymbol{y}_m \end{pmatrix} \right\|$$

$$
\begin{aligned}
&= \left\| [\boldsymbol{C}\ \boldsymbol{V}_{m+1}] \left(\begin{pmatrix} \boldsymbol{C}^T \boldsymbol{r}_0 \\ \beta \boldsymbol{e}_1 \end{pmatrix} - \begin{bmatrix} \boldsymbol{I} & \boldsymbol{B}_m \\ \boldsymbol{0} & \underline{\boldsymbol{H}}_m \end{bmatrix} \begin{pmatrix} \boldsymbol{z}_m \\ \boldsymbol{y}_m \end{pmatrix} \right) \right\| \\
&= \left\| \begin{pmatrix} \boldsymbol{C}^T \boldsymbol{r}_0 \\ \beta \boldsymbol{e}_1 \end{pmatrix} - \begin{bmatrix} \boldsymbol{I} & \boldsymbol{B}_m \\ \boldsymbol{0} & \underline{\boldsymbol{H}}_m \end{bmatrix} \begin{pmatrix} \boldsymbol{z}_m \\ \boldsymbol{y}_m \end{pmatrix} \right\|, \qquad (4.13)
\end{aligned}
$$

where $\beta = \|(\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{r}_0\|$ and $\boldsymbol{B}_m = \boldsymbol{C}^T \boldsymbol{A} \boldsymbol{V}_m$. This least squares problem can be solved using the QR decomposition of $\underline{\mathbf{H}}_m$. This approach derives from the GCRO method [28] and is also used in the GCRODR method and GCROT with recycling [51, 39, 29].

An important issue for GMRES is that it relies (for general matrices) on a complete orthogonalization of the Krylov subspace. Therefore, as the Krylov subspace expands, the memory needed for the orthogonal basis vectors and the computational time for orthogonalization increase. As a result, normally restarting is required for GMRES, and we call the iterations between two restarts a *cycle*. To mitigate the reduced convergence rate due to the loss of orthogonality to the old Arnoldi vectors caused by restarting, we use the recycle space immediately in the next cycle for the same system.

4.3 Recycling Minimum Residual Method

In this section, we consider the MINRES method for symmetric systems, which are the most common cases in topology optimization problems. Both MINRES and GMRES minimize the two-norm of the residual over the Krylov subspace, and thus have the same convergence rate for symmetric systems in exact arithmetic. The difference is that MINRES utilizes the symmetry of the matrix, and the resulting Lanczos three-term recurrence leads to significant reductions in memory requirements and computational cost. For symmetric systems, GMRES-based GCRODR may still improve the convergence rate over MINRES. However, because of its long orthogonalization, GCRODR becomes less efficient compared to MINRES in terms of overall performance. Therefore, we need to keep the short-term

recurrence as we do recycling and keep the recycle space selection cheap.

We can use the matrices $\boldsymbol{U}$ and $\boldsymbol{C}$, which define the recycle space and are obtained from solving previous linear systems, see 4.11), in the same way as in GCRODR. This leads to the same recurrence as in (4.12). And since we make the Lanczos vectors orthogonal to the recycle space $\boldsymbol{C}$, we have

$$\boldsymbol{C}^T\boldsymbol{V}_m = 0 \quad \Longrightarrow \quad \boldsymbol{V}_m = (\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{V}_m. \tag{4.14}$$

Then (4.12) leads to

$$\begin{aligned} & (\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{V}_m = \boldsymbol{V}_{m+1}\underline{\boldsymbol{H}}_m \\ \Longrightarrow \quad & \boldsymbol{V}_m(\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{A}(\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{V}_m = \boldsymbol{H}_m \end{aligned} \tag{4.15}$$

where $\boldsymbol{H}_m$ is the leading $m \times m$ submatrix of $\underline{\boldsymbol{H}}_m$. Therefore, the symmetry of $\boldsymbol{A}$ implies the symmetry of $\boldsymbol{H}_m$. Since $\underline{\boldsymbol{H}}_m$ is also an upper Hessenberg matrix, this gives a tridiagonal $\underline{\boldsymbol{H}}_m$, which we will denote as $\underline{\boldsymbol{T}}_m$ from now on. So, including the recycle space into the Krylov subspace does not affect the Lanczos recurrence of MINRES. Now we explain how we adapt the MINRES method as described in [72, p. 84–86] or [33, p. 41–44] to include the recycle space. Similarly as in GCRODR, we need to compute the vector

$$\boldsymbol{\varepsilon}_m = \boldsymbol{U}\boldsymbol{z}_m + \boldsymbol{V}_m\boldsymbol{y}_m, \tag{4.16}$$

such that $\boldsymbol{x}_m = \boldsymbol{x}_0 + \boldsymbol{\varepsilon}_m$ minimizes $\|\boldsymbol{r}_m\|$. For symmetric $\boldsymbol{A}$, (4.13) becomes

$$\|\boldsymbol{r}_m\| = \left\| \begin{pmatrix} \boldsymbol{C}^T\boldsymbol{r}_0 \\ \beta\boldsymbol{e}_1 \end{pmatrix} - \begin{bmatrix} \boldsymbol{I} & \boldsymbol{B}_m \\ \boldsymbol{0} & \underline{\boldsymbol{T}}_m \end{bmatrix} \begin{pmatrix} \boldsymbol{z}_m \\ \boldsymbol{y}_m \end{pmatrix} \right\|, \tag{4.17}$$

where $\beta = \|(\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{r}_0\|$ and $\boldsymbol{B}_m = \boldsymbol{C}^T\boldsymbol{A}\boldsymbol{V}_m$. The QR decomposition of $\underline{\boldsymbol{T}}_m$ gives

$$\underline{\boldsymbol{T}}_m = \widehat{\boldsymbol{G}}_m^T \underline{\boldsymbol{F}}_m, \tag{4.18}$$

where $\widehat{\boldsymbol{G}}_m$ is an orthogonal matrix of size $(m+1) \times (m+1)$, and $\underline{\boldsymbol{F}}_m$ is an upper triangular matrix of size $(m+1) \times m$ with bandwidth 3. The matrix $\widehat{\boldsymbol{G}}$ is the product of a series of orthogonal matrices defining plane rotations, also called Givens rotations, $\widehat{\boldsymbol{G}}_m = \boldsymbol{G}_m \cdots \boldsymbol{G}_2 \boldsymbol{G}_1$ (see Algorithm 1). Let $\boldsymbol{F}_m$ be the leading $m \times m$ submatrix of $\underline{\boldsymbol{F}}_m$, and $\{f_{ij}\}$ and $\{t_{ij}\}$ be the coefficients of $\underline{\boldsymbol{F}}_m$ and $\underline{\boldsymbol{T}}_m$ respectively. The solution of the least squares problem (4.17) is then

$$\boldsymbol{y}_m = \boldsymbol{F}_m^{-1} \widehat{\boldsymbol{G}}_m \beta \boldsymbol{e}_1, \quad \boldsymbol{z}_m = \boldsymbol{C}^T \boldsymbol{r}_0 - \boldsymbol{B}_m \boldsymbol{y}_m. \tag{4.19}$$

This leads to

$$\begin{aligned} \boldsymbol{x}_m &= \boldsymbol{x}_0 + \boldsymbol{U}\boldsymbol{z}_m + \boldsymbol{V}_m \boldsymbol{y}_m \\ &= \boldsymbol{x}_0 + \boldsymbol{C}^T \boldsymbol{r}_0 - \boldsymbol{U}\boldsymbol{B}_m \boldsymbol{y}_m + \boldsymbol{V}_m \boldsymbol{y}_m \\ &= \hat{\boldsymbol{x}}_0 - \boldsymbol{U}\boldsymbol{B}_m \boldsymbol{y}_m + \boldsymbol{V}_m \boldsymbol{y}_m, \end{aligned} \tag{4.20}$$

where $\hat{\boldsymbol{x}}_0 = \boldsymbol{x}_0 + \boldsymbol{C}^T \boldsymbol{r}_0$. Since $\boldsymbol{V}_m$ can be computed by a three-term recurrence, we only need the last two columns of $\boldsymbol{V}_m$ for the recurrence. However, $\boldsymbol{y}_m$ and $\boldsymbol{y}_{m-1}$ may differ in each coefficient, so that we still need all the columns of $\boldsymbol{V}_m$ and $\boldsymbol{U}\boldsymbol{B}_m$ to update $\boldsymbol{u}_m$. Let $\{\boldsymbol{v}_i\}$ and $\{\hat{\boldsymbol{b}}_i\}$ be the columns of $\boldsymbol{V}_m$ and $\boldsymbol{U}\boldsymbol{B}_m$ respectively. To be able to discard the old $\boldsymbol{v}_i$ and $\hat{\boldsymbol{b}}_i$ vectors we use the same transformations as in MINRES. Let

$$\widehat{\boldsymbol{B}}_m = \boldsymbol{U}\boldsymbol{B}_m, \quad \widetilde{\boldsymbol{B}}_m = \widehat{\boldsymbol{B}}_m \boldsymbol{F}_m^{-1}, \quad \widetilde{\boldsymbol{V}}_m = \boldsymbol{V}_m \boldsymbol{F}_m^{-1}, \quad \tilde{\boldsymbol{y}}_m = \boldsymbol{F}_m \boldsymbol{y}_m. \tag{4.21}$$

Then

$$\tilde{\boldsymbol{y}}_m = \widehat{\boldsymbol{G}}_m \beta \boldsymbol{e}_1 = \boldsymbol{G}_m \boldsymbol{G}_{m-1} \cdots \boldsymbol{G}_1 \beta \boldsymbol{e}_1 = \boldsymbol{G}_m \tilde{\boldsymbol{y}}_{m-1}, \tag{4.22}$$

and only the mth and $(m+1)$th coefficients of $\tilde{\boldsymbol{y}}_{m-1}$ and $\tilde{\boldsymbol{y}}_m$ differ. The update (4.20) becomes

$$\begin{aligned} \boldsymbol{x}_m &= \hat{\boldsymbol{x}}_0 - \widetilde{\boldsymbol{B}}_m \tilde{\boldsymbol{y}}_m + \widetilde{\boldsymbol{V}}_m \tilde{\boldsymbol{y}}_m \\ &= \hat{\boldsymbol{x}}_0 - (\widetilde{\boldsymbol{B}}_{m-1} \tilde{\boldsymbol{y}}_{m-1} + \tilde{\boldsymbol{b}}_m \tilde{y}_{m,m}) + (\widetilde{\boldsymbol{V}}_{m-1} \tilde{\boldsymbol{y}}_{m-1} + \tilde{\boldsymbol{v}}_m \tilde{y}_{m,m}) \\ &= \boldsymbol{x}_{m-1} - \tilde{\boldsymbol{b}}_m \tilde{y}_{m,m} + \tilde{\boldsymbol{v}}_m \tilde{y}_{m,m}, \end{aligned} \tag{4.23}$$

where $\tilde{\boldsymbol{b}}_m$ and $\tilde{\boldsymbol{v}}_m$ are the mth columns of $\widetilde{\boldsymbol{B}}_m$ and $\widetilde{\boldsymbol{V}}_m$ respectively, and $\tilde{y}_{m,m}$ is the mth coefficient of vector $\tilde{\boldsymbol{y}}_m$. Therefore, we only need the last column of $\widetilde{\boldsymbol{B}}_m$ and $\widetilde{\boldsymbol{V}}_m$ to update $\boldsymbol{u}$. From the definition of $\widetilde{\boldsymbol{B}}_m$ and $\widetilde{\boldsymbol{V}}_m$ in (4.21), we have

$$\tilde{\boldsymbol{b}}_{m-2} f_{m-2,m} + \tilde{\boldsymbol{b}}_{m-1} f_{m-1,m} + \tilde{\boldsymbol{b}}_m f_{m,m} = \hat{\boldsymbol{b}}_m, \tag{4.24}$$

$$\tilde{\boldsymbol{v}}_{m-2} f_{m-2,m} + \tilde{\boldsymbol{v}}_{m-1} f_{m-1,m} + \tilde{\boldsymbol{v}}_m f_{m,m} = \boldsymbol{v}_m, \tag{4.25}$$

so that the columns of $\widetilde{\boldsymbol{B}}_m$ and $\widetilde{\boldsymbol{V}}_m$ can be computed by three-term recurrences as well.

Algorithm 4.3 outlines the modified MINRES that includes the recycle space into the search space. In this algorithm, because of the three-term recurrences, we do not need to restart. So, in exact arithmetic, there is no need to use the recycle space generated during the solution of a linear system in the solution of that same system[1]. As a consequence, we can derive a more efficient method for recycling for symmetric matrices. Although the Lanczos recurrence requires only the latest two basis vectors from the Krylov subspace (Lanczos vectors) for orthogonalization and restarting is not necessary, we do need all the Lanczos vectors to compute a recycle space. Therefore, to limit the memory requirements, we update

[1]In floating point arithmetic, including the recycle space obtained from the current Krylov subspace may help remedy the loss of orthogonality that generally occurs due to rounding errors.

Algorithm 4.3: MODIFIED MINRES

$\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$;
$\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_0 + \boldsymbol{U}\boldsymbol{C}^T\boldsymbol{r}_0$; $\quad \boldsymbol{r}_0 \leftarrow \boldsymbol{r}_0 - \boldsymbol{C}\boldsymbol{C}^T\boldsymbol{r}_0$;
$\boldsymbol{v}_1 \leftarrow \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$; $\quad \tilde{\boldsymbol{y}} \leftarrow \|\boldsymbol{r}_0\|\boldsymbol{e}_1$;
for $m = 1, \cdots$ **do**
  $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{A}\boldsymbol{v}_m$;
  /* use modified Gram-Schmidt orthogonalization for updating $\hat{\boldsymbol{v}}$ */
  $\hat{\boldsymbol{v}} \leftarrow \hat{\boldsymbol{v}} - \boldsymbol{C}(\boldsymbol{C}^T\hat{\boldsymbol{v}})$; $\quad \hat{\boldsymbol{b}}_m \leftarrow \boldsymbol{U}(\boldsymbol{C}^T\hat{\boldsymbol{v}})$;
  $t_{m-1,m} \leftarrow t_{m,m-1}$; $\quad \hat{\boldsymbol{v}} \leftarrow \hat{\boldsymbol{v}} - t_{m-1,m}\hat{\boldsymbol{v}}_{m-1}$;
  $t_{m,m} \leftarrow \langle \hat{\boldsymbol{v}}, \boldsymbol{v}_m \rangle$; $\quad \hat{\boldsymbol{v}} \leftarrow \hat{\boldsymbol{v}} - t_{m,m}\boldsymbol{v}_m$;
  $t_{m+1,m} \leftarrow \|\hat{\boldsymbol{v}}\|$; $\quad \boldsymbol{v}_{m+1} \leftarrow \hat{\boldsymbol{v}}/t_{m+1,m}$;
  $\boldsymbol{F}_{:,m} \leftarrow \boldsymbol{G}_{m-1}\boldsymbol{G}_{m-2}\underline{\boldsymbol{T}}_{:,m}$; /* apply the Givens rotations from the previous two iterations to the new column of $\underline{\boldsymbol{T}}_m$ */
  Compute Givens rotation $\boldsymbol{G}_m$ such that $\boldsymbol{F}_{:,m} \leftarrow \boldsymbol{G}_m\boldsymbol{F}_{:,m}$ has a zero coefficient at position $(m+1, m)$; /* see MINRES [33, p. 41–44] */
  $\tilde{\boldsymbol{y}} \leftarrow \boldsymbol{G}_m\tilde{\boldsymbol{y}}$;
  $\tilde{\boldsymbol{v}}_m \leftarrow f_{m,m}^{-1}(\boldsymbol{v}_m - \tilde{\boldsymbol{v}}_{m-1}f_{m-1,m} - \tilde{\boldsymbol{v}}_{m-2}f_{m-2,m})$;
  $\tilde{\boldsymbol{b}}_m \leftarrow f_{m,m}^{-1}(\hat{\boldsymbol{b}}_m - \tilde{\boldsymbol{b}}_{m-1}f_{m-1,m} - \tilde{\boldsymbol{b}}_{m-2}f_{m-2,m})$;
  $\boldsymbol{x}_m \leftarrow \boldsymbol{x}_{m-1} + \tilde{\boldsymbol{v}}_m\tilde{y}_m - \tilde{\boldsymbol{b}}_m\tilde{y}_m$; /* $\tilde{y}_m$ is the mth entry of vector $\tilde{\boldsymbol{y}}$ */
end

the selected recycle space periodically. In this case, a *cycle* refers to the iterations between two updates of the recycle space. We keep using the recycle space from the last system for orthogonalization, and compute and update another recycle space for the next system.

We use s to denote the maximum length of a cycle (and hence the maximum number of Lanczos vectors kept), and k to denote the number of linearly independent vectors selected for recycling. We use RMINRES(s, k) to indicate the recycling MINRES method with the parameters s and k. The matrix $\boldsymbol{V}_j$ contains the Lanczos vectors generated in the jth cycle, $\boldsymbol{V}_j = [\boldsymbol{v}_{(j-1)s+1}, \cdots, \boldsymbol{v}_{js}]$, and the matrix $\overline{\boldsymbol{V}}_j = [\boldsymbol{v}_{(j-1)s}, \cdots, \boldsymbol{v}_{js+1}]$ denotes $\boldsymbol{V}_j$ extended with with one previous and one subsequent Lanczos vector. Then, for the jth cycle, the modified Lanczos process gives

$$(\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^T)\boldsymbol{A}\boldsymbol{V}_j = \overline{\boldsymbol{V}}_j\overline{\boldsymbol{T}}_j, \tag{4.26}$$

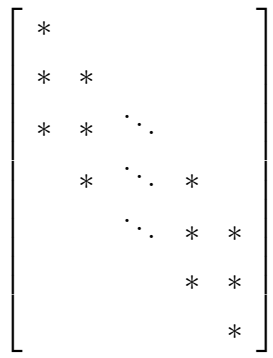


Figure 4.1: Nonzero pattern of $\overline{\underline{\boldsymbol{T}}}_j$.

where $\overline{\underline{\boldsymbol{T}}}_j$ is the tridiagonal matrix $\underline{\boldsymbol{T}}_j$ with an additional row corresponding to $\boldsymbol{v}_{(j-1)s}$ at the top. The bottom row corresponds to $\boldsymbol{v}_{js+1}$. To be specific, $\overline{\underline{\boldsymbol{T}}}_j$ has the nonzero pattern shown in Figure 4.1.

Let $\boldsymbol{U}_{j-1}$ give the basis of a subspace that was selected at the end of cycle $j-1$ for the current linear system. $\boldsymbol{U}_{j-1}$ is used only to compute $\boldsymbol{U}_j$ after cycle j; it is not used in solving the current linear system. The final $\boldsymbol{U}_j$ will be used for the next linear system. Below, we discuss several options to compute $\boldsymbol{U}_j$ from $\boldsymbol{U}$, $\boldsymbol{U}_{j-1}$, and the matrix $\boldsymbol{V}_j$ containing the Lanczos vectors generated in the latest cycle for the current system.

The modified Lanczos recurrence that includes the orthogonalization against $\boldsymbol{C}$ gives

$$\boldsymbol{A}[\boldsymbol{U}\ \boldsymbol{U}_{j-1}\ \boldsymbol{V}_j] = [\boldsymbol{C}\ \boldsymbol{C}_{j-1}\ \overline{\underline{\boldsymbol{V}}}_j] \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{B}_j \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \overline{\underline{\boldsymbol{T}}}_j \end{bmatrix}, \tag{4.27}$$

where $\boldsymbol{B}_j = \boldsymbol{C}^T \boldsymbol{A} \boldsymbol{V}_j$ has been computed in the course of the iteration (see (4.26)).

Now, we have several options for selecting a new matrix $\boldsymbol{U}_j$ for recycling. The first option is to compute the harmonic Ritz vectors of $\boldsymbol{A}$ with respect to range $([\boldsymbol{U}\ \boldsymbol{U}_{j-1}\ \boldsymbol{V}_j])$. We must include range $(\boldsymbol{U})$, because it is recycled from the last system and typically contains some of the approximate eigenvectors we need. Now, all the Lanczos vectors are orthogonal to $\boldsymbol{C}$.

Leaving range $(\boldsymbol{U})$ out, we may lose those important approximate eigenvectors permanently in the recycle space for the next system. However, including range $(\boldsymbol{U})$ for every cycle is inefficient. So, as the second option, we only include range $(\boldsymbol{U})$ for the last cycle. The third option is to obtain $\boldsymbol{U}_1$ from range $([\boldsymbol{U}\ \boldsymbol{V}_1])$ for the first cycle and $\boldsymbol{U}_j$ from range $([\boldsymbol{U}_{j-1}\ \boldsymbol{V}_j])$ for the jth cycle. In the last approach, the reappearance of $\boldsymbol{U}$ can be avoided as well. If we think of $\boldsymbol{U}$ as $\boldsymbol{U}_0$, the third approach leads to more consistent formulation. So, we only discuss the third option here.

Let

$$\boldsymbol{W}_j = [\boldsymbol{U}_{j-1}\ \boldsymbol{V}_j], \qquad \widetilde{\boldsymbol{W}}_j = [\boldsymbol{C}\ \boldsymbol{C}_{j-1}\ \overline{\underline{\boldsymbol{V}}}_j], \qquad \widetilde{\underline{\boldsymbol{H}}}_j = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{B}_j \\ \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \overline{\underline{\boldsymbol{T}}}_j \end{bmatrix}. \tag{4.28}$$

Then, (4.27) gives

$$\boldsymbol{A}\boldsymbol{W}_j = \widetilde{\boldsymbol{W}}_j \widetilde{\underline{\boldsymbol{H}}}_j. \tag{4.29}$$

Now, we compute the harmonic Ritz values and vectors of $\boldsymbol{A}$ with respect to the subspace range $(\boldsymbol{W}_j)$. These harmonic Ritz pairs $(\theta, \boldsymbol{w})$ are defined by the condition

$$\boldsymbol{A}\boldsymbol{w} - \theta\boldsymbol{w} \perp \text{range}\,(\boldsymbol{A}\boldsymbol{W}_j), \tag{4.30}$$

where $\boldsymbol{w} \in \text{range}\,(\boldsymbol{W}_j)$. If we write $\boldsymbol{w} = \boldsymbol{W}_j\boldsymbol{p}$, computing harmonic Ritz pairs is equivalent to solving the generalized eigenvalue problem

$$\widetilde{\underline{\boldsymbol{H}}}_j^T \widetilde{\boldsymbol{W}}_j^T \widetilde{\boldsymbol{W}}_j \widetilde{\underline{\boldsymbol{H}}}_j \boldsymbol{p} = \theta \widetilde{\underline{\boldsymbol{H}}}_j^T \widetilde{\boldsymbol{W}}_j^T \boldsymbol{W}_j \boldsymbol{p}. \tag{4.31}$$

We discuss the construction of this generalized eigenvalue problem later, and assume we get the solution of it for now. We choose the k harmonic Ritz vectors with the (absolute)

smallest harmonic Ritz values for recycling, and set $\widetilde{\boldsymbol{U}}_j = \boldsymbol{W}_j \boldsymbol{P}_j$, where the columns of $\boldsymbol{P}_j$ are the chosen eigenvectors for (4.31). Now we have $\widetilde{\boldsymbol{C}}_j = \boldsymbol{A}\widetilde{\boldsymbol{U}}_j = \widetilde{\boldsymbol{W}}_j \underline{\widetilde{\boldsymbol{H}}}_j \boldsymbol{P}_j$. To obtain $\boldsymbol{C}_j$ with orthonormal columns, we compute the QR decomposition of $\widetilde{\boldsymbol{W}}_j$ (note that by construction almost all columns are already orthogonal),

$$\widetilde{\boldsymbol{W}}_j = \widehat{\boldsymbol{W}}_j \boldsymbol{K}_j, \tag{4.32}$$

and of $\boldsymbol{K}_j \underline{\widetilde{\boldsymbol{H}}}_j \boldsymbol{P}_j$,

$$\boldsymbol{K}_j \underline{\widetilde{\boldsymbol{H}}}_j \boldsymbol{P}_j = \boldsymbol{Q}_j \boldsymbol{R}_j. \tag{4.33}$$

Next, we set

$$\boldsymbol{U}_j = \boldsymbol{W}_j \widehat{\boldsymbol{P}}_j, \qquad \boldsymbol{C}_j = \widehat{\boldsymbol{W}}_j \boldsymbol{Q}_j = \widetilde{\boldsymbol{W}}_j \widehat{\boldsymbol{Q}}_j, \tag{4.34}$$

where $\widehat{\boldsymbol{P}}_j = \boldsymbol{P}_j \boldsymbol{R}_j^{-1}$, and $\widehat{\boldsymbol{Q}}_j = \boldsymbol{K}_j^{-1} \boldsymbol{Q}_j$. Then $\boldsymbol{C}_j$ is orthogonal and $\boldsymbol{A}\boldsymbol{U}_j = \boldsymbol{C}_j$. The two QR decompositions (4.32–4.33) are cheap to compute because $\boldsymbol{K}_j$ has very few nonzeros, whose positions are known in advance, and $\boldsymbol{K}_j \underline{\widetilde{\boldsymbol{H}}}_j \boldsymbol{P}_j$ is a product of matrices of small dimensions.

Finally, to solve the generalized eigenvalue problem (4.31), we need the matrices $\underline{\widetilde{\boldsymbol{H}}}_j^T \widetilde{\boldsymbol{W}}_j^T \widetilde{\boldsymbol{W}}_j \underline{\widetilde{\boldsymbol{H}}}_j$ and $\underline{\widetilde{\boldsymbol{H}}}_j^T \widetilde{\boldsymbol{W}}_j^T \boldsymbol{W}_j$. We can simplify $\widetilde{\boldsymbol{W}}_j^T \widetilde{\boldsymbol{W}}_j$ and $\widetilde{\boldsymbol{W}}_j^T \boldsymbol{W}_j$ as follows:

$$\widetilde{\boldsymbol{W}}_j^T \widetilde{\boldsymbol{W}}_j = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{C}^T \boldsymbol{C}_{j-1} & \boldsymbol{0} \\ \boldsymbol{C}_{j-1}^T \boldsymbol{C} & \boldsymbol{I} & \boldsymbol{C}_{j-1}^T \underline{\overline{\boldsymbol{V}}}_j \\ \boldsymbol{0} & \underline{\overline{\boldsymbol{V}}}_j^T \boldsymbol{C}_{j-1} & \boldsymbol{I} \end{bmatrix}, \tag{4.35}$$

$$\widetilde{\boldsymbol{W}}_j^T \boldsymbol{W}_j = \begin{bmatrix} \boldsymbol{C}^T \boldsymbol{U}_{j-1} & \boldsymbol{0} \\ \boldsymbol{C}_{j-1}^T \boldsymbol{U}_{j-1} & \boldsymbol{C}_{j-1}^T \boldsymbol{V}_j \\ \underline{\overline{\boldsymbol{V}}}_j^T \boldsymbol{U}_{j-1} & \underline{\overline{\boldsymbol{I}}} \end{bmatrix}, \tag{4.36}$$

where $\underline{\overline{\boldsymbol{I}}}$ is an extended identity matrix with an additional row of zeros at the top and at the bottom. We can simplify the computation of most blocks in these two matrices further.

$$\boldsymbol{C}^T\boldsymbol{C}_{j-1} = \boldsymbol{C}^T\widetilde{\boldsymbol{W}}_{j-1}\widehat{\boldsymbol{Q}}_{j-1} = [\boldsymbol{I} \;\; \boldsymbol{C}^T\boldsymbol{C}_{j-2} \;\; \boldsymbol{0}]\widehat{\boldsymbol{Q}}_{j-1}, \tag{4.37}$$

$$\underline{\overline{\boldsymbol{V}}}_j^T\boldsymbol{C}_{j-1} = \underline{\overline{\boldsymbol{V}}}_j^T\widetilde{\boldsymbol{W}}_{j-1}\widehat{\boldsymbol{Q}}_{j-1} = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & 0 & 0 & 1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix}\widehat{\boldsymbol{Q}}_{j-1}, \tag{4.38}$$

$$\boldsymbol{C}^T\boldsymbol{U}_{j-1} = \boldsymbol{C}^T\boldsymbol{W}_{j-1}\widehat{\boldsymbol{P}}_{j-1} = [\boldsymbol{C}^T\boldsymbol{U}_{j-2} \;\; \boldsymbol{0}]\widehat{\boldsymbol{P}}_{j-1}, \tag{4.39}$$

$$\boldsymbol{C}_{j-1}^T\boldsymbol{U}_{j-1} = \widehat{\boldsymbol{Q}}_{j-1}^T(\widetilde{\boldsymbol{W}}_{j-1}^T\boldsymbol{W}_{j-1})\widehat{\boldsymbol{P}}_{j-1}, \tag{4.40}$$

$$\boldsymbol{C}_{j-1}^T\boldsymbol{V}_j = \widehat{\boldsymbol{Q}}_{j-1}^T\begin{bmatrix} \boldsymbol{C}^T \\ \boldsymbol{C}_{j-2}^T \\ \underline{\overline{\boldsymbol{V}}}_{j-1} \end{bmatrix}\boldsymbol{V}_j = \widehat{\boldsymbol{Q}}_{j-1}^T\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{4.41}$$

From the above derivation, we can obtain $\underline{\overline{\boldsymbol{V}}}_j^T\boldsymbol{C}_{j-1}$ and $\boldsymbol{C}_{j-1}^T\boldsymbol{V}_j$ simply from $\widehat{\boldsymbol{Q}}_{j-1}$ (known from the previous cycle), and we can compute $\boldsymbol{C}^T\boldsymbol{C}_{j-1}$, $\boldsymbol{C}^T\boldsymbol{U}_{j-1}$ and $\boldsymbol{C}_{j-1}^T\boldsymbol{U}_{j-1}$ recursively with $2k^3$, $2k^3$ and $2k(k+s)(3k+s)$ flops, respectively. Therefore, the computation of these submatrices is very cheap. Only $\underline{\overline{\boldsymbol{V}}}_j^T\boldsymbol{U}_{j-1}$ must be computed explicitly by matrix-matrix product, which takes about $2ksn$ flops. In summary, the cost of each update of the recycle space is about $(12k^2 + 6ks + 6k + 4)n$ flops, ignoring the terms less than $O(n)$. Compared with the cost of MINRES, which is mainly determined by the matrix-vector product and the preconditioner for each iteration, the overhead of the subspace selection is modest (see the timing results in Section 4.5).

The general form of the RMINRES is outlined in Algorithms 4.4 and 4.5. For brevity, in the algorithms we do not explicitly write out the slight changes for the first linear system in the sequence, when we do not have a recycle space $\widehat{\boldsymbol{U}}$ yet; and for the first cycle for each

linear system, i.e., $j = 1$, when we do not have $\boldsymbol{U}_{j-1}$ and $\boldsymbol{C}_{j-1}$ yet. For the first system in the sequence, we define

$$\begin{aligned} \boldsymbol{W}_1 &= \boldsymbol{V}_1, \qquad \widetilde{\boldsymbol{W}}_1 = \underline{\boldsymbol{V}}_1, \\ \boldsymbol{W}_j &= [\boldsymbol{U}_{j-1}\ \boldsymbol{V}_j], \qquad \widetilde{\boldsymbol{W}}_j = [\boldsymbol{C}_{j-1}\ \overline{\underline{\boldsymbol{V}}}_j], \quad \text{for } j > 1. \end{aligned} \tag{4.42}$$

And for the first cycle of each subsequent system, we let

$$\boldsymbol{W}_1 = \boldsymbol{V}_1, \quad \widetilde{\boldsymbol{W}}_1 = [\boldsymbol{C}\ \underline{\boldsymbol{V}}_1]. \tag{4.43}$$

The $\boldsymbol{U}_j$ and $\boldsymbol{C}_j$ for these special cases can be easily derived using the simplified definitions of $\boldsymbol{W}$ and $\widetilde{\boldsymbol{W}}$ in (4.42) and (4.43) following the approach that we describe for the general case, and Algorithm 4.5 can be modified correspondingly.

4.4 Implementation Issues

For the experiments in the next section, we have developed our C/C++ code including 3D topology optimization and recycling MINRES. We store sparse matrices in compressed sparse row format (CSR). The (column) vectors in $\boldsymbol{U}$, $\boldsymbol{C}$, $\boldsymbol{V}_j$, and so on, are stored as one-dimensional arrays linked by a one-dimensional array of pointers. The memory required by the system matrix and the incomplete Cholesky factor is linear in the number of unknowns, n, since the number of nonzero coefficients per row is never greater than 81, for our choice of elements and mesh. The RMINRES method requires only matrix-vector multiplications, dot products, vector updates, forward and backward solves with the incomplete Cholesky factors, and the incomplete Cholesky decomposition itself. All of these operations have linear computational cost.

The small matrices, e.g., the matrices in (4.33), are all stored as dense matrices in column-wise ordering (F77 format), so that dense matrix routines from `LAPACK` and `BLAS`

Algorithm 4.4: RECYCLING MINRES: solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

Input: initial guess $\boldsymbol{x}_0$ and recycle space $\widehat{\boldsymbol{U}}$ obtained from the previous system, and parameters s and k

$\widehat{\boldsymbol{C}} \leftarrow \boldsymbol{A}\widehat{\boldsymbol{U}}$;

compute the QR decomposition of $\widehat{\boldsymbol{C}}$ as $\widehat{\boldsymbol{C}} = \boldsymbol{C}\widetilde{\boldsymbol{R}}$;

$\boldsymbol{U} \leftarrow \widehat{\boldsymbol{U}}\widetilde{\boldsymbol{R}}^{-1}$; /* using backward substitution */

$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$; $\boldsymbol{r} \leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$;

$\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{U}(\boldsymbol{C}^T\boldsymbol{r})$; $\boldsymbol{r} \leftarrow \boldsymbol{r} - \boldsymbol{C}(\boldsymbol{C}^T\boldsymbol{r})$; /* use Modified Gram-Schmidt */

$\gamma = \|\boldsymbol{r}\|$; $\boldsymbol{v}_1 \leftarrow \boldsymbol{r}/\gamma$; $\tilde{\boldsymbol{y}} \leftarrow \gamma\boldsymbol{e}_1$;

$m \leftarrow 0$; $j \leftarrow 0$;

while $\gamma/\|\boldsymbol{b}\| > \text{tol}$ **do**

  $m \leftarrow m + 1$;

  $\hat{\boldsymbol{v}} = \boldsymbol{A}\boldsymbol{v}_m$;

  $\hat{\boldsymbol{v}} \leftarrow \hat{\boldsymbol{v}} - \boldsymbol{C}(\boldsymbol{C}^T\hat{\boldsymbol{v}})$; $\hat{\boldsymbol{b}} \leftarrow \boldsymbol{U}(\boldsymbol{C}^T\hat{\boldsymbol{v}})$; /* use Modified Gram-Schmidt */

  $t_{m-1,m} \leftarrow t_{m,m-1}$; $\hat{\boldsymbol{v}} \leftarrow \hat{\boldsymbol{v}} - t_{m-1,m}\hat{\boldsymbol{v}}_{m-1}$;

  $t_{m,m} \leftarrow \langle \hat{\boldsymbol{v}}, \boldsymbol{v}_m \rangle$; $\hat{\boldsymbol{v}} \leftarrow \hat{\boldsymbol{v}} - t_{m,m}\boldsymbol{v}_m$;

  $t_{m+1,m} \leftarrow \|\hat{\boldsymbol{v}}\|$; $\boldsymbol{v}_{m+1} \leftarrow \hat{\boldsymbol{v}}/t_{m+1,m}$;

  $\boldsymbol{F}_{:,m} \leftarrow \boldsymbol{G}_{m-1}\boldsymbol{G}_{m-2}\underline{\boldsymbol{T}}_{:,m}$;

  compute Givens rotation $\boldsymbol{G}_m$ such that $\boldsymbol{F}_{:,m} \leftarrow \boldsymbol{G}_m\boldsymbol{F}_{:,m}$ has a zero coefficient at position $(m+1, m)$;

  $\tilde{\boldsymbol{y}} \leftarrow \boldsymbol{G}_m\tilde{\boldsymbol{y}}$;

  $\tilde{\boldsymbol{v}}_m \leftarrow f_{m,m}^{-1}(\boldsymbol{v}_m - \tilde{\boldsymbol{v}}_{m-1}f_{m-1,m} - \tilde{\boldsymbol{v}}_{m-2}f_{m-2,m})$; then drop $\tilde{\boldsymbol{v}}_{m-2}$;

  $\tilde{\boldsymbol{b}}_m \leftarrow f_{m,m}^{-1}(\hat{\boldsymbol{b}} - \tilde{\boldsymbol{b}}_{m-1}f_{m-1,m} - \tilde{\boldsymbol{b}}_{m-2}f_{m-2,m})$; then drop $\tilde{\boldsymbol{b}}_{m-2}$;

  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \tilde{\boldsymbol{v}}_m\tilde{y}_m - \tilde{\boldsymbol{b}}_m\tilde{y}_m$; $\gamma \leftarrow \tilde{y}_{m+1}$; /* $\tilde{y}_m$ and $\tilde{y}_{m+1}$ are the mth and $(m+1)$th entries of vector $\tilde{\boldsymbol{y}}$, so $\gamma = \|\boldsymbol{r}\|$ */

  if $(\gamma/\|\boldsymbol{b}\| \le \text{tol})$ *or* $(\text{mod}(m, s) = 0)$ **then**

    $j \leftarrow j + 1$;

    compute $\boldsymbol{U}_j$ and $\boldsymbol{C}_j$ of dimension k following Algorithm 4.5 ;

    drop $\boldsymbol{U}_{j-1}$, $\boldsymbol{C}_{j-1}$, and all $\boldsymbol{v}$ vectors except $\boldsymbol{v}_{m+1}$ and $\boldsymbol{v}_m$;

  end

end

Output: $\boldsymbol{x}$ as the solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and as the initial guess $\boldsymbol{x}_0$ for the next system, and $\boldsymbol{U}_j$ as the $\widehat{\boldsymbol{U}}$ for the next system

Algorithm 4.5: RECYCLE SPACE SELECTION: compute $\boldsymbol{U}_j$ and $\boldsymbol{C}_j$

1 compute $\boldsymbol{C}^T\boldsymbol{C}_{j-1}$ following (4.37) with $\boldsymbol{C}^T\boldsymbol{C}_{j-2}$ and $\widehat{\boldsymbol{Q}}_{j-1}$ already available from the $(j-1)$th cycle ;

2 compute $\underline{\overline{\boldsymbol{V}}}_j^T\boldsymbol{C}_{j-1}$ following (4.38) with $\widehat{\boldsymbol{Q}}_{j-1}$ already available from the $(j-1)$th cycle ; /* Copy the last two rows of $\widehat{\boldsymbol{Q}}_{j-1}$ */

3 compute $\boldsymbol{C}^T\boldsymbol{U}_{j-1}$ following (4.39) with $\boldsymbol{C}^T\boldsymbol{U}_{j-2}$ and $\widehat{\boldsymbol{P}}_{j-1}$ already available from the $(j-1)$th cycle ;

4 compute $\boldsymbol{C}_{j-1}^T\boldsymbol{U}_{j-1}$ following (4.40) with $\widehat{\boldsymbol{Q}}_{j-1}$, $\widetilde{\boldsymbol{W}}_{j-1}^T\boldsymbol{W}_{j-1}$ and $\widehat{\boldsymbol{P}}_{j-1}$ already available from the $(j-1)$th cycle ;

5 compute $\boldsymbol{C}_{j-1}^T\boldsymbol{V}_j$ following (4.41) with $\widehat{\boldsymbol{Q}}_{j-1}$ already available from the $(j-1)$th cycle ; /* Copy the last column of $\widehat{\boldsymbol{Q}}_{j-1}^T$ */

6 compute $\underline{\overline{\boldsymbol{V}}}_j^T\boldsymbol{U}_{j-1}$ by matrix-matrix product ;

7 assemble $\widetilde{\boldsymbol{W}}_j^T\widetilde{\boldsymbol{W}}_j$ and $\widetilde{\boldsymbol{W}}_j^T\boldsymbol{W}_j$ following (4.35) and (4.36) ;

8 solve the generalized eigenvalue problem (4.31) and pick the k generalized eigenvectors corresponding to the k smallest eigenvalues to form the columns of $\boldsymbol{P}_j$;

9 compute the QR decomposition of $\widetilde{\boldsymbol{W}}_j$ as $\widetilde{\boldsymbol{W}}_j = \widehat{\boldsymbol{W}}_j\boldsymbol{K}_j$; /* Orthogonalize the first two columns of $\underline{\overline{\boldsymbol{V}}}_j$ against $\boldsymbol{C}_{j-1}$ */

10 compute the QR decomposition of $\boldsymbol{K}_j\underline{\widetilde{\boldsymbol{H}}}_j\boldsymbol{P}_j$ as $\boldsymbol{K}_j\underline{\widetilde{\boldsymbol{H}}}_j\boldsymbol{P}_j = \boldsymbol{Q}_j\boldsymbol{R}_j$;

11 $\widehat{\boldsymbol{P}}_j \leftarrow \boldsymbol{P}_j\boldsymbol{R}_j^{-1}$; $\widehat{\boldsymbol{Q}}_j \leftarrow \boldsymbol{K}_j^{-1}\boldsymbol{Q}_j$;

12 $\boldsymbol{U}_j \leftarrow \boldsymbol{W}_j\widehat{\boldsymbol{P}}_j$; $\boldsymbol{C}_j \leftarrow \widetilde{\boldsymbol{W}}_j\widehat{\boldsymbol{Q}}_j$;

can be used. For the generalized eigenvalue problem (4.31) we use the `LAPACK` routine DSYGV, and for QR decompositions we use the `LAPACK` routine DGEQRF. We use the `BLAS` routines DROTG, to compute Givens rotations, and DROT, to apply Givens rotations. We use the `BLAS` routine DGEMM for (small) dense matrix-matrix products. For convenience we use the `CLAPACK` library, which provides an interface for C programs to `LAPACK`. However, since `CLAPACK` routines call the corresponding `LAPACK` routines, we still need to adhere to F77 storage formats. The computational cost of the work with these small matrices is negligible. Finally, we note that the computational cost is significantly reduced by taking the simplifications in (4.35)–(4.41) into account.

4.5 Numerical Experiments with Topology Optimization Problems

We demonstrate the performance of our RMINRES method on a 3D design problem shown in Figure 4.2. We compute the optimal design for a 3D beam in a hexahedron with the left end fixed and a distributed load applied on the right bottom edge. The scale of the hexahedron is $X : Y : Z = 3 : 1 : 1$. The volume fraction is 50%, and the radius of the filter is $Y/10$.

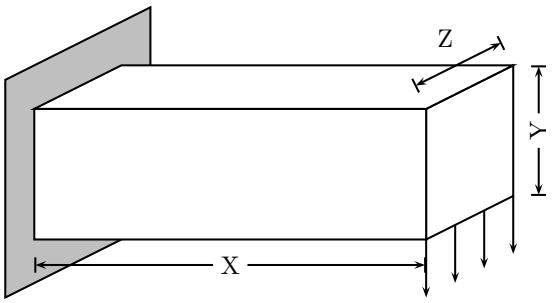


Figure 4.2: Design problem: finding optimal material distribution in a hexahedron with the left end fixed and a distributed load applied on the right bottom edge. ($X : Y : Z = 3 : 1 : 1$).

We use continuation on the density penalization, ranging from 1 to 3 with increments

Table 4.1: Three discretizations used for the example in Figure 4.2.

problem	mesh size	#unknowns (in simulation)	solution time	optimization steps
small	$36 \times 12 \times 6$	$9,360$	0.1h	142
medium	$84 \times 28 \times 14$	$107,184$	2.4h	139
large	$180 \times 60 \times 30$	$1,010,160$	45.7h	130

of 0.5. We use the OC method as the optimization algorithm. The convergence criterion is that either the maximum change in the design variables is less than 0.01 or the relative change of the compliance is less than 10^{-6}. For all the iterative solvers discussed, we always use the solution of the previous system as the initial guess of the next system to reduce the initial error. Exploiting the symmetry of the problem, we model and simulate only half of the domain.

We test three discretizations of increasing mesh resolution. Exploiting the symmetry of the problem, we model and simulate only half of the domain. For each test case, Table 4.1 lists the mesh size (for half of the domain), the number of unknowns, the overall solution time, the number of optimization steps, and the parameters used for the recycling MINRES solver. The timings are obtained on a PC with an AMD Opteron$^{\text{TM}}$252 2.6GHz 64-bit processor, 8GB RAM of memory, and the SuSE Linux system. Figure 4.3 shows the final topologies.

4.5.1 Recycling Results and Discussion

First, we analyze the convergence properties of RMINRES for several parameter choices on the medium size ($84 \times 28 \times 14$) mesh. The number of optimization steps to compute the optimal design is 139, requiring the solution of 139 linear systems.

As mentioned in the first section, we can vary the tolerance for the iterative solver, since less accurate finite element solutions are sufficient at the beginning of the topology optimization process. So, we can also apply a continuation approach to the tolerance of the

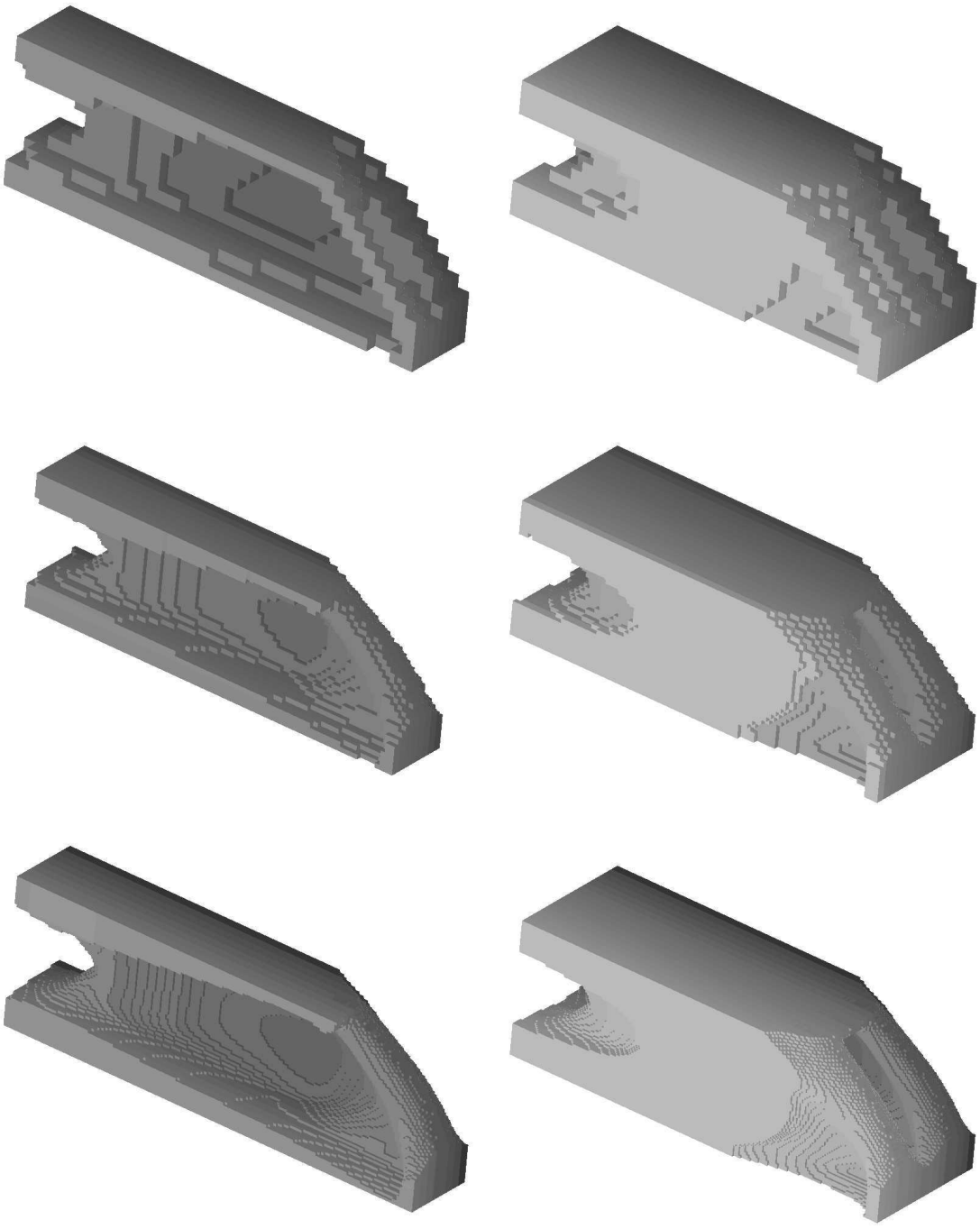


Figure 4.3: Final topologies for the problem shown in Figure 4.2 on different uniform meshes. Left: half domain; right: full domain. Top row: small mesh; middle row: medium size mesh; bottom row: large mesh.

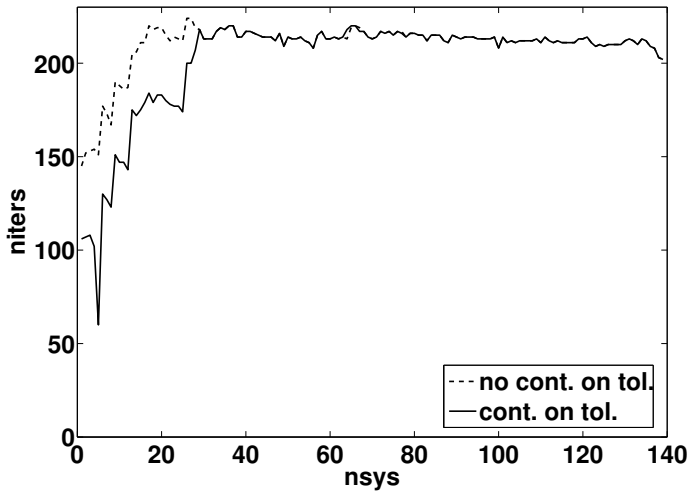


Figure 4.4: Reduction in the number of iterations using a relaxed tolerance for the linear solver (MINRES without recycling). The jumps in the iteration counts over the first 30 iterations are caused by continuation on the solver tolerance and the penalization parameter.

linear solver, which reduces the number of iterations in the early phase of the optimization process, as shown in Figure 4.4. The jumps in the iteration counts correspond to the steps where the tolerance of the linear solver τ is decreased or the penalization parameter p is increased. We start with $\tau = 10^{-4}$ and $p = 1$; we decrease τ by a factor of $1/10$ and increase p by 0.5 every time the maximum change of the design variables drops below 0.1; and we stop updating them when $\tau = 10^{-10}$ and $p = 3$. Finally, we note that allowing a higher tolerance for the linear solver in the beginning of the optimization process did not affect the number of optimization steps required.

Next, we consider the parameters that govern the recycling for the MINRES solver, namely k, the dimension of the subspace that is recycled from one linear system to the next, and s, the maximum dimension of the Krylov subspace kept to periodically update the approximate invariant subspace that will be recycled. We carry out two sets of experiments to analyze the effects of varying these two parameters. To make a fair comparison, we use the solution from the previous system as the initial guess of the next system and we use

continuation on the tolerance for both RMINRES and MINRES.

In the first set of experiments, we fix $k = 10$ and vary s. Figure 4.5 compares the number of iterations and computation time for each linear system, for several choices of s. In the first few optimization steps, the topology changes significantly, and the effect of recycling is modest. After this, the recycling approach greatly reduces the number of iterations to solve each linear system. We see that if we keep a larger Krylov subspace to update the approximate invariant subspace, the recycling becomes more effective in reducing iteration counts. Since the dimension of the recycle space itself does not change, this suggests that we obtain a more accurate approximation to the invariant subspace this way. This reduction in iterations significantly reduces the computation time for RMINRES, in spite of the computational overhead from the orthogonalizations against the recycle space and from the updates of the recycle space. Towards the end of the optimization process, recycling leads to a 40% reduction in computation time and a 50% reduction in iterations. Notice that increasing s beyond 100 has limited effect since RMINRES rarely takes more than 100 iterations for this problem. However, for harder problems, e.g., for finer meshes, the solver may not converge so fast. In that case, larger values for s can be helpful. Note that increasing s does not increase the computational cost of RMINRES. The only limit on s is the memory size.

In the second set of experiments, we fix $s = 100$ and vary k. The parameter k affects both the computational cost per iteration, specifically the number of orthogonalizations and the cost of subspace selection, and the total number of iterations for the solver. There is a trade-off between these two factors, and in Figure 4.6 we compare the number of iterations and computational time for several values of k. Increasing k leads to a significant improvement in the convergence rate; towards the end we obtain a factor 3 reduction in the number of iterations. Time-wise, we obtain a 40% improvement. We also see that the computation time is not overly sensitive to the choice of k. For reference, both Figures 4.5 and 4.6 show the maximum change in the element densities for every optimization step.

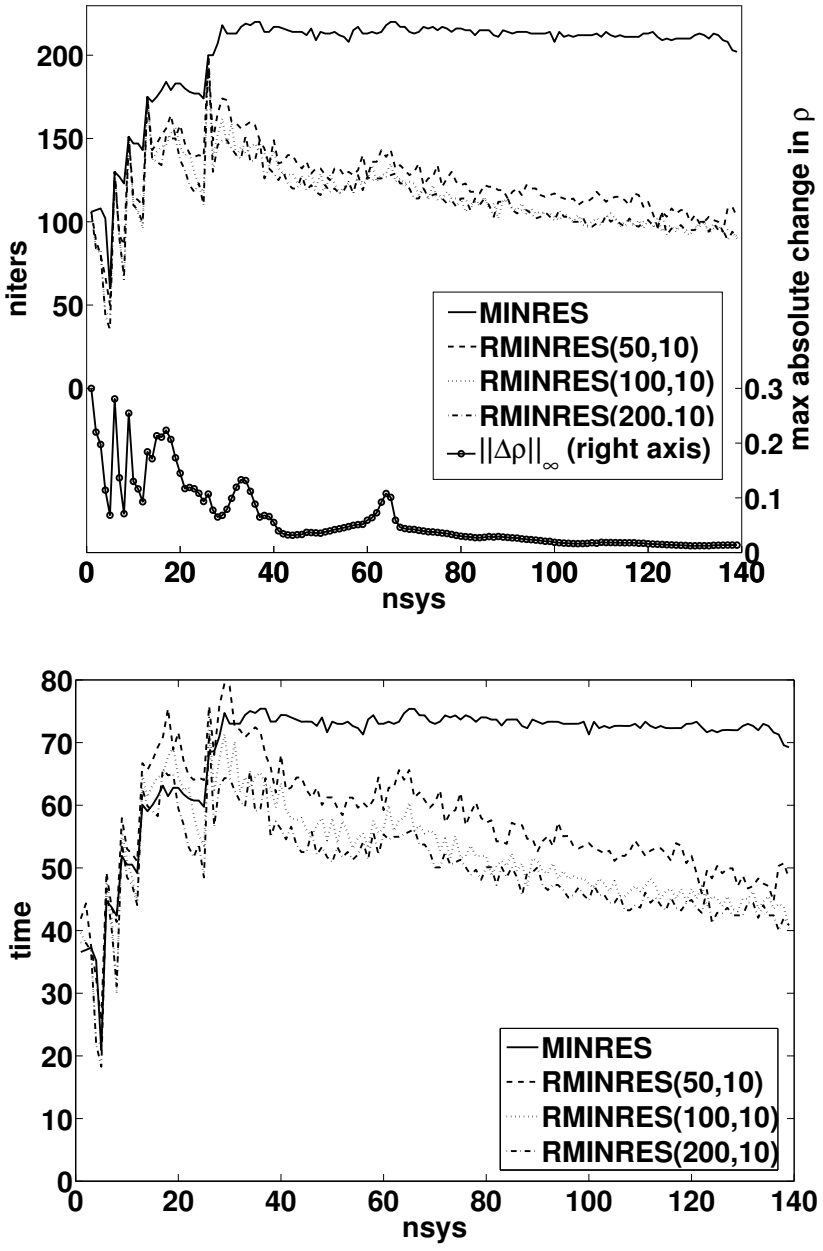


Figure 4.5: Number of iterations (niters) and time (seconds) of RMINRES(s, k) with fixed $k = 10$ and varying s.

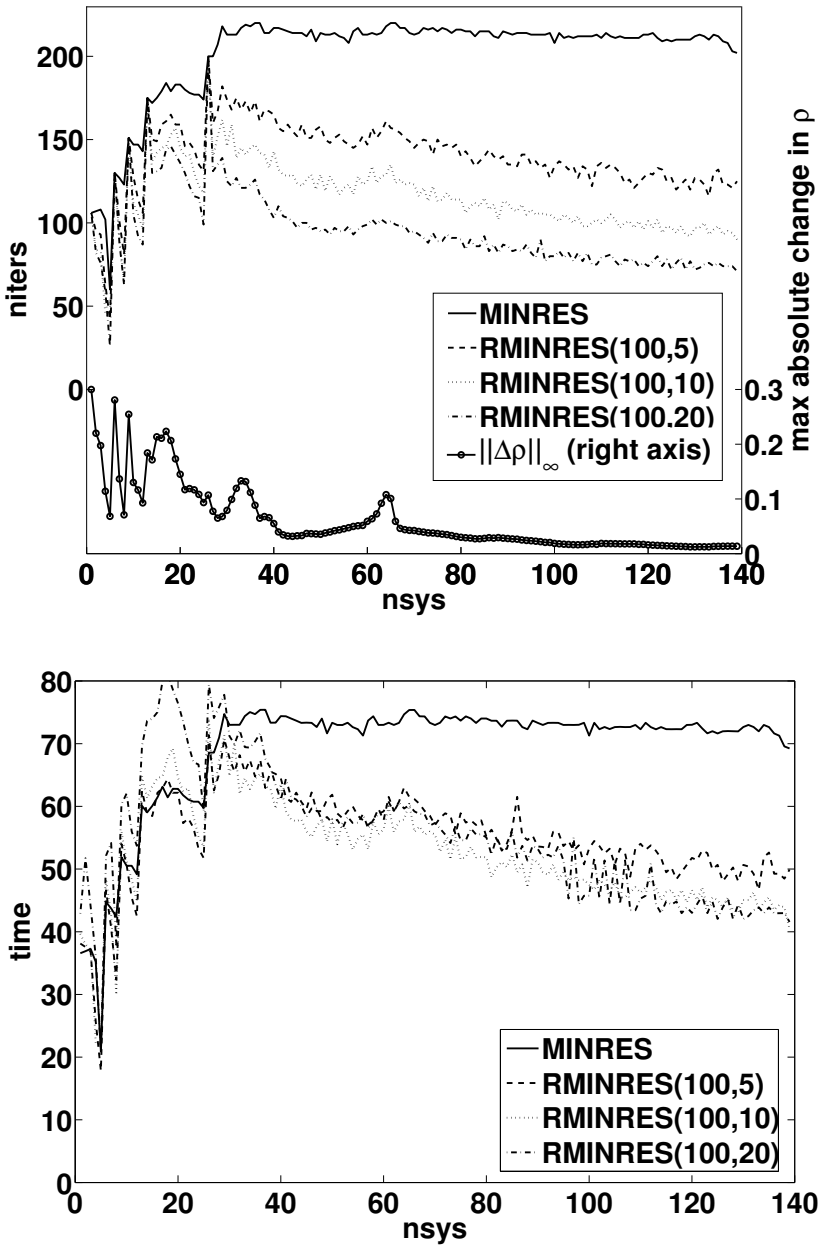


Figure 4.6: Number of iterations (niters) and time (seconds) of RMINRES(s, k) with varying k and fixed $s = 100$.

4.6 Recycling on Adaptive Meshes

When we use adaptive mesh refinement, every time the mesh changes, the size of the linear system changes, and the ordering of the degrees of freedom may change too. The recycle space obtained from the old system on the old mesh becomes meaningless for the new system on the new mesh without any interpretation. Moreover, the preconditioner introduces further complications to recycling. In general, preconditioning changes the spectrum of the linear system, in order to improves the convergence rate. When we carry out mesh refinement, we need to make sure that our recycle space from the old mesh well approximates the invariant subspace of the preconditioned systems corresponding to the smallest eigenvalues on the new mesh. For convenience, we refer to the eigenvectors corresponding to the smallest eigenvalues as "the smallest eigenvectors" in the following of this section. Also for the ease of discussion, we use the following notation. First, let $\boldsymbol{K}$ be the original stiffness matrix, $\widetilde{\boldsymbol{K}} = \boldsymbol{D}^{-1/2}\boldsymbol{K}\boldsymbol{D}^{-1/2}$ be the rescaled matrix as defined in (3.9), and $\widehat{\boldsymbol{K}} = \boldsymbol{L}^{-1}\widetilde{\boldsymbol{K}}\boldsymbol{L}^{-T}$ be the rescaled and preconditioned matrix as defined in (3.11). And let $\boldsymbol{v}$, $\tilde{\boldsymbol{v}}$ and $\hat{\boldsymbol{v}}$ be their smallest eigenvectors respectively. We use subscript "old" to denote matrices and vectors defined on the old mesh, and subscript "new" to denote matrices and vectors defined on the new mesh. We denote as $\boldsymbol{P}$ the projection operator that maps an vector from the old mesh to the new mesh by means of interpolation.

In topology optimization, we start with a homogeneous density distribution. The eigenvectors corresponding to the smallest eigenvalues of the linear system are smooth. When holes appear, e.g., in Figure 4.7, the smallest eigenvectors of the original stiffness matrix consist of smooth modes on the void and are close to zero on the material, see $\boldsymbol{v}$ in Figure 4.8. These eigenvectors contribute little to the solution of the equilibrium system. The smallest eigenvectors of the rescaled system, however, consist of the smooth global modes on the material instead of on the void, see $\tilde{\boldsymbol{v}}$ in Figure 4.9.

Figure 4.10 shows the smallest eigenvectors $\boldsymbol{v}$ of the IC preconditioned systems on both

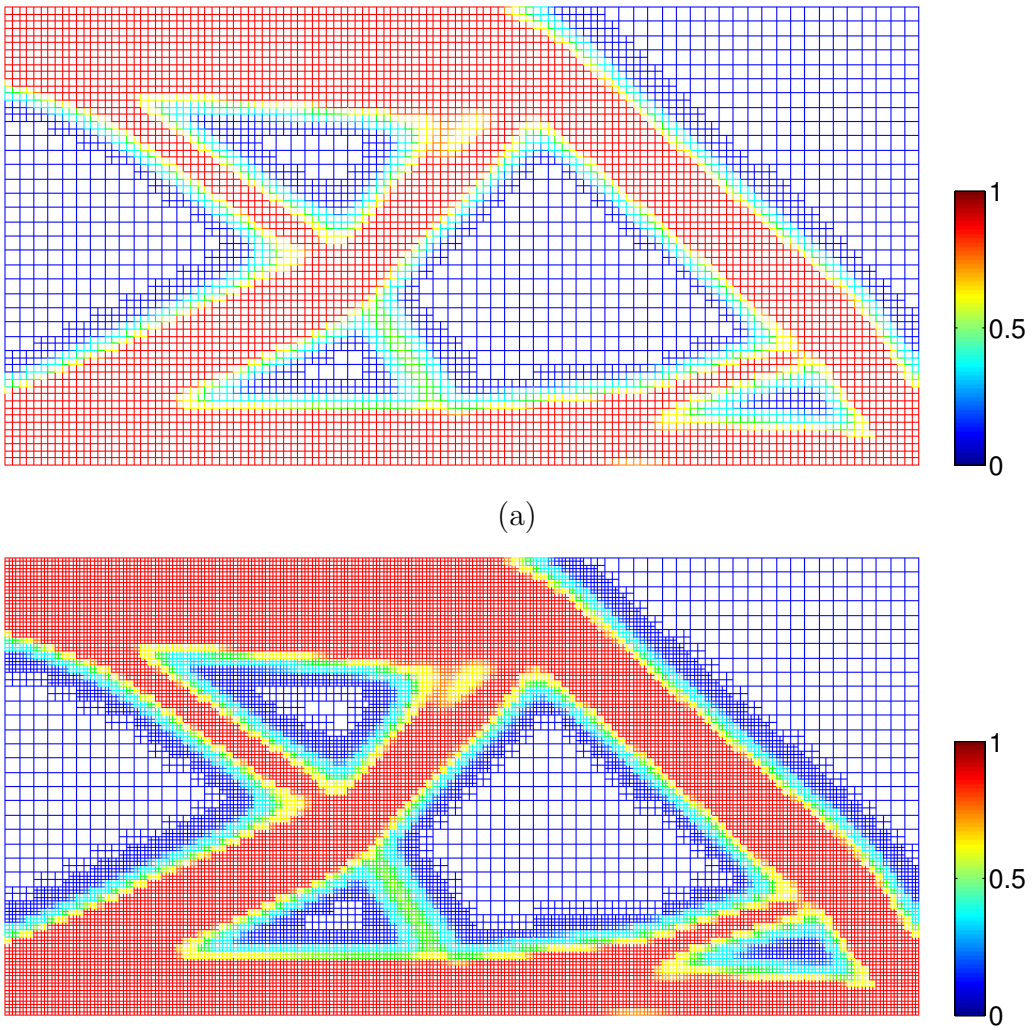


Figure 4.7: A typical density distribution. (a) the density distribution on a nonuniform mesh; (b) the density distribution projected on the new mesh that is adaptively refined from the mesh in (a).

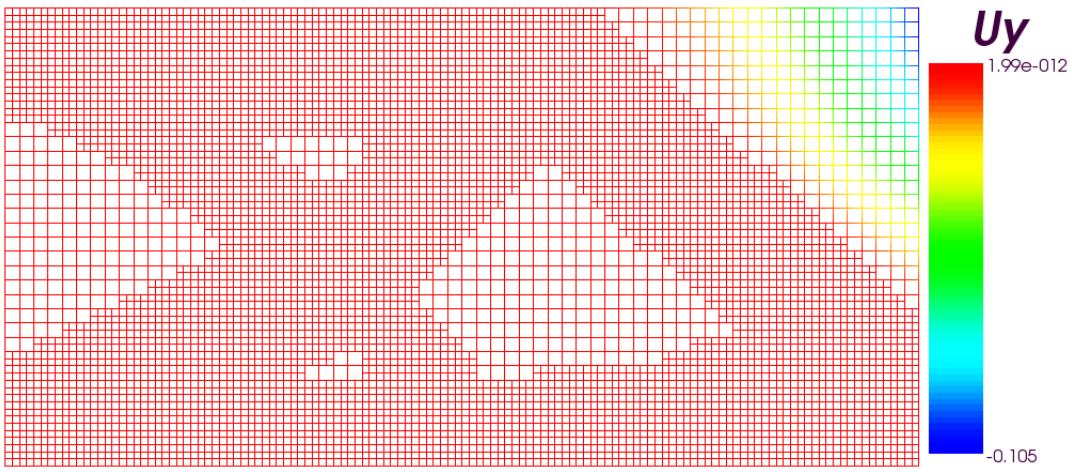


Figure 4.8: The smallest eigenvector $\boldsymbol{v}$ of the original stiffness matrix $\boldsymbol{K}$ on the old mesh shown in Figure 4.7(a).

the old and new meshes. Although, preconditioners do not address smooth modes well in general, incomplete Cholesky preconditioners do make the smallest eigenvectors less smooth than those of the unpreconditioned but rescaled system. If we project $\hat{\boldsymbol{v}}_{\text{old}}$ from the old mesh to the new mesh using the interpolation operator $\boldsymbol{P}$, $\boldsymbol{w} = \boldsymbol{P}\hat{\boldsymbol{v}}_{\text{old}}$ would not be a good approximation to the smallest eigenvectors $\hat{\boldsymbol{v}}_{\text{new}}$. To measure how well $\boldsymbol{w}$ approximates an eigenvector of $\widehat{\boldsymbol{K}}$, we normalize $\boldsymbol{w}$ and evaluate the following residual norm:

$$\left\| \widehat{\boldsymbol{K}}\boldsymbol{w} - (\boldsymbol{w}^T \widehat{\boldsymbol{K}}\boldsymbol{w}) \cdot \boldsymbol{w} \right\|. \tag{4.44}$$

In the example shown in Figures 4.7–4.10, this residual norm of the normalized $\boldsymbol{w}$ is 0.4507, which is not particularly small. Therefore, for Krylov subspace recycling with mesh refinement, the recycled harmonic Ritz vectors on the old mesh, which tend to approximate the smallest eigenvectors of the preconditioned system on the old mesh, would not make good approximations to the smallest eigenvectors of the preconditioned system on the new mesh after being projected to the new mesh by interpolation.

We discover that $\boldsymbol{L}^{-T}\hat{\boldsymbol{v}}$ is usually a very good approximation to $\tilde{\boldsymbol{v}}$, see Figure 4.11,

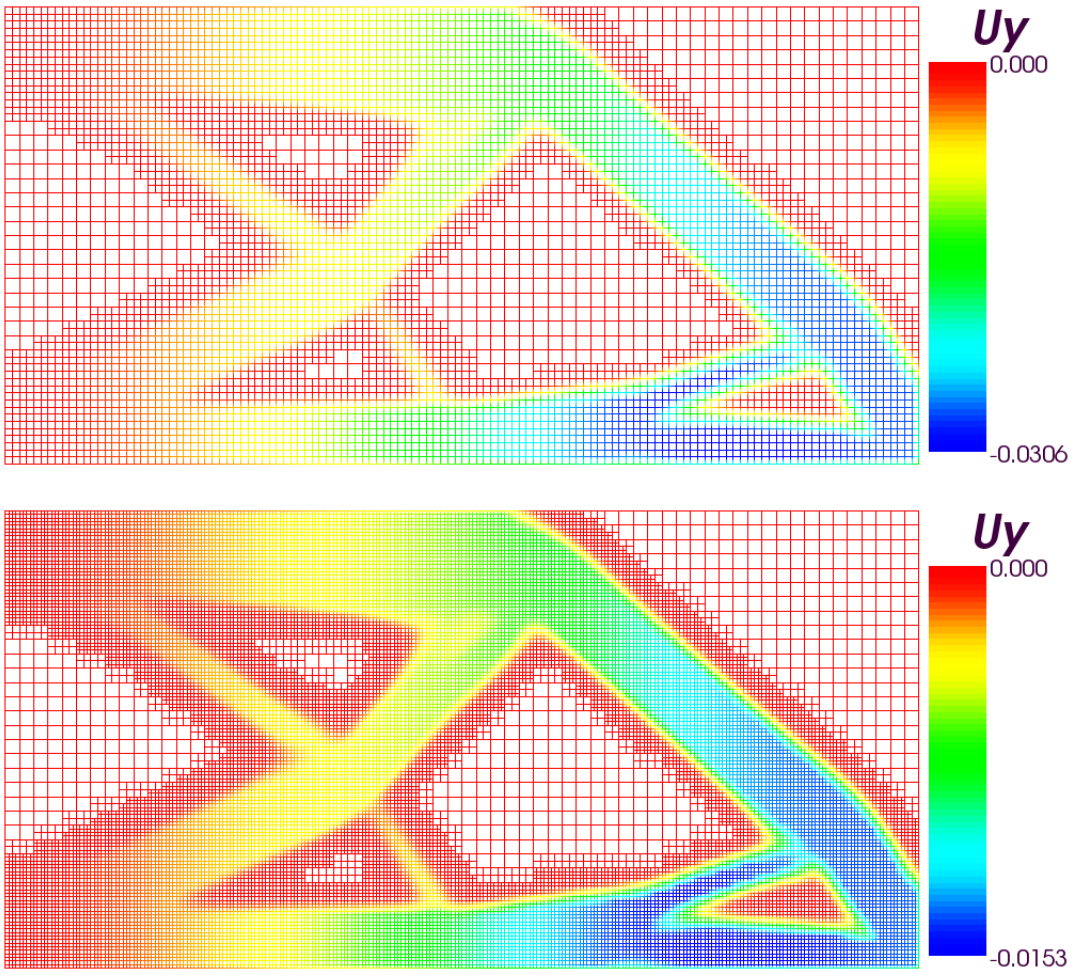


Figure 4.9: The smallest eigenvectors $\tilde{\boldsymbol{v}}$ (y direction DOFs only) of the diagonally rescaled matrices $\widetilde{\boldsymbol{K}}$ on both the old and new meshes shown in Figure 4.7.

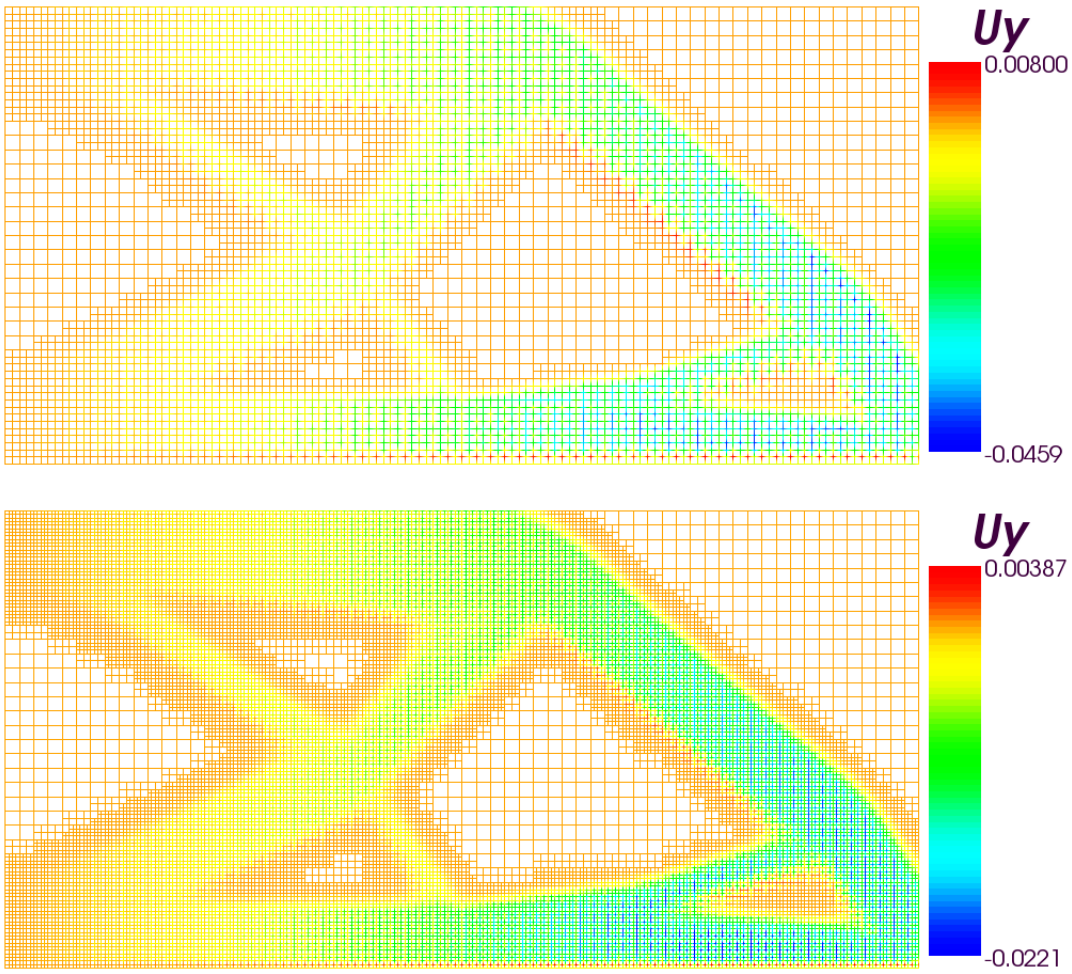


Figure 4.10: The smallest eigenvectors $\hat{\boldsymbol{v}}$ (y direction DOFs only) of the IC preconditioned systems $\widehat{\boldsymbol{K}} = \boldsymbol{L}^{-1}\widetilde{\boldsymbol{K}}\boldsymbol{L}^{-T}$ on both the old and new meshes shown in Figure 4.7.

Table 4.2: Approximation to the eigenvectors. Column 1 lists the matrices; column 2 lists the exact smallest eigenvectors of the matrices in column 1; column 3 lists the vectors we use to approximate the eigenvectors of the matrices in column 1; column 4 lists the eigenvector residual norms of the vectors in column 3 with respect to the matrices in column 1; column 5 lists the cosines of the angles between vectors in columns 2 and 3. The vectors in columns 2 and 3 are normalized before the evaluation of columns 4 and 5.

$\boldsymbol{A}$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\left\|\boldsymbol{A}\boldsymbol{x}_2 - (\boldsymbol{x}_2^T\boldsymbol{A}\boldsymbol{x}_2)\cdot\boldsymbol{x}_2\right\|$	$\boldsymbol{x}_1^T\boldsymbol{x}_2$
$\widetilde{\boldsymbol{K}}_{\text{old}}$	$\tilde{\boldsymbol{v}}_{\text{old}}$	$\boldsymbol{L}_{\text{old}}^{-T}\hat{\boldsymbol{v}}_{\text{old}}$	0.000010	0.999946
$\widetilde{\boldsymbol{K}}_{\text{new}}$	$\tilde{\boldsymbol{v}}_{\text{new}}$	$\boldsymbol{L}_{\text{new}}^{-T}\hat{\boldsymbol{v}}_{\text{new}}$	0.000002	0.999971
$\widehat{\boldsymbol{K}}_{\text{old}}$	$\hat{\boldsymbol{v}}_{\text{old}}$	$\boldsymbol{L}_{\text{old}}^{T}\tilde{\boldsymbol{v}}_{\text{old}}$	0.000000	0.999922
$\widehat{\boldsymbol{K}}_{\text{new}}$	$\hat{\boldsymbol{v}}_{\text{new}}$	$\boldsymbol{L}_{\text{new}}^{T}\tilde{\boldsymbol{v}}_{\text{new}}$	0.000010	0.999957
$\widetilde{\boldsymbol{K}}_{\text{new}}$	$\tilde{\boldsymbol{v}}_{\text{new}}$	$\boldsymbol{P}\tilde{\boldsymbol{v}}_{\text{old}}$	0.049190	0.998877
$\widehat{\boldsymbol{K}}_{\text{new}}$	$\hat{\boldsymbol{v}}_{\text{new}}$	$\boldsymbol{P}\hat{\boldsymbol{v}}_{\text{old}}$	0.450701	0.866497
$\widehat{\boldsymbol{K}}_{\text{new}}$	$\hat{\boldsymbol{v}}_{\text{new}}$	$\boldsymbol{L}_{\text{new}}^{T}\boldsymbol{P}\boldsymbol{L}_{\text{old}}^{-T}\hat{\boldsymbol{v}}_{\text{old}}$	0.166628	0.986535

and $\boldsymbol{L}^T\tilde{\boldsymbol{v}}$ is also usually a very good approximation to $\hat{\boldsymbol{v}}$. As long as the the projection of $\tilde{\boldsymbol{v}}_{\text{old}}$ to the new mesh makes a good approximation to $\tilde{\boldsymbol{v}}_{\text{new}}$, $\boldsymbol{L}_{\text{new}}^{T}\boldsymbol{P}\boldsymbol{L}_{\text{old}}^{-T}\hat{\boldsymbol{v}}_{\text{old}}$ would be a good approximation to $\hat{\boldsymbol{v}}_{\text{new}}$.

Unfortunately, although the angle between the projection of $\tilde{\boldsymbol{v}}_{\text{old}}$ to the new mesh and $\tilde{\boldsymbol{v}}_{\text{new}}$ is small, the residual of $\boldsymbol{P}\tilde{\boldsymbol{v}}_{\text{old}}$ as an eigenvector of $\widehat{\boldsymbol{K}}_{\text{new}}$ is not particularly small. Therefore, transforming the recycle space obtained on the old mesh to the new mesh in this way do not give a good approximation to the invariant subspace of the preconditioned system on the new mesh either. The comparison between different pairs of vectors and the measure of their approximations to the eigenvectors are listed in Table 4.2.

Further investigation is required for Krylov subspace recycling on adaptive meshes. For example, the fact that the angle between of $\boldsymbol{L}_{\text{new}}^{T}\boldsymbol{P}\boldsymbol{L}_{\text{old}}^{-T}\hat{\boldsymbol{v}}_{\text{old}}$ and $\hat{\boldsymbol{v}}_{\text{new}}$ is fairly small suggests that we may use a few iterations of the Jacobi-Davidson method [66] to make $\boldsymbol{L}_{\text{new}}^{T}\boldsymbol{P}\boldsymbol{L}_{\text{old}}^{-T}\hat{\boldsymbol{v}}_{\text{old}}$ a better approximation to the eigenvectors of $\widehat{\boldsymbol{K}}$.

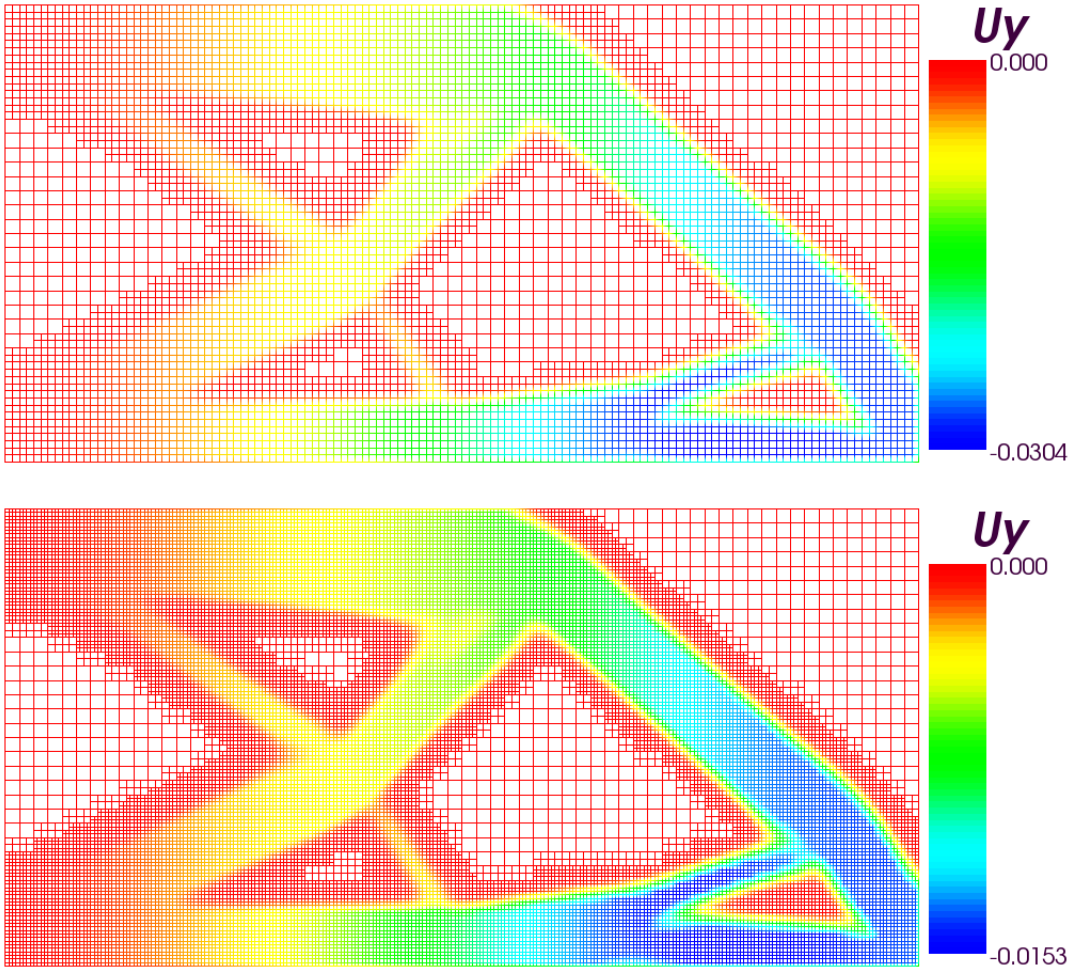


Figure 4.11: $\boldsymbol{L}^{-T}\hat{\boldsymbol{v}}$ (y direction DOFs only) on both the old and new meshes shown in Figure 4.7.

Chapter 5

Multilevel Sparse Approximate Inverse Preconditioner

Beyond recycling, preconditioning is an additional way to improve the performance of Krylov subspace methods. In general, discretized linear systems resulting from engineering and physical problems are ill-conditioned, which leads to slow convergence for iterative solvers. For large-scale simulations, preconditioners are usually required for iterative solvers to converge in a reasonable amount of time. In this chapter, we introduce a multilevel sparse approximate inverse preconditioner on adaptive meshes.

5.1 Introduction and Motivation

Since adaptive mesh refinement was proposed [14, 13], it has gained its popularity in the field of scientific simulations, mainly because it achieves high accuracy while keeping the overall computational cost relatively low. To make AMR more flexible and efficient, especially for parallel machines, the computational domain is usually partitioned into many small blocks, each of which represents a uniform mesh with a small fixed number of mesh cells. In parallel implementation, to maintain a good load balance, those blocks are redistributed over the processors after mesh refinement [38, 45], see Figure 5.1. Moreover, local refinement and/or derefinement on the mesh cause the system matrix to change. These properties of AMR introduce a variety of difficulties for preconditioners.

First, preconditioners with global coupling like ILU are constructed with respect to a certain ordering of the unknowns. In the factorization, every row and column depends on the previous ones. Even if the mesh refinement happens only locally, it will change

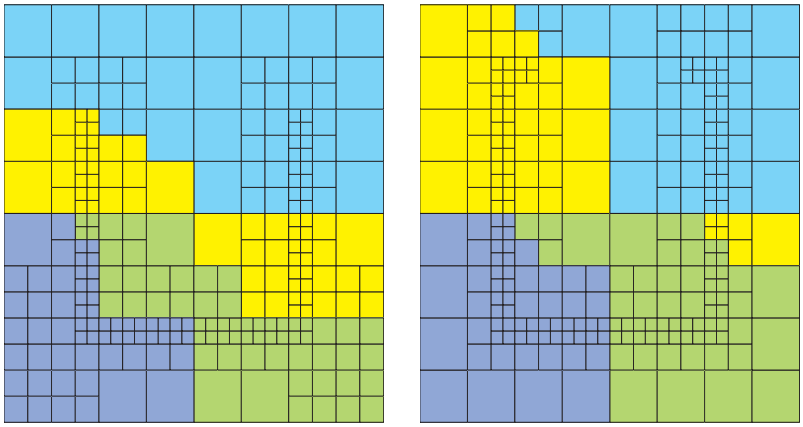


Figure 5.1: A typical data distribution before and after mesh refinement.

the system matrix, and thus affect the factorization for every following rows and columns with respect to the chosen ordering. Moreover, the forward and backward substitutions in these preconditioners have a strong data dependency. Although people have developed techniques to limit global dependencies while maintaining reasonable convergence [26, 27, 74], the redistribution of blocks for load balancing will turn local dependencies into global ones. So, mesh adaptation and load balancing make it hard for this class of preconditioners to adapt to highly dynamic meshes. Second, domain decomposition preconditioners, another important class of preconditioners, are not very suitable for AMR either, especially if changes in the mesh are relatively frequent. For domain decomposition, the whole physical domain is decomposed into large blocks of contiguous subdomains, which are assigned to different processors. In the AMR context, the decomposition of the domain and the boundaries of the subdomains will change dynamically, requiring frequent recomputing of the local factorizations and of the Schur complement (or analogous components of a domain decomposition algorithm). This reconstruction of the local solvers and Schur complement at every time step would be very expensive.

Sparse approximate inverses are another class of preconditioners. They are sparse matrices that approximate the inverse of a sparse matrix either in explicit or implicit (factorized) form [35, 11, 22, 24, 23, 12]. There are several ways to construct such sparse

approximate inverses. One method, usually referred as SPAI, minimizes the Frobenius norm of $\boldsymbol{AM} - \boldsymbol{I}$ subject to some sparsity pattern [35, 24, 22], and gives an explicit representation of the approximation to the matrix inverse. Another way is to construct an approximate factorization of $\boldsymbol{A}^{-1} \approx \boldsymbol{Z}\boldsymbol{D}^{-1}\boldsymbol{W}^T$, where $\boldsymbol{Z}$ and $\boldsymbol{W}$ are unit upper triangular and $\boldsymbol{D}$ is diagonal. This includes FSAI [42], AINV [11], and SAINV [10], which in many cases are more effective than SPAI in term of improving the convergence rate. However, they suffer the same problems as ILU does.

In the AMR context, explicit SPAI turns out to overcome the difficulties mentioned above. The approximate inverses have very weak data dependency. To calculate each column of an approximate inverse, we need only the few columns of the system matrix that correspond to the neighboring cells. After mesh refinement, only those columns of the system matrix that are associated with the changed cells are changed. Hence, we can update only a few columns of the approximate inverse, and reuse all the other columns without changes. This makes updating the approximate inverse relatively cheap.

Moreover, the approximate inverses are stored explicitly as sparse matrices form and applied to vectors by matrix vector multiplications. There is no forward or backward substitution for SPAI. So, the data redistribution does not affect the sparse approximate inverses very much. Moreover, due to the weak data dependency, both the construction and the application of SPAI are easy to parallelize.

Unfortunately, SPAI has a serious drawback. SPAI does not effectively capture the low-frequency global modes of the underlying operator. Therefore, in many applications, a very large sparsity pattern is required for an approximate inverse to get a good convergence rate. Such a large sparsity pattern leads to high computational cost to construct the approximate inverse and apply it at every iteration. However, we can remedy this problem by multilevel techniques at low cost. Several approaches for improving sparse approximate inverses using multilevel techniques have been proposed. In [70], sparse approximate inverses are used as smoothers for multigrid methods. In [19], sparse approximate inverses are combined with

changes of basis using wavelets to derive a hierarchical structure. In [16, 75], algebraic information is explored to construct multilevel sparse approximate inverse preconditioners. We discuss these approaches more in depth at the end of section 5.3. In this thesis, for adaptive hierarchical meshes, we introduce a new multilevel method to improve sparse approximate inverse preconditioners. Exploiting the hierarchical structure of refined meshes, our method significantly reduces the number of iterations while remaining efficient in cost per iteration and in computing the updating the preconditioner for highly dynamic mesh. This is achieved without requiring global algebraic information from the matrix.

We first present and analyze our preconditioner in the context of scalar diffusion and convection-diffusion problems in Sections 5.2, 5.3, and 5.4. Then, we generalize the preconditioner to 2D and 3D elasticity problems, and subsequently apply it to topology optimization problems in Sections 5.5 and 5.6.

5.2 Sparse Approximate Inverse Preconditioner

We consider the solution of linear time-dependent diffusion and convection-diffusion problems on adaptive meshes. In general, we can write these problems as

$$u_t = (a_1 u_x)_x + (a_2 u_y)_y + b_1 u_x + b_2 u_y + cu + f, \tag{5.1}$$

where coefficients a_1, a_2, b_1, b_2, c, and f can be functions of x and/or t but not of u or its partial derivatives. We discretize these partial differential equations in space using a finite difference (or finite volume) method and in time using the backward Euler method or the Crank-Nicolson method. This results in the following systems of linear equations:

$$\boldsymbol{A}\boldsymbol{u}^{(n+1)} + \boldsymbol{B}\boldsymbol{u}^{(n)} = \boldsymbol{u}^{(n)}, \tag{5.2}$$

where n denotes the time step. We are concerned with linear solvers and preconditioners to solve such systems efficiently. If the coefficient functions do not change with time, the linear system (5.2) depends only on the time step and the mesh. In many situations, the time step can be fixed, but mesh refinement and derefinement locally change the discretization. So, we need preconditioners that are not be affected too much by the local changes of the mesh and are cheap to update. Sparse approximate inverses satisfy our requirements.

We consider the preconditioned system $\boldsymbol{AMy} = \boldsymbol{b}$ with preconditioner $\boldsymbol{M}$ and $\boldsymbol{x} = \boldsymbol{My}$. We want to choose $\boldsymbol{M}$ such that $\boldsymbol{AM}$ is a good approximation to the identity matrix while $\boldsymbol{M}$ is easy to compute, update, and apply. A popular way to compute such sparse matrix $\boldsymbol{M}$ is to minimize the Frobenius norm [32, p. 55] of $\boldsymbol{AM} - \boldsymbol{I}$ [35, 24, 22]. Since

$$\|\boldsymbol{AM} - \boldsymbol{I}\|_F^2 = \sum_j \|\boldsymbol{Am}_j - \boldsymbol{e}_j\|_2^2, \tag{5.3}$$

where $\boldsymbol{m}_j$ is the jth column of $\boldsymbol{M}$, we can compute each column of $\boldsymbol{M}$ independently by minimizing $\|\boldsymbol{Am}_j - \boldsymbol{e}_j\|_2$ for a chosen sparsity pattern (with a few nonzeros per column). So $\boldsymbol{M}$ can be computed by solving many small least squares problems in parallel, and can be stored in explicit matrix form.

The exact inverse of a system matrix for a convection-diffusion problem is typically full. Every column has nonzero coefficients for all rows. Figure 5.2 shows the Green's function for a 1D diffusion problem and a point source in the middle. Each the column of the exact inverse corresponds to a (discrete) Green's function, and the largest coefficients correspond to the mesh points around the point source. Therefore, for elliptic problems we typically choose a sparsity pattern for $\boldsymbol{M}$ that contains only a few neighboring mesh cells. This also makes it cheap to compute and apply $\boldsymbol{M}$.

The choice of the sparsity pattern is usually the key issue for an effective sparse approximate inverse preconditioner. A small sparsity pattern (a pattern/stencil with few nonzeros) yields a cheap preconditioner, but generally leads to slow convergence. For an

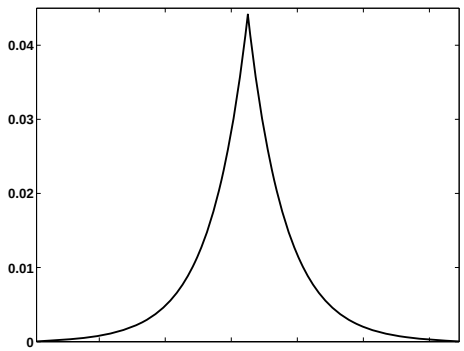
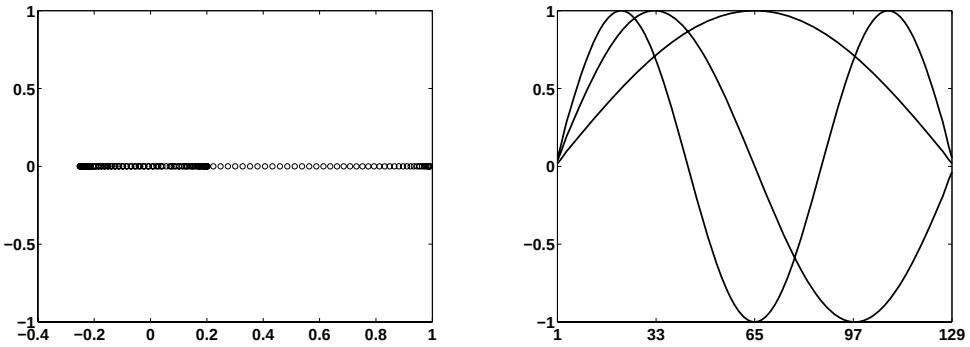


Figure 5.2: Discrete Green's function for the 1D diffusion problem $-au_{xx} + u = \delta$ on a 129 point grid with a point source δ in the middle.

intuitive explanation, consider the 1D equation $u_{xx} = 0$ for $x \in (0, 1)$ with $u(0) = 1$ and $u(1) = 0$, whose solution would be $u(x) = 1 - x$. We discretize the problem in the standard fashion using $N + 2$ grid points and a 3-point stencil, and we choose $\boldsymbol{M}$ to have the same sparsity pattern as the system matrix $\boldsymbol{A}$. If we take $\boldsymbol{u} = 0$ as the initial guess, then with every iteration of a Krylov subspace method, the nonzero values in the approximate solution propagate by two grid points to the right. Therefore, it will take at least $N/2$ iterations to converge, because a boundary condition or a local source term in elliptic problems influences the solution over the entire domain. In another word, the solution procedure only stepwise updates neighboring regions. So, the number of iterations for convergence is inherently bounded from below by the "diameter" of the domain and the "diameter" of the image of the operator $\boldsymbol{AM}$ for a point function. A larger sparsity pattern for $\boldsymbol{M}$ would include more global information per iteration, which would improve the rate of convergence. However a large sparsity pattern results in high computational cost in both constructing and applying the sparse approximate inverse. This makes the preconditioner too expensive.

Before introducing our approach to improve SPAI, we review this problem further. An observation in [15] for the Laplace equation shows that although most eigenvalues of the residual matrix for the right preconditioned system, $\boldsymbol{E} = \boldsymbol{I} - \boldsymbol{AM}$, are close to zero, there are a small number of eigenvalues very close to 1. Even a significantly larger stencil (more



(a) (b)

Figure 5.3: Eigenvalues and eigenvectors of the residual matrix for the discretized system of (5.4) and its SPAI. (a) eigenvalues of $\boldsymbol{E} = \boldsymbol{I} - \boldsymbol{AM}$; (b) eigenvectors corresponding to the largest 3 eigenvalues.

nonzero coefficients) for the approximate inverse $\boldsymbol{M}$ does not cure this. In general, for this problem, the eigenvalues of $\boldsymbol{E}$ that are close to 1 correspond to smooth eigenvectors. That means sparse approximate inverses are not good at handling global low-frequency modes. For the 1D diffusion problem $u_t = u_{xx}, x \in (0,1)$ with a homogeneous Dirichlet boundary condition, the discretization along the time axis for a specified time step $\triangle t$ results in the following equation:

$$-\triangle t u_{xx} + u = u_{\text{old}}, \quad \text{for } x \in (0,1) \text{ and } u(0) = u(1) = 0. \tag{5.4}$$

We discretize 5.4 with 129 points uniformly along the x-axis and obtain the system matrix $\boldsymbol{A}$, and then we compute its SPAI $\boldsymbol{M}$ with the same sparsity pattern as $\boldsymbol{A}$. Figure 5.3 shows the eigenvalues of $\boldsymbol{E} = \boldsymbol{I} - \boldsymbol{AM}$ and the eigenvectors corresponding to the 3 largest eigenvalues.

Now consider the following equation:

$$-a u_{xx} + u = f \quad \text{on } (0,1), \quad u(0) = u(1) = 0. \tag{5.5}$$

Let $g(x,\tau)$ be its Green's function for a point source (Dirac's delta function) at τ [18]. We

can represent the continuous solution $u(x)$ as

$$u(x) = \int_0^1 g(x,\tau) f(\tau)\,\mathrm{d}\tau, \tag{5.6}$$

where the Green's function can be explicitly written out as follows using Fourier transform [18, p. 95]:

$$g(x,\tau) = \sum_{k=1}^{\infty} \frac{2\sin k\pi\tau}{1 + ak^2\pi^2} \sin k\pi x. \tag{5.7}$$

Analogously, we can represent the discrete solution of $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$ by

$$\boldsymbol{u} = \sum_{j=1}^{n} (\boldsymbol{A}^{-1})_j \boldsymbol{f}_j, \tag{5.8}$$

where $(\boldsymbol{A}^{-1})_j$ indicates the jth column of $\boldsymbol{A}^{-1}$. $\boldsymbol{A}^{-1}$ is the discrete analog and approximation to the Green's function $g(x,\tau)$, each column $(\boldsymbol{A}^{-1})_j$ representing the approximate solution for a point source. It is clear from (5.7) that the low-frequency components have much larger weights than the high-frequency components. Similarly, the columns of $\boldsymbol{A}^{-1}$ have relatively large low-frequency components and small high-frequency components. In fact, (5.6) and (5.7) indicate that unless $\boldsymbol{f}$ has very large high-frequency components the solution is largely determined by the low-frequency components. Therefore, we cannot expect to approximate $\boldsymbol{A}^{-1}$ accurately unless we represent the low-frequency components reasonably accurately.

Unfortunately, accurately representing the low-frequency components with respect to the standard basis provided by the mesh requires the approximation to the inverse, $\boldsymbol{M}$, to be fairly dense (even if many of the coefficients are relatively small). This makes the construction of $\boldsymbol{M}$ and the multiplication by $\boldsymbol{M}$ very expensive, particularly on a parallel computer. Therefore, for the purpose of efficiency we require a practical sparsity pattern for an approximate inverse to have a small local stencil, often the same as that of the matrix $\boldsymbol{A}$ itself. In that case, the Frobenius norm minimization (5.3) gives the columns of the approximate inverse a small wedge shape (see Figure 5.4(a)). If we look at the frequency

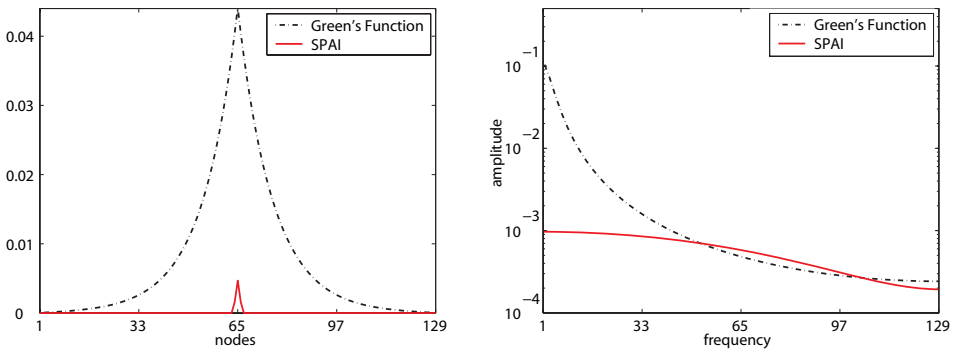


Figure 5.4: Green's function vs. SPAI. (a) the Green's function for point source at the midpoint, and its SPAI approximation; (b) frequency domain representations of the Green's function and its SPAI approximation.

decomposition in Figure 5.4(b), SPAI approximates the Green's function very well on high-frequency modes, but has large errors on low-frequency modes. As the approximation is highly localized, SPAI is unable to capture the low-frequency components well.

This problem has been recognized by several people, and various methods have been proposed to remedy it [19, 16, 21, 20]. The common underlying idea for these approaches is to construct a new basis, such that the representation of the Green's function with respect to this basis is nearly a diagonal matrix; that is, outside a narrow band, the representation of the Green's function is nearly zero. This allows an accurate approximation of the inverse by a sparse approximate inverse with few nonzeros. In these approaches one has to construct the new basis, the basis transformation and its inverse, and the representation of the approximate inverse with respect to this new basis. These procedures are not cheap but for hard problems they may pay off in a greatly reduced number of iterations. In [19, 21, 20] the authors discuss various approaches using hierarchical wavelet bases. In particular, in [19] two hierarchical wavelet bases are constructed using second generation wavelets, so that unstructured meshes can be handled well. The idea is that smooth regions in the Green's function lead to sufficiently small wavelet coefficients that are accurately approximated by zeros. In [16], algebraic information is used to find a multilevel basis. To be specific, the coarsening process

is based on the construction of the sparse approximate inverse.

Although the ideas behind our method for improving the approximation of smooth components of $\boldsymbol{A}^{-1}$ are similar, our approach is different. First, we use the hierarchy already present in the AMR meshes to construct a hierarchical preconditioner. Therefore, we do not need to construct any new basis to work with. Second, we exploit the fact that the smooth components of the Green's function can be represented cheaply and reasonably accurately using only a few nonzero coefficients at coarse levels. Hence, we do not aim for a basis in which many or most coefficients of the approximate inverse can be approximated accurately by zeros. Rather, we exploit the hierarchy of meshes in the standard basis associated with the mesh to approximate the components of the Green's function as economically as possible at coarser levels. The key observation is that representing 'most' of $\boldsymbol{M}$ at the coarse levels leads to efficient storage of $\boldsymbol{M}$, and makes the multiplication by $\boldsymbol{M}$ very cheap. We will introduce our method in detail in the next section.

5.3 Multilevel Sparse Approximate Inverses on Adaptive Meshes

We first introduce some notation for adaptive (AMR) meshes. The adaptive mesh refinement yields a hierarchy of uniform meshes, denoted by Ω_ℓ (see the 2D example in Figure 5.5). Higher level meshes have increasingly finer resolution, and are typically restricted to smaller and smaller part of the domain. Note that only at the coarsest level the mesh must be contiguous. In addition to these meshes, we consider their compositions. As indicated in Figure 5.5, we recursively define $\hat{\Omega}_\ell$ as the composite mesh that results from combining meshes Ω_ℓ and $\hat{\Omega}_{\ell-1}$, where those mesh components (points, faces, cells) on level $\ell-1$ that are covered by Ω_ℓ are excluded. The initial composite mesh is $\hat{\Omega}_1 = \Omega_1$. In many applications, we have a minimum level of refinement ℓ^*, i.e., Ω_{ℓ^*} covers the whole domain. So $\hat{\Omega}_\ell = \Omega_\ell$ for $\ell \leq \ell^*$. We obtain meshes above level ℓ^* by local refinements as required by the solution

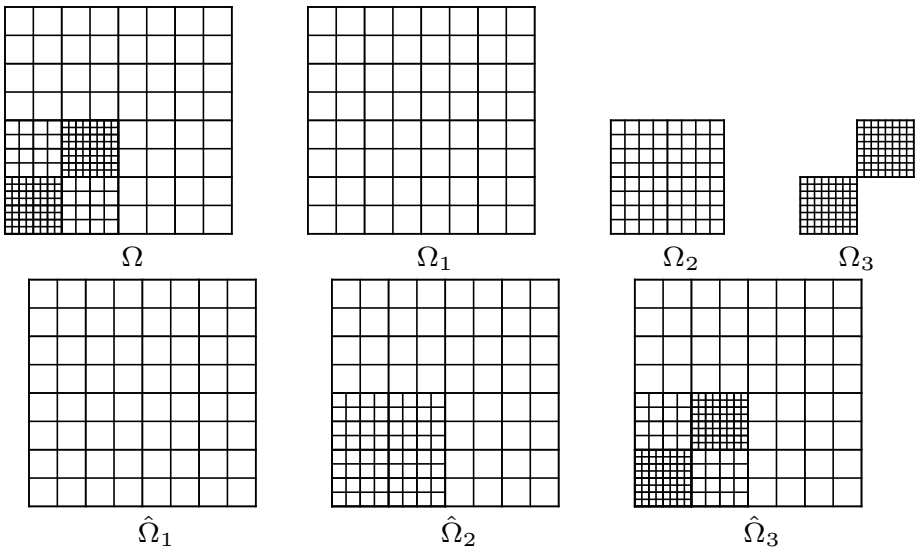


Figure 5.5: A hierarchy of meshes (first row) and the corresponding composite meshes (second row).

accuracy.

Now we explain how matrix-vector multiplications can be done on a nonuniform, hierarchical mesh with the help of ghost cells. We give a one-dimensional example first. Figure 5.6 shows a composite mesh of two levels, with cells $i-1$ and i on level 1, and cells j and $j+1$ on level 2. On level 1, cell i would have a ghost neighbor cell $i+1$ on its right, since its real left neighbor cell is on a different level. Similarly, on level 2, cell j would have a ghost neighbor cell $j-1$ on its left. We discretize for example a Laplace PDE at cells i and j in the following way:

$$-u_{i-1} + 2u_i - \tilde{u}_{i+1} = 0, \tag{5.9}$$

$$-\tilde{u}_{j-1} + 2u_j - u_{j+1} = 0. \tag{5.10}$$

We compute the values at the ghost cells using linear interpolations:

$$\tilde{u}_{i+1} = \frac{1}{2}u_j + \frac{1}{2}u_{j+1}, \tag{5.11}$$

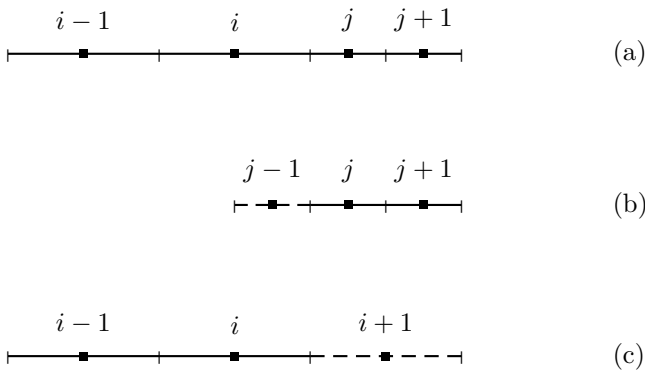


Figure 5.6: One-dimensional composite mesh with ghost cells: (a) one dimensional composite mesh with two levels; (b) the fine level uniform submesh with a ghost cell; (c) the coarse level uniform submesh with a ghost cell.

$$\tilde{u}_{j-1} = \frac{3}{4}u_i + \frac{1}{4}\tilde{u}_{i+1} = \frac{3}{4}u_i + \frac{1}{8}u_j + \frac{1}{8}u_{j+1}. \tag{5.12}$$

A matrix-vector multiplication $\boldsymbol{v} = \boldsymbol{A} * \boldsymbol{u}$ would give us

$$v_i = -u_{i-1} + 2u_i - (\frac{1}{2}u_j + \frac{1}{2}u_{j+1}), \tag{5.13}$$

$$v_j = -(\frac{3}{4}u_i + \frac{1}{8}u_j + \frac{1}{8}u_{j+1}) + 2u_j - u_{j+1}. \tag{5.14}$$

Therefore, we can write the matrix at each cell using a standard finite difference discretization, and store the coefficients with respect to a same standard stencil as in (5.9) and (5.10). A matrix-vector multiplication can be done in two steps. First, we fill the ghost cells by interpolation. Then, we compute the matrix-vector product at each cell using its neighboring cells in the discretization stencil, which are at the same level and can be either existing cells or ghost cells. In this way, every cell does not see the mesh change in its neighborhood. Mesh refinement only changes how its neighboring cells (either ghost or not) are filled. Furthermore, this homogeneous representation allow us to represent the system matrix on coarser meshes by discretizing the PDE with the same stencil but larger cells.

To make matrix-vector multiplication cheap and convenient, we store our sparse matrices

row by row. The rows of $\boldsymbol{A}$ and $\boldsymbol{M}$ are the columns of $\boldsymbol{A}^T$ and $\boldsymbol{M}^T$. So, we minimize

$$\|\boldsymbol{M}\boldsymbol{A} - \boldsymbol{I}\|_F = \|\boldsymbol{A}^T\boldsymbol{M}^T - \boldsymbol{I}\|_F, \tag{5.15}$$

instead of $\|\boldsymbol{A}\boldsymbol{M} - \boldsymbol{I}\|_F$. Due to the chosen sparsity pattern, each row of $\boldsymbol{M}$ depends only on the rows of $\boldsymbol{A}$ that correspond to the neighboring cells. We store and multiply $\boldsymbol{M}$ in the same way as $\boldsymbol{A}$. Through the use of ghost cells, the rows of $\boldsymbol{M}$ remain unchanged if the corresponding cells are not refined or derefined.

Since we are minimizing $\|\boldsymbol{M}\boldsymbol{A} - \boldsymbol{I}\|_F$, we choose to use left preconditioning, which leads to the preconditioned system $\boldsymbol{M}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}\boldsymbol{b}$ and the residual matrix $\boldsymbol{E} = \boldsymbol{I} - \boldsymbol{M}\boldsymbol{A}$. However, in general, the preconditioning methods we introduce below can be used for both left and right preconditioning.

Next, using the definition of composite meshes, we define our multilevel preconditioner, given the matrix $\boldsymbol{A}_\ell$ and its sparse approximate inverse $\boldsymbol{M}_\ell$ defined on each composite mesh $\hat{\Omega}_\ell$. We start with a two-level version and define the multilevel version recursively. Algorithm 5.1 describes the multiplication of the basic two-level preconditioner $\boldsymbol{M}_2$ and a vector $\boldsymbol{z}$ for a fine composite grid, $\hat{\Omega}_h$, and a coarse composite grid, $\hat{\Omega}_H$. Note that $\hat{\Omega}_H$ does not need to be the next coarser grid of $\hat{\Omega}_h$. It can be an arbitrary coarser grid.

Algorithm 5.1: TWO-LEVEL SPAI: Compute $\boldsymbol{y} \leftarrow \boldsymbol{M}_2\boldsymbol{z}$

1. Multiply fine mesh sparse approximate inverse to $\boldsymbol{z}$: $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{M}_h\boldsymbol{z}$;
2. Compute fine mesh "residual": $\boldsymbol{r}_h \leftarrow \boldsymbol{z} - \boldsymbol{A}_h\hat{\boldsymbol{y}}$;
3. Restrict $\boldsymbol{r}_h$ to the coarse mesh: $\boldsymbol{r}_H \leftarrow \boldsymbol{I}_h^H\boldsymbol{r}_h$;
4. Apply coarse mesh sparse approximate inverse to $\boldsymbol{r}_H$: $\boldsymbol{e}_H \leftarrow \boldsymbol{M}_H\boldsymbol{r}_H$;
5. Prolong $\boldsymbol{e}_H$ to the fine mesh: $\boldsymbol{e}_h \leftarrow \boldsymbol{I}_H^h\boldsymbol{e}_H$;
6. Correct the preconditioned vector: $\boldsymbol{y} \leftarrow \hat{\boldsymbol{y}} + \boldsymbol{e}_h$;

The final result $\boldsymbol{y} = \boldsymbol{P}\boldsymbol{z}$ consists of an initial approximation $\boldsymbol{M}_h\boldsymbol{z}$ to $\boldsymbol{A}^{-1}\boldsymbol{z}$ (step 1), defined by the standard sparse approximate inverse preconditioner $\boldsymbol{M}_h$, and a coarse mesh correction using $\boldsymbol{M}_H$ (steps 2–6).

We denote this two-level method by SPAI2, where $\hat{\Omega}_h$ is $\hat{\Omega}_{\ell_{\max}}$, the finest composite mesh.

However, $\hat{\Omega}_H$ does not need to be the next coarser grid $\hat{\Omega}_{\ell_{\max}-1}$. A good choice for $\hat{\Omega}_H$ is $\hat{\Omega}_{\ell^*}$, for $\hat{\Omega}_{\ell^*}$ is invariant. In that case, it is worthwhile to construct a more accurate sparse approximate inverse there. The overall preconditioning operator for SPAI2 is

$$\boldsymbol{M}_2 = \boldsymbol{M}_h + \boldsymbol{I}_H^h \boldsymbol{M}_H \boldsymbol{I}_h^H (\boldsymbol{I} - \boldsymbol{A}\boldsymbol{M}_h), \tag{5.16}$$

and the residual matrix becomes

$$\boldsymbol{E}_2 = \boldsymbol{I} - \boldsymbol{M}_2\boldsymbol{A} = (\boldsymbol{I} - \boldsymbol{I}_H^h \boldsymbol{M}_H \boldsymbol{I}_h^H \boldsymbol{A})(\boldsymbol{I} - \boldsymbol{M}_h\boldsymbol{A}). \tag{5.17}$$

It has most of the eigenvalues moved closer to the origin compared with the residual matrix of standard SPAI, since $\boldsymbol{I}_H^h \boldsymbol{M}_H \boldsymbol{I}_h^H \boldsymbol{A}$ in general captures smooth mode much better than $\boldsymbol{M}\boldsymbol{A}$.

To turn this two-level preconditioner into a multilevel preconditioner, instead of applying $\boldsymbol{M}_H$ to get $\boldsymbol{e}_H = \boldsymbol{M}_H \boldsymbol{r}_H$ on the coarse grid $\hat{\Omega}_H$ at step 4, we recursively apply this algorithm to $\boldsymbol{r}_H$ to get a more accurate correction $\boldsymbol{e}_H$, especially in the sense of getting more global information. We denote our multilevel sparse approximate inverse preconditioners as MSPAI. In most cases, we carry out the recursive correction down to the coarsest level. However, for PDEs with high oscillatory coefficients, a discretization on an overly coarse mesh may not be able to capture the physics. Therefore, a correction on those coarse levels would be less useful. In these cases, the recursive scheme should stop at an appropriate level.

For the same 1D problem as in Figure 5.2, we demonstrate the improvement of MSPAI over SPAI in Figure 5.7. Unlike SPAI, the MSPAI approximation is almost full, which means it captures the global information. It approximates the Green's function on the low-frequency modes much better than SPAI. However, the full sparsity pattern of MSPAI here does not reflect its actual computational cost. By collecting global information in a multilevel way, we obtain a very good approximation to the exact inverse at fairly low cost. We discuss this

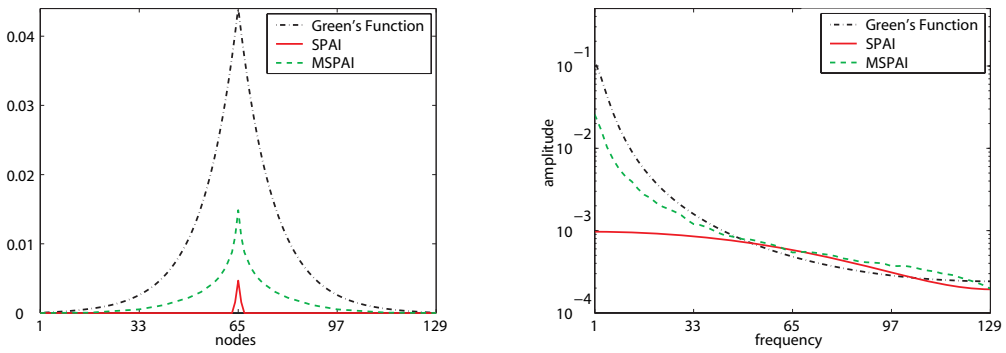


Figure 5.7: Green's function vs. SPAI and MSPAI. (a) the Green's Function for point source at the midpoint, and its SPAI and MSPAI approximations; (b) frequency domain representation of the Green's function, and its SPAI and MSPAI approximations.

issue more in Section 5.4.

5.4 Numerical Experiments with Diffusion and Convection-Diffusion Problems

In this section, we use NONE to denote no preconditioning, SPAI to denote the one-level sparse approximate inverse preconditioner with predetermined sparsity pattern, SPAI2 to denote the two-level sparse approximate inverse with the coarse mesh chosen on the coarsest refinement level ℓ^* as mentioned before, and MSPAI to denote the full multilevel sparse approximate inverse down to the coarsest refinement level. Moreover, we use $\boldsymbol{I}$, $\boldsymbol{M}$, $\boldsymbol{M}_2$, $\boldsymbol{M}_m$ to denote the preconditioning matrices for NONE, SPAI, SPAI2 and MSPAI respectively.

We explain some implementation details first, and then present experimental results for three model problems.

5.4.1 Implementation with `PARAMESH`

To explain the implementation details, we introduce `PARAMESH` [45], the AMR package we use. `PARAMESH` is a FORTRAN90 parallel package, which supports simulation on multidimensional

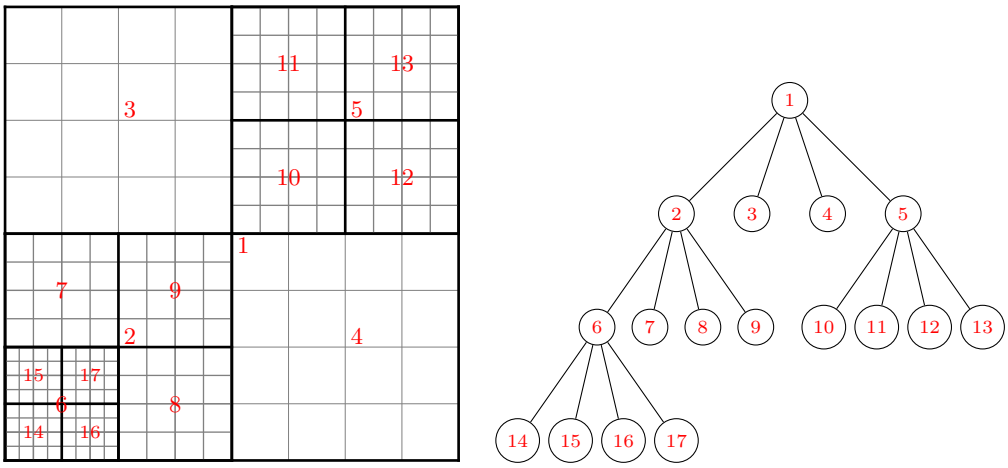


Figure 5.8: Mesh and data structure in PARAMESH.

adaptive meshes. It builds a hierarchy of sub-meshes to cover the computational domain, with spatial resolution varying to satisfy the demands of applications. These sub-mesh blocks form the nodes of a tree data structure (see Figure 5.8), and they are of small size, e.g., 4×4, to enable better load balance and to make mesh (de)refinement more convenient. Exploiting this data structure, we can easily represent a discretized matrix and its sparse approximate inverse on meshes at different levels. Moreover, as explained in Section 5.3, on the boundary between blocks of different mesh resolution, the discretization of a PDE is handled in a special fashion using ghost cells.

5.4.2 An Isotropic Diffusion Problem

The first model problem is an isotropic diffusion equation

$$u_t = \nabla \cdot (a \nabla u) \tag{5.18}$$

in the unit square $[0,1] \times [0,1]$, with a discontinuous coefficient a and the boundary conditions shown in Figure 5.9. The initial solution is $u = 0$ in the whole domain. We denote this problem as DIFF.

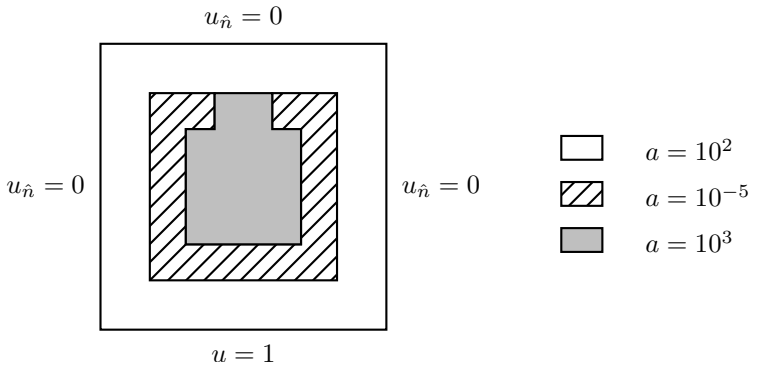


Figure 5.9: DIFF: diffusion problem $u_t = \nabla \cdot (a \nabla u)$ in the unit square $[0, 1] \times [0, 1]$ with a Dirichlet boundary condition on the bottom and a Neumann boundary condition on the other three sides.

Although this is a self-adjoint PDE, the discretization on nonuniform meshes results in a nonsymmetric linear system. This makes it inappropriate to use CG [37] as the solver here. Therefore, we choose BiCGStab [71], since it can deal with nonsymmetric systems and does not need matrix-vector multiplication with the matrix transpose.

We compare the sparsity pattern and the actual amount of work for SPAI, SPAI2 and MSPAI in Table 5.1. The algebraic sparsity pattern is measured by the number of nonzeros per column in the preconditioning operators when they are written explicitly in matrix form on the finest composite mesh. It indicates how much information the operator collects for a single point. The actual amount of work is measured by the average number of floating point operations done per column in a matrix vector product. We give the comparison for time steps 1 to 5, with the maximum refinement level increasing from 5 to 8. While MSPAI obtains almost full approximate inverse, it requires only about 4 times the work that the 5-point SPAI needs, and the cost of MSPAI is linearly scalable to the mesh size. This is because we collect the global information on coarse grids, where one cell represents a bunch of cells on the finest composite grid.

In Figure 5.10, we also compare the spectra of the different preconditioned systems – $\boldsymbol{A}$, $\boldsymbol{MA}$, $\boldsymbol{M}_2\boldsymbol{A}$ and $\boldsymbol{M}_m\boldsymbol{A}$ for a typical time step in the DIFF problem. Without preconditioning, the eigenvalues of the systems are spanned on the real line from 1 to 4053

Table 5.1: Comparison of algebraic sparsity pattern and actual work for SPAI, SPAI2 and MSPAI. Algebraic sparsity pattern is measured by the average number of nonzeros per column of the explicit preconditioning matrix; the actual work is measured by the average number of floating point operations done per column in a matrix vector product.

time step	$l_{\max}$	n	algebraic sparsity pattern (nnzpc)			actual work (#flop/n)		
			$\boldsymbol{M}$	$\boldsymbol{M}_2$	$\boldsymbol{M}_m$	SPAI	SPAI2	MSPAI
1	5	4096	4.9	24.1	4022	10	30	26.6
2	6	6112	5.5	55.8	6027	10	26.8	38.0
3	7	12448	5.7	204.3	12420	10	23.2	41.6
4	8	23488	5.9	601.0	23421	10	21.8	44.2
5	8	27136	5.9	636.4	27110	10	21.6	42.6

Table 5.2: Convergence results (number of iterations) for DIFF.

time step	1	2	3	4	5
$\ell_{\max}$	5	6	7	8	8
n	4096	6112	12448	23488	27136
NONE	864	681	1296	2309	2159
SPAI	118	111	164	235	265
SPAI2	82	72	83	85	85
MSPAI	17	18	16	17	19

with a few non-real eigenvalues due to the nonsymmetric AMR discretization. The basic SPAI rescales the system and moves a lot of the eigenvalues closer to 1. However, it still leaves a great many eigenvalues very close to the origin. A two-level correction gives us a better clustering of the eigenvalues at 1, while leaving fewer of them close to the origin. With the full multilevel SPAI, almost all the eigenvalues are laid in a small circle centered at 1 with a radius of about 0.15. Only a few eigenvalues are out of the circle, and they are further away from the origin as compared to the other preconditioners. This becomes easy for Krylov subspace methods, as they are efficient at handling a system with a good clustering of eigenvalues and a few sparsely separated ones.

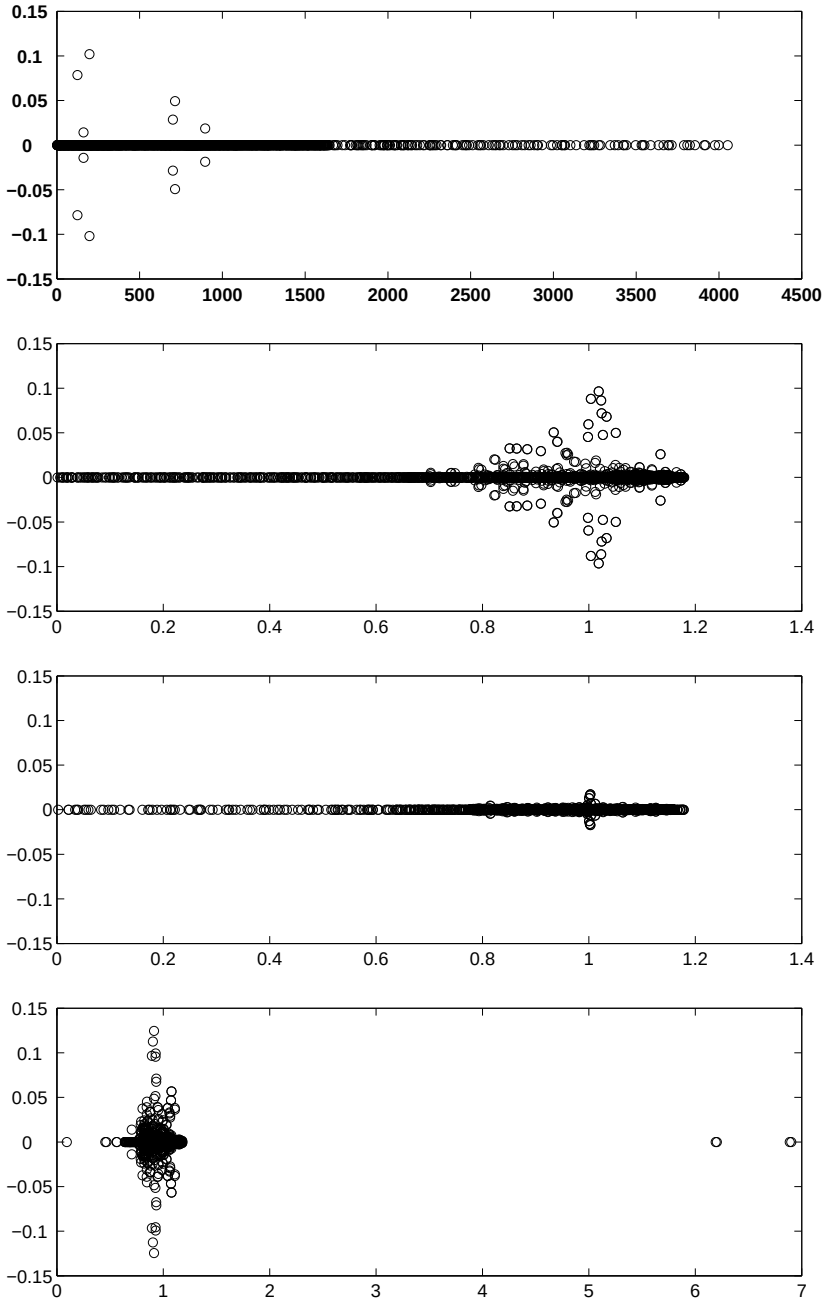


Figure 5.10: Spectra of $\boldsymbol{A}$, $\boldsymbol{MA}$, $\boldsymbol{M}_2\boldsymbol{A}$, and $\boldsymbol{M}_m\boldsymbol{A}$ for DIFF plotted in the complex plane (the horizontal axes correspond to the real numbers and the vertical axes correspond to the imaginary numbers).

Table 5.3: Results of MSPAI for different meshes at the 2nd time step of DIFF.

$\ell_{\max}$	n	niters	solver time
5	4096	16	0.92
6	6112	18	1.71
7	16096	14	3.26
8	36448	14	7.05

The convergence results for DIFF are listed in Table 5.2, where we set the convergence criterion as

$$\frac{\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k\|}{\|\boldsymbol{b}\|} < 10^{-12}. \tag{5.19}$$

MSPAI significantly reduces the number of iterations compared to standard SPAI and SPAI2. Furthermore, it yields level independent convergence rate, as the mesh is nonuniformly refined from level 5 to level 8 for time steps 1 to 5. However, the right hand sides we solve for different time steps are different. To give a fair comparison, in Table 5.3, we also list the number of iterations to solve the same (second) time step but on meshes with different maximum refinement levels. Multigrid methods are known to have a h-independent convergence rate, but they have difficulties handling problems with discontinuous coefficients, convection and strong anisotropy. Now, with the Krylov subspace method on top, our combination of the multilevel approach and the sparse approximate inverse preconditioners obtains level independent convergence rate for the DIFF problem with a discontinuous coefficient.

Moreover, Table 5.4 gives the timing results for the DIFF problem. Although MSPAI requires a small amount of extra work for multilevel correction, it reduces the overall solver time by a factor of 4 as compared to the standard SPAI, and it makes the computational cost scale linearly as the problem size increases.

Table 5.4: Timing results (seconds) for DIFF.

time step	1	2	3	4	5	total
NONE	9.85	11.64	43.71	146.28	157.79	369.27
SPAI	2.23	3.17	9.20	25.08	32.17	73.19
SPAI2	2.82	3.55	7.42	13.75	15.84	44.72
MSPAI	0.96	1.71	2.97	6.06	7.54	20.58
Update $\boldsymbol{M}$	0.12	0.58	0.21	0.32	0.11	1.34

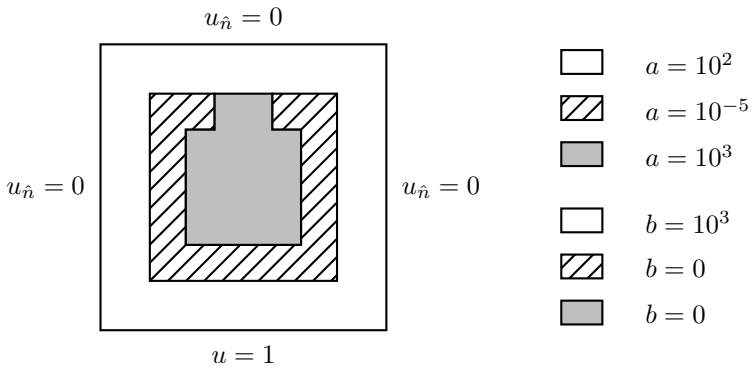


Figure 5.11: CONVECT: convection-diffusion problem $u_t = \nabla \cdot (a\nabla u) + bu_x$ in the unit square $[0,1] \times [0,1]$ with a Dirichlet boundary condition on the bottom and a Neumann boundary condition on the other three sides.

5.4.3 A Convection-Diffusion Problem

The second model problem is a convection-diffusion problem

$$u_t = \nabla \cdot (a\nabla u) + bu_x \tag{5.20}$$

in the unit square $[0,1] \times [0,1]$. The coefficient b is shown in Figure 5.11. The coefficient a, the boundary conditions, and the initial solution are the same as in DIFF. We denote this problem as CONVECT.

The spectra of the preconditioned systems on a typical adaptive mesh for the CONVECT problem are demonstrated in Figure 5.12. MSPAI shows similar improvement here as in the DIFF problem, although the complex magnitudes of the eigenvalues are slightly larger due

Table 5.5: Convergence and timing results for CONVECT.

time step	1	2	3	4	5
$\ell_{\max}$	5	6	7	8	8
n	4096	6208	12064	23056	32848
	convergence (niters)				
NONE	832	692	1270	3985	10051
SPAI	140	125	169	249	342
SPAI2	90	80	84	98	92
MSPAI	22	18	19	21	21
	timing (secs)				
NONE	9.47	12.00	41.88	248.78	882.01
SPAI	2.65	3.62	9.23	25.83	49.64
SPAI2	3.08	3.97	7.30	15.55	19.95
MSPAI	1.25	1.72	3.46	7.35	9.83
Update $\boldsymbol{M}$	0.14	0.69	0.22	0.36	0.32

to the asymmetry from the convection term.

We list the convergence and timing results for the CONVECT problem in Table 5.5. They show similar improvement as that in the DIFF problem. The multilevel sparse approximate inverse preconditioner significantly reduces the number of the iterations and the overall computational time. Moreover, it achieves level independent convergence rate.

5.4.4 An Anisotropic Diffusion Problem

Anisotropic problems are in general hard for multigrid solvers. So we choose an anisotropic diffusion problem to demonstrate the capability of our multilevel preconditioner. We solve a discontinuous anisotropic problem

$$u_t = au_{xx} + bu_{yy} + f \tag{5.21}$$

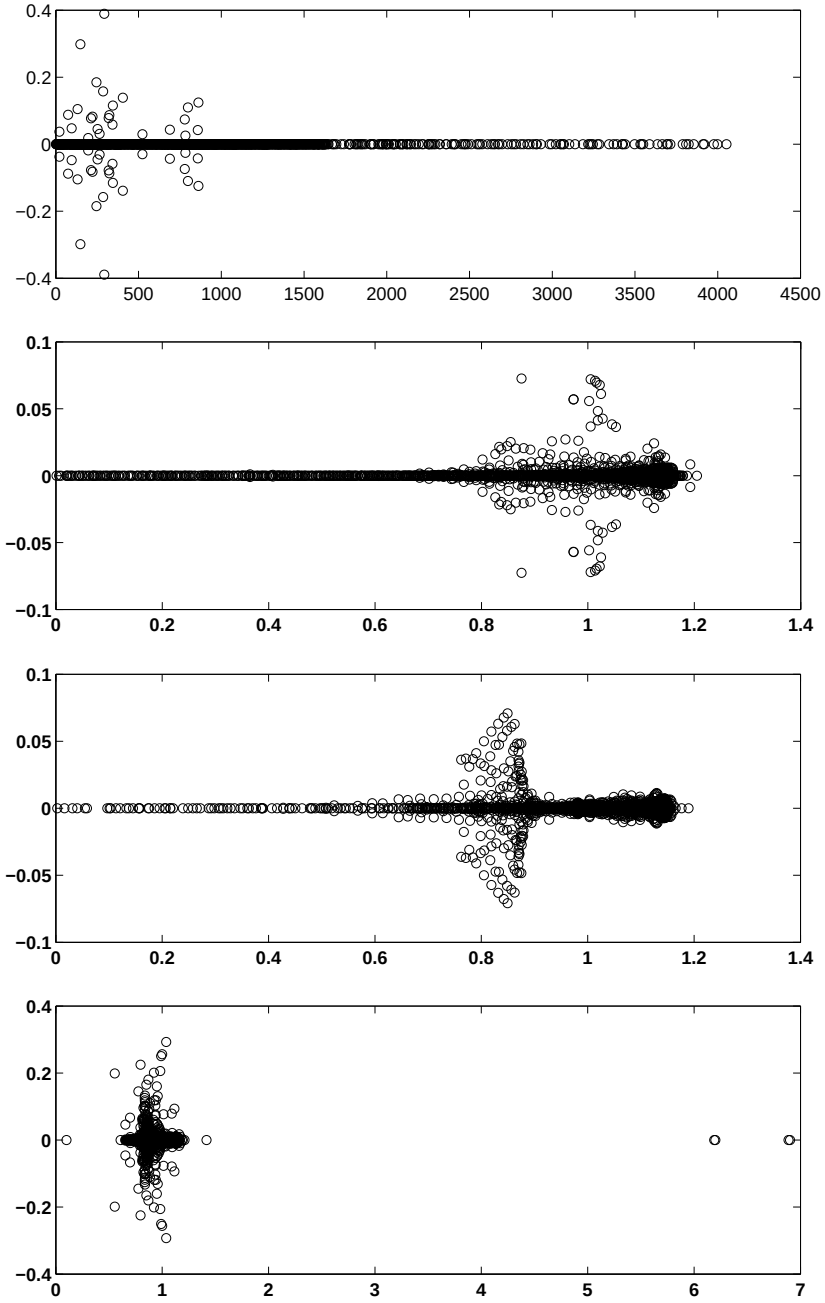


Figure 5.12: Spectra of $\boldsymbol{A}$, $\boldsymbol{MA}$, $\boldsymbol{M}_2\boldsymbol{A}$, and $\boldsymbol{M}_m\boldsymbol{A}$ for CONVECT plotted in complex planes (the horizontal axes correspond to the real numbers and the vertical axes correspond to the imaginary numbers).

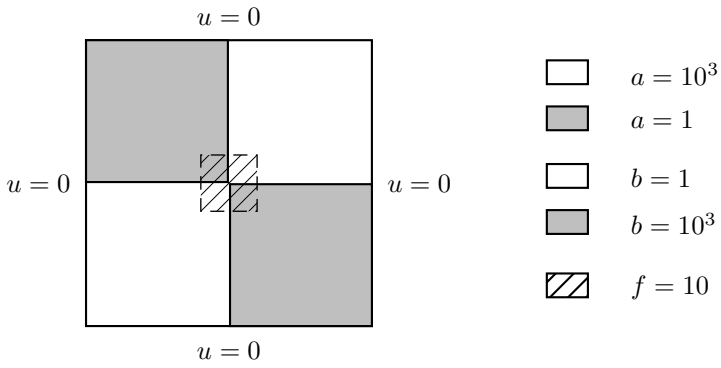


Figure 5.13: ANISO: anisotropic problem $u_t = au_{xx} + bu_{yy} + f$ in the unit square $[0, 1] \times [0, 1]$ with a homogeneous boundary condition on all sides.

in the unit square, with a homogeneous boundary condition. The coefficients a and b are shown in Figure 5.13. The source function f is 10 in the shadowed area $[0.4, 0.6] \times [0.4, 0.6]$ and zero anywhere else. The initial solution is set as $u_0 = 0$ again. We denote this problem as ANISO. For this problem, we choose a 5-point sparsity pattern with all 5 points lie on the direction of strong anisotropy.

We give the convergence and timing results for ANISO in Table 5.6. Again, the MSPAI preconditioner greatly reduces the number of iterations for the solver and improves the overall performance. Moreover, it achieves level-independent convergence rate for the anisotropic diffusion problem too.

5.5 MSPAI for Topology Optimization

The finite element analysis in compliance minimization problems leads to linear elasticity systems. Figure 5.14 shows the Green's function for a point source for a 2D elasticity problem. The problem is defined on the unit square $[0, 1] \times [0, 1]$ with homogeneous material properties and homogeneous boundary conditions on all four sides. The Green's function now has x and y components. The point source (loading) is in the x direction and in the center of the domain. The most significant values in the Green's function, for both x and y directions,

Table 5.6: Convergence and timing results for ANISO.

time step	1	2	3	4	5
$\ell_{\max}$	4	5	6	7	8
n	4096	12544	28672	42496	48640
	convergence (niters)				
NONE	140	312	558	1271	2563
SPAI	54	127	291	306	430
SPAI2	31	44	77	156	156
MSPAI	28	40	34	36	37
	timing (secs)				
NONE	1.17	7.30	30.07	100.25	227.71
SPAI	0.79	5.14	26.58	41.51	65.65
SPAI2	0.89	2.86	9.91	29.33	33.49
MSPAI	1.09	3.89	7.25	13.19	19.20
Update $\boldsymbol{M}$	0.385	0.793	1.516	1.298	0.793

are in the neighborhood of the point source. A SPAI preconditioner can capture the spikes in the Green's function with a relatively small sparsity pattern. Unfortunately, the Green's function for our elasticity problem is smooth with slowly decaying global components, similar to diffusion problems. We derive the 2D Green's function in continuum setting using the Fourier transform. It has the following form:

$$\boldsymbol{g}(x,y;x_0,y_0) = \begin{pmatrix} \sum_{j,k\geq 1} a_{j,k} \sin j\pi x \sin k\pi y \\ \sum_{j,k\geq 1} b_{j,k} \sin j\pi x \sin k\pi y \end{pmatrix}, \tag{5.22}$$

where

$$a_{j,k} = \frac{k^2 + \frac{1-\nu}{2} j^2}{\frac{1-\nu}{2}(j^2+k^2)^2} \cdot \frac{4(1-\nu^2)}{\pi^2 E} \sin j\pi x_0 \sin k\pi y_0, \tag{5.23}$$

$$b_{j,k} = \frac{-jk}{\frac{1-\nu}{2}(j^2+k^2)^2} \cdot \frac{4(1-\nu^2)}{\pi^2 E} \sin j\pi x_0 \sin k\pi y_0, \tag{5.24}$$

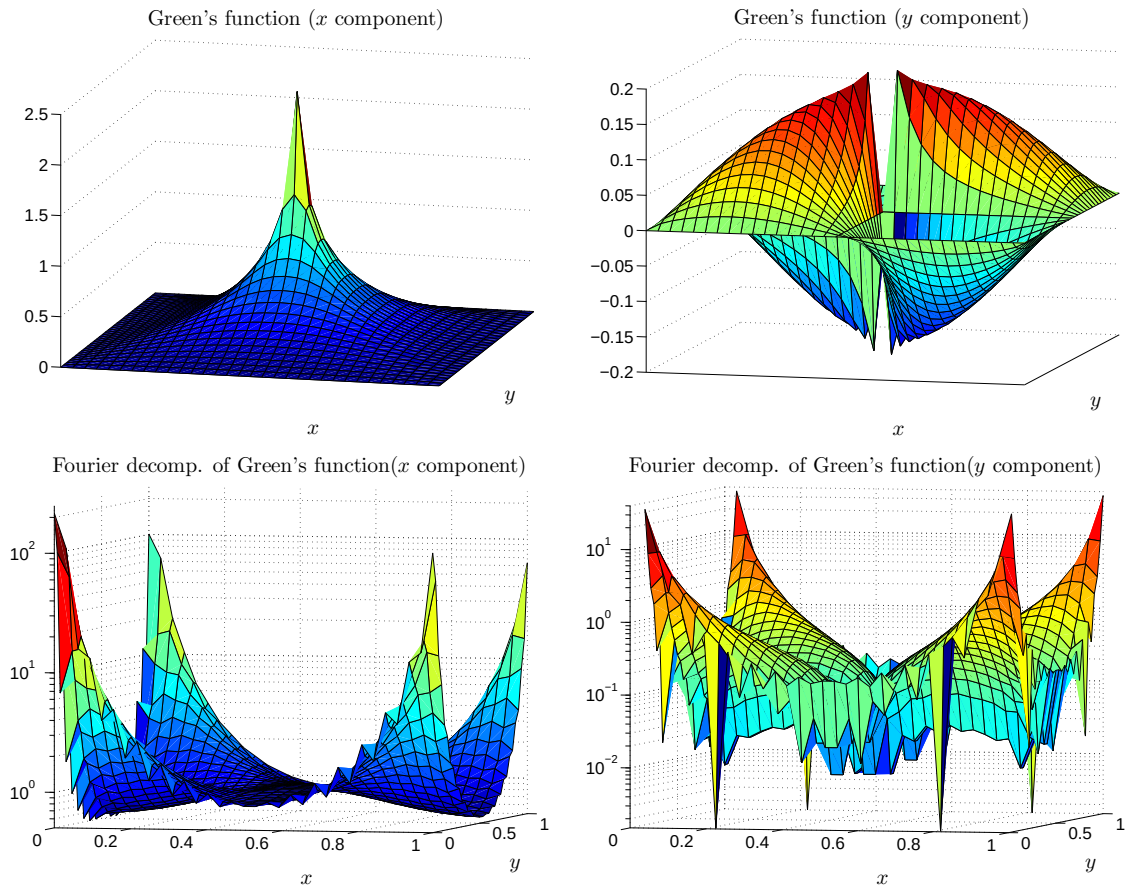


Figure 5.14: Green's function and its Fourier decomposition for a 2D elasticity problem on the unit square with homogeneous material properties and homogeneous boundary conditions. The point source is in the x direction and in the center of the domain. The low-frequency components are at the corners of the frequency domain, and the high-frequency components are in the middle of the frequency domain.

E is the Young's modulus, and ν is the Poisson's ratio. The coefficients $a_{j,k}$ and $b_{j,k}$ indicate that the Green's function has larger low-frequency components and smaller high-frequency components. These relatively large low-frequency components can not be captured by standard SPAI while still maintaining a small sparsity pattern (see Figure 5.15). To remedy this problem, we use multilevel techniques to improve the approximation to the low-frequency modes. In the following section, we introduce a few extensions to the multilevel SPAI preconditioner for topology optimization problems.

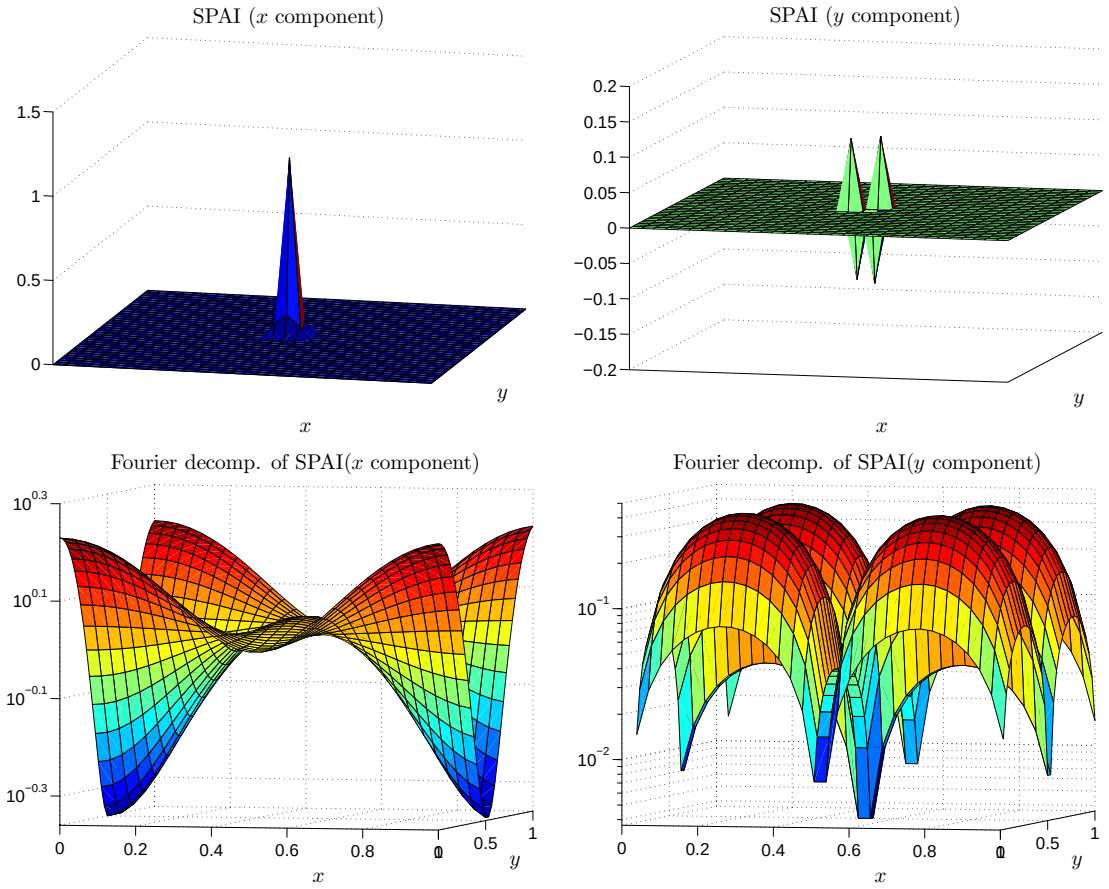


Figure 5.15: SPAI and its Fourier decomposition for a 2D elasticity problem on the unit square with homogeneous material properties and homogeneous boundary conditions. The point source is in the x direction and in the center of the domain. The low-frequency components are at the corners of the frequency domain, and the high-frequency components are in the middle of the frequency domain.

5.5.1 Adaptation for Topology Optimization

Although topology optimization shares some important properties with diffusion problems, there are a few more things we need to consider. First, unlike the diffusion problems we discussed before, topology optimization is a nonlinear process. The differential operator changes, because it is a function of the changing design variables. Therefore, even though we do not update the mesh at every optimization step as discussed in Section 2.3.1, we still need to update the sparse approximate inverse due to the evolution of the design variables. However, the design variables change slowly, especially towards the end of the optimization. Many times, there is no change in most of the simulation domain, particularly in the regions away from the solid/void interface. For those regions, the stiffness matrix will not change. Hence, we can limit the update of the sparse approximate inverse to the regions where the design variables do change from the previous optimization step. To be even more economical, we can postpone updating the columns of the sparse approximate inverse until the accumulated change of their neighboring design variables exceeds a given threshold. When this happens, we update these columns and reset the accumulated change of the corresponding design variables to zero. This further reduces the cost of updating the sparse approximate inverse. We can control the accuracy of the SPAI and the cost of its update by tuning the threshold. A higher threshold leads to less frequent updates and less accurate sparse approximate inverses. We call this approach *delayed updating*.

Second, as we have discussed in Chapter 3, for topology optimization problems, it is essential to rescale the system matrix before computing the preconditioner and solving the system. The residual and the error of the original system and the rescaled system are defined with respect to different scalings. On a multilevel mesh, system matrices on different levels may be rescaled differently. So, in order to make a proper multilevel correction, we need to incorporate rescaling into the multilevel sparse approximate inverse.

Let $\boldsymbol{A}_h$ be defined on a fine mesh and $\boldsymbol{A}_H$ on a coarse mesh. We rescale them by their

respective diagonals as

$$\widetilde{\boldsymbol{A}}_h = \boldsymbol{D}_h^{-1/2}\boldsymbol{A}_h\boldsymbol{D}_h^{-1/2}, \qquad \widetilde{\boldsymbol{A}}_H = \boldsymbol{D}_H^{-1/2}\boldsymbol{A}_H\boldsymbol{D}_H^{-1/2}. \tag{5.25}$$

Now we solve a rescaled system $\widetilde{\boldsymbol{A}}_h\tilde{\boldsymbol{x}}_h = \tilde{\boldsymbol{b}}$. Therefore, in our multilevel SPAI preconditioner, we need to approximate $\widetilde{\boldsymbol{A}}_h^{-1}$ instead of $\boldsymbol{A}_h^{-1}$ on both the fine mesh and the coarse mesh. At both level, we compute the sparse approximate inverses $\widetilde{\boldsymbol{M}}_h$ and $\widetilde{\boldsymbol{M}}_H$ for the rescaled system matrices:

$$\widetilde{\boldsymbol{M}}_h \approx \widetilde{\boldsymbol{A}}_h^{-1}, \qquad \widetilde{\boldsymbol{M}}_H \approx \widetilde{\boldsymbol{A}}_H^{-1}. \tag{5.26}$$

We can approximate $\widetilde{\boldsymbol{A}}_h^{-1}$ on the coarse level as follows:

$$\begin{aligned}
\widetilde{\boldsymbol{A}}_h^{-1} &= \boldsymbol{D}_h^{1/2}\boldsymbol{A}_h^{-1}\boldsymbol{D}_h^{1/2} \\
&\approx \boldsymbol{D}_h^{1/2}\boldsymbol{I}_H^h\boldsymbol{A}_H^{-1}\boldsymbol{I}_h^H\boldsymbol{D}_h^{1/2} \\
&= \boldsymbol{D}_h^{1/2}\boldsymbol{I}_H^h\boldsymbol{D}_H^{-1/2}\widetilde{\boldsymbol{A}}_H^{-1}\boldsymbol{D}_H^{-1/2}\boldsymbol{I}_h^H\boldsymbol{D}_h^{1/2} \\
&\approx \boldsymbol{D}_h^{1/2}\boldsymbol{I}_H^h\boldsymbol{D}_H^{-1/2}\widetilde{\boldsymbol{M}}_H\boldsymbol{D}_H^{-1/2}\boldsymbol{I}_h^H\boldsymbol{D}_h^{1/2}.
\end{aligned} \tag{5.27}$$

If we define

$$\widetilde{\boldsymbol{I}}_H^h = \boldsymbol{D}_h^{1/2}\boldsymbol{I}_H^h\boldsymbol{D}_H^{-1/2}, \quad \widetilde{\boldsymbol{I}}_h^H = \boldsymbol{D}_H^{-1/2}\boldsymbol{I}_h^H\boldsymbol{D}_h^{1/2}, \tag{5.28}$$

(5.27) can be written as

$$\widetilde{\boldsymbol{A}}_h^{-1} \approx \widetilde{\boldsymbol{I}}_H^h\widetilde{\boldsymbol{M}}_H\widetilde{\boldsymbol{I}}_h^H. \tag{5.29}$$

This implies that we can incorporate the difference in scaling at different levels into the mapping operators.

In Algorithm 5.1, we replace the system matrices $\boldsymbol{A}_h$ and $\boldsymbol{A}_H$ with the rescaled matrices $\widetilde{\boldsymbol{A}}_h$ and $\widetilde{\boldsymbol{A}}_H$, the sparse approximate inverses $\boldsymbol{M}_h$ and $\boldsymbol{M}_H$ with $\widetilde{\boldsymbol{M}}_h$ and $\widetilde{\boldsymbol{M}}_H$, and the mapping operators $\boldsymbol{I}_h^H$ and $\boldsymbol{I}_H^h$ with the modified $\widetilde{\boldsymbol{I}}_h^H$ and $\widetilde{\boldsymbol{I}}_H^h$ defined in (5.28). This leads to a two-level sparse approximate inverse preconditioner for $\widetilde{\boldsymbol{A}}_h$. Just as in Algorithm 5.1,

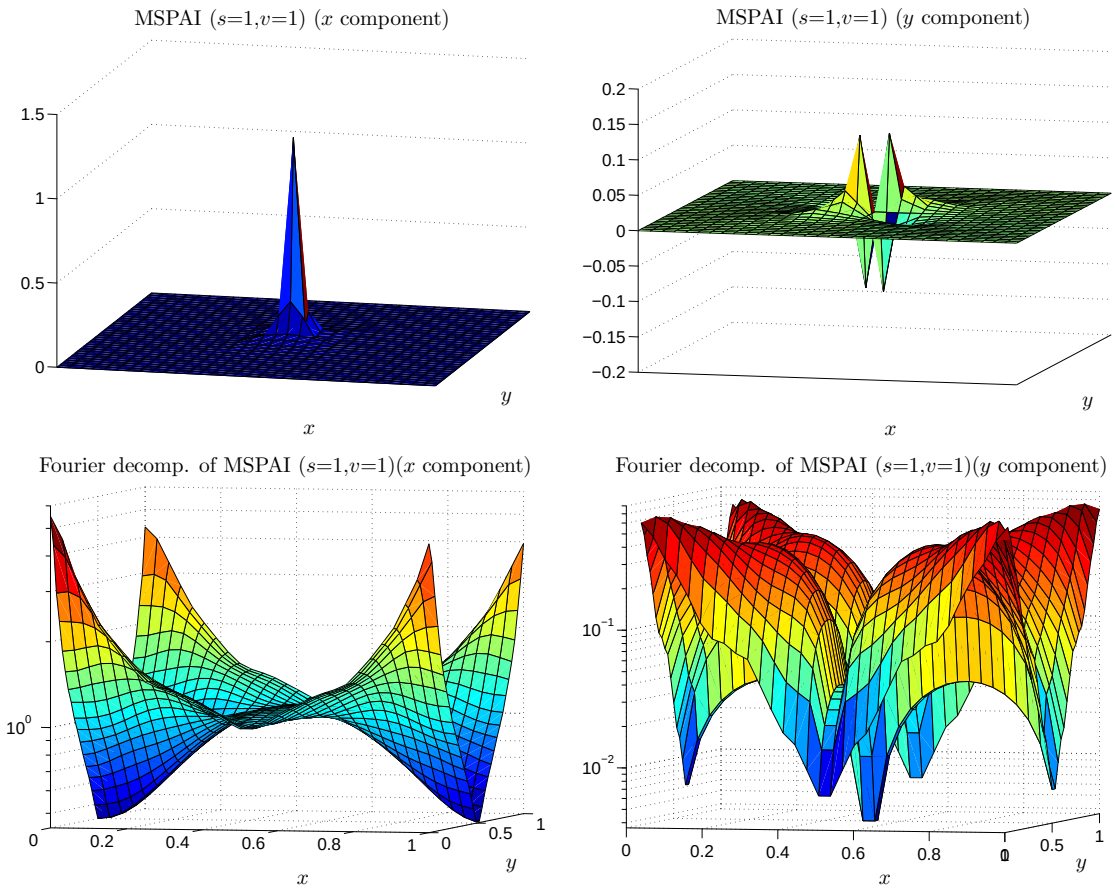


Figure 5.16: MSPAI ($v = 1, s = 1$) and its Fourier decomposition for a 2D elasticity problem on the unit square with homogeneous material properties and homogeneous boundary conditions. The point source is in the x direction and in the center of the domain. The low-frequency components are at the corners of the frequency domain, and the high-frequency components are in the middle of the frequency domain.

a recursive call at step 4 makes it a multilevel SPAI preconditioner of $\widetilde{\boldsymbol{A}}_h$, which we denote as $\widetilde{\boldsymbol{M}}_m$.

Last, we notice that the multilevel SPAI preconditioner proposed for the scalar problems is not accurate enough for elasticity problems. Figure 5.16 shows the MSPAI and its Fourier decomposition for the 2D the elasticity problem on the unit square with homogeneous material properties and homogeneous boundary conditions on all the sides. Compared with Figures 5.14 and 5.15, this MSPAI is improved over the SPAI, but still has noticeable errors in the low frequency modes. Therefore, to improve the accuracy of the MSPAI for elasticity

problems, we carry out iterative refinement at the same level and perform multiple coarse level corrections. This leads to Algorithm 5.2. We denote as s the number of same level corrections, which can be viewed as multiple smoothing steps in multigrid solvers. We denote as v the number of coarse level corrections, which can be viewed as multiple V-cycles, although we do not solve exactly at the coarsest level. Setting the parameters $s = 0$ and $v = 1$, we get our original two-level SPAI method. Similarly, a recursive call at step 9 in Algorithm 5.2 makes this algorithm a multilevel scheme. Figure 5.17 shows the MSPAI with one step of iterative refinement and two coarse level corrections for a point source for the same elasticity problem as in Figures 5.14, 5.15, and 5.16.

Algorithm 5.2: TWO-LEVEL SPAI WITH RESCALING AND MULTIPLE CORRECTIONS: Compute $\boldsymbol{y} \leftarrow \widetilde{\boldsymbol{M}}_m \boldsymbol{z}$

1 $\boldsymbol{y} \leftarrow \widetilde{\boldsymbol{M}}_h \boldsymbol{z}$;
2 **for** $i = 1$ *to* s **do** /* iterative refinement */
3 $\quad \boldsymbol{r}_h \leftarrow \boldsymbol{z} - \widetilde{\boldsymbol{A}}_h \boldsymbol{y}$;
4 $\quad \boldsymbol{y} \leftarrow \boldsymbol{y} + \widetilde{\boldsymbol{M}}_h \boldsymbol{r}$;
5 **end**
6 **for** $i = 1$ *to* v **do** /* multiple coarse level corrections */
7 $\quad \boldsymbol{r}_h \leftarrow \boldsymbol{z} - \widetilde{\boldsymbol{A}}_h \boldsymbol{y}$;
8 $\quad \boldsymbol{r}_H \leftarrow \widetilde{\boldsymbol{I}}_h^H \boldsymbol{r}_h$;
9 $\quad \boldsymbol{e}_H \leftarrow \widetilde{\boldsymbol{M}}_H \boldsymbol{r}_H$;
10 $\quad \boldsymbol{e}_h \leftarrow \widetilde{\boldsymbol{I}}_H^h \boldsymbol{e}_H$;
11 $\quad \boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{e}$;
12 **end**

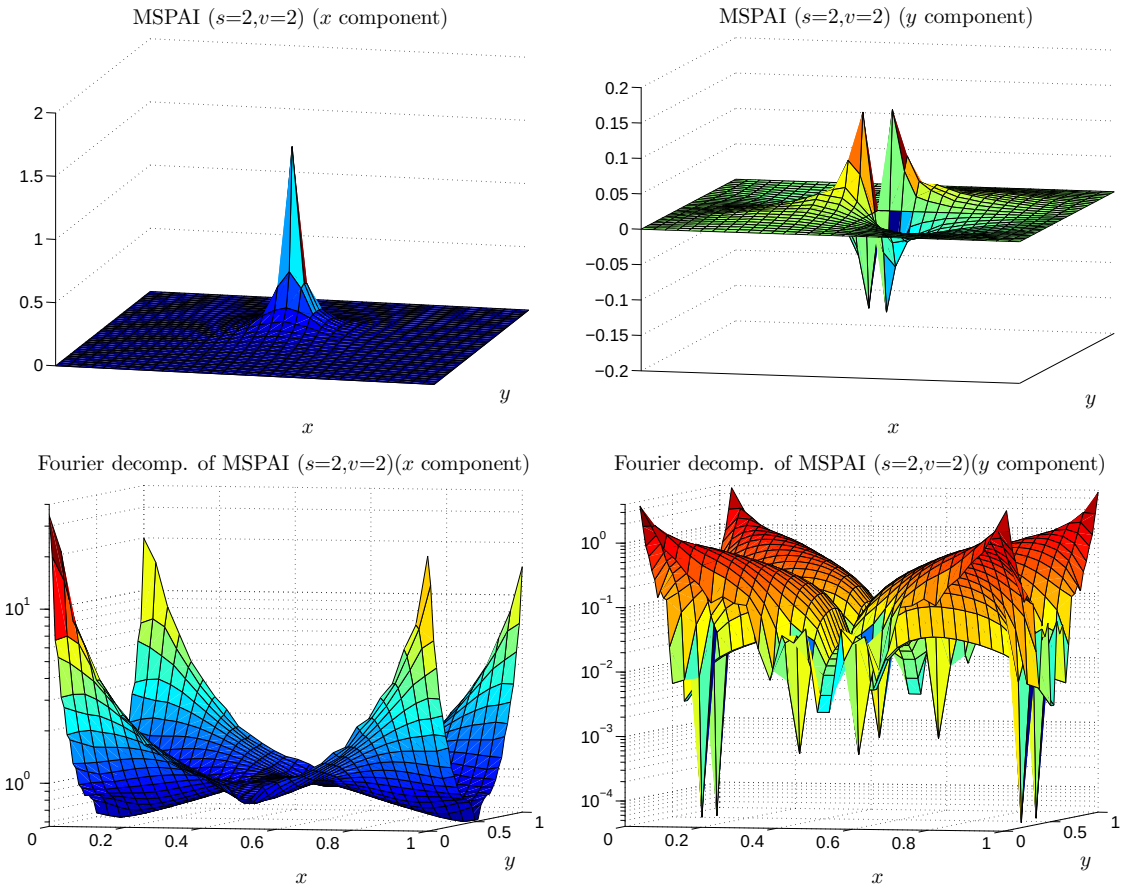


Figure 5.17: MSPAI ($v = 2, s = 2$) and its Fourier decomposition for a 2D elasticity problem on the unit square with homogeneous material properties and homogeneous boundary conditions. The point source is in the x direction and in the center of the domain. The low-frequency components are at the corners of the frequency domain, and the high-frequency components are in the middle of the frequency domain.

5.6 Numerical Experiments with Topology Optimization Problems

5.6.1 Implementation with `libMesh` and `PETSc`

We use `libMesh` [41] to support finite element analysis and adaptive mesh refinement, and `PETSc` [4, 5] for linear solvers and preconditioners. We build our multilevel sparse approximate inverse preconditioner based on the multigrid solver in `PETSc`. Due to the limitations of these libraries, we need to explicitly build our mapping operators between fine and coarse meshes in matrix form in `libMesh`, and pass them to `PETSc`. The most efficient way would be to carry out the mapping operations on the mesh directly in `libMesh`. Again, because of the asymmetry in the multilevel SPAI preconditioner, we choose to use BiCGStab solver. `PETSc` has a BiCGStab implementation.

There are various ways to handle Dirichlet boundary conditions. The default way in `libMesh` is to add a large diagonal coefficient to all the fixed DOFs. This is called penalization. Another way is to set the submatrix corresponding to the fixed degrees of freedom to an identity matrix. Since the first approach would interfere with the diagonal scaling in our multilevel SPAI preconditioner, we modify the `libMesh` code to adopt the second approach of handling Dirichlet boundary conditions.

To make delayed updating possible, we need more control over the code. Therefore, we implemented SPAI in `PETSc` ourselves, even though `PETSc` has an implementation of SPAI, which is provided by the `SPAI` package from Universität Basel [34] and follows the method in [35]. However, our SPAI code is not optimized in terms of performance.

For delayed updating of SPAI, we update a column only after the accumulated change of any of its neighboring design variables exceeds a given threshold. However, since `libMesh` uses `PETSc` to solve linear systems and `PETSc` does not see the mesh information, every time we do mesh refinement, we need to recalculate the sparse approximate inverse at each level.

If we had better integration between `libMesh` and `PETSc`, we could have had more efficient implementation for SPAI updates during mesh refinement, like what we have done with the `PARAMESH` package.

5.6.2 Numerical Results and Discussion

As we have seen, for the convection-diffusion problems, our multilevel sparse approximate inverse preconditioner leads to level independent convergence rates for the iterative solver. This makes the computational cost linear in problem size, which is a major advantage of this preconditioner. However, the elasticity problems with large jumps in elasticity tensors that arise in topology optimization are in general harder problems. We list the results of some convergence tests for 2D and 3D elasticity problems in Tables 5.7 and 5.8. We use BiCGStab as the solver, since the preconditioned system is not symmetric even though the elastic problem is symmetric. We compare the number of iterations using both the MSPAI preconditioner and the incomplete Cholesky preconditioner.

We use α to measure the average growth rate of number of iterations in terms of the problem size n. To be specific, the number of iterations is $O(n^{\alpha})$ when the number of unknowns n increases. Ideally, we would like to see $\alpha = 0$, which means the number of iterations does not grow as problem size increases. For topology optimization, although MSPAI does not lead to level independent convergence rate, a proper choice of s and v makes the number of iterations close to level independent. Increasing s (the number of same level corrections) improves the convergence rate, but does not affect the asymptotic convergence rate α much. Whereas, increasing v improves the convergence rate asymptotically. Compared with the incomplete Cholesky preconditioner, MSPAI with multiple corrections improves the convergence rate significantly.

For a 2D elasticity problem demonstrated in Figure 2.8(a) with homogeneous material properties, we analyze the spectra of the unpreconditioned and preconditioned systems. The problem is discretized on a 24×16 mesh. Figure 5.18(a) shows the spectrum of the rescaled

Table 5.7: Level dependency test of MSPAI for the 2D elasticity problem shown in Figure 2.8(a) with homogeneous density distribution on uniform meshes. The numbers of iterations for the `PETSc` BiCGStab solver with our multilevel SPAI preconditioner and with the IC preconditioner are listed. The multilevel recursive calls always go down to level 0. $\ell_{\max}$ is the level of the finest mesh. n is the number of nonzeros. α is the average growth rate of the number of iterations in terms of the number of unknowns, i.e., niters $\sim O(n^{\alpha})$.

$\ell_{\max}$	mesh	n	niters with MSPAI										niters with IC
			$s=1$				$s=2$			$s=3$			
			$v=0$	$v=1$	$v=2$	$v=3$	$v=1$	$v=2$	$v=3$	$v=1$	$v=2$	$v=3$	
0	8×4	90	22	22	22	22	14	14	14	12	12	12	15
1	16×8	306	49	34	28	26	22	19	18	19	16	16	24
2	32×16	1122	97	60	42	33	39	27	22	32	24	20	46
3	64×32	4290	196	129	66	52	64	39	30	61	35	25	93
4	128×64	16770	402	299	126	55	121	54	37	101	50	31	214
α			0.556	0.499	0.334	0.175	0.413	0.258	0.186	0.408	0.273	0.186	0.508

Table 5.8: Level dependency test of MSPAI for the 3D elasticity problem shown in Figure 2.11(a) with homogeneous density distribution on uniform meshes. The numbers of iterations for the `PETSc` BiCGStab solver with our multilevel SPAI preconditioner and with the IC preconditioner are listed. The multilevel recursive calls always go down to level 0. $\ell_{\max}$ is the level of the finest mesh. n is the number of nonzeros. α is the average growth rate of the number of iterations in terms of the number of unknowns, i.e., niters $\sim O(n^{\alpha})$.

$\ell_{\max}$	mesh	n	niters with MSPAI										niters with IC
			$s=1$				$s=2$			$s=3$			
			$v=0$	$v=1$	$v=2$	$v=3$	$v=1$	$v=2$	$v=3$	$v=1$	$v=2$	$v=3$	
0	$8 \times 2 \times 2$	243	24	24	24	24	14	14	14	12	12	12	12
1	$16 \times 4 \times 4$	1275	56	43	39	32	26	23	19	23	19	17	23
2	$32 \times 8 \times 8$	8019	129	84	65	52	53	45	39	50	39	36	41
3	$64 \times 16 \times 16$	56355	244	191	116	83	103	74	51	90	72	53	84
4	$128 \times 32 \times 32$	421443	487	350	198	130	191	123	82	192	124	81	184
α			0.404	0.359	0.283	0.227	0.350	0.291	0.237	0.372	0.313	0.256	0.366

linear system. The eigenvalues are almost uniformly spread from 0 to 1.8 on the real axis, while a lot of them are very close to the origin. Figure 5.18(b) shows the spectrum of the rescaled system left preconditioned by SPAI. The eigenvalues form a cluster around 1, but there are still a large number of them spread densely from 0 to 1 and a few of them very close to the origin. Note that some of the eigenvalues are no longer real because the left preconditioning yields a nonsymmetric system. If we precondition the system using our MSPAI with only one coarse level correction and no iterative refinement on the same level, the spectrum of the preconditioned system is shown in Figure 5.18(c). Now, we have a cluster of more eigenvalues around 1 and much fewer eigenvalues close to the origin. If we use two coarse level corrections and one step of iterative refinement on the same level, the spectrum of the preconditioned system as shown in Figure 5.18(d) is further improved. More eigenvalues are clustered more closely at 1. All eigenvalues are away from the origin except for one, which is not so close compared to the smallest eigenvalues of the previous two preconditioned systems.

Now, we apply our multilevel SPAI preconditioner for a real topology optimization problem on adaptive meshes. To analyze the convergence rate for MSPAI preconditioned BiCGStab, we turn off the continuation on the material penalization parameter and the continuation on the convergence tolerance. We solve the 3D cantilever beam problem shown in Figure 2.4. Figure 5.19 compares the MSPAI results with and without delayed updating. Delayed updating causes only slight deterioration in the convergence rate, but it reduces the cost of computing the approximate inverses significantly, and thus improves the overall performance. The spikes in Figures 5.19 (c) and(d) indicate the optimization steps when adaptive mesh refinement happens. As we have mentioned in Section 5.6.1, we recalculate the approximate inverses from scratch after mesh refinement, because `libMesh` may change the ordering of the DOFs and `PETSc` has no access to that information. A better integration of the meshing package and the solver package, with proper data structures that connects the mesh and the matrices, may further reduce the cost of computing the sparse approximate

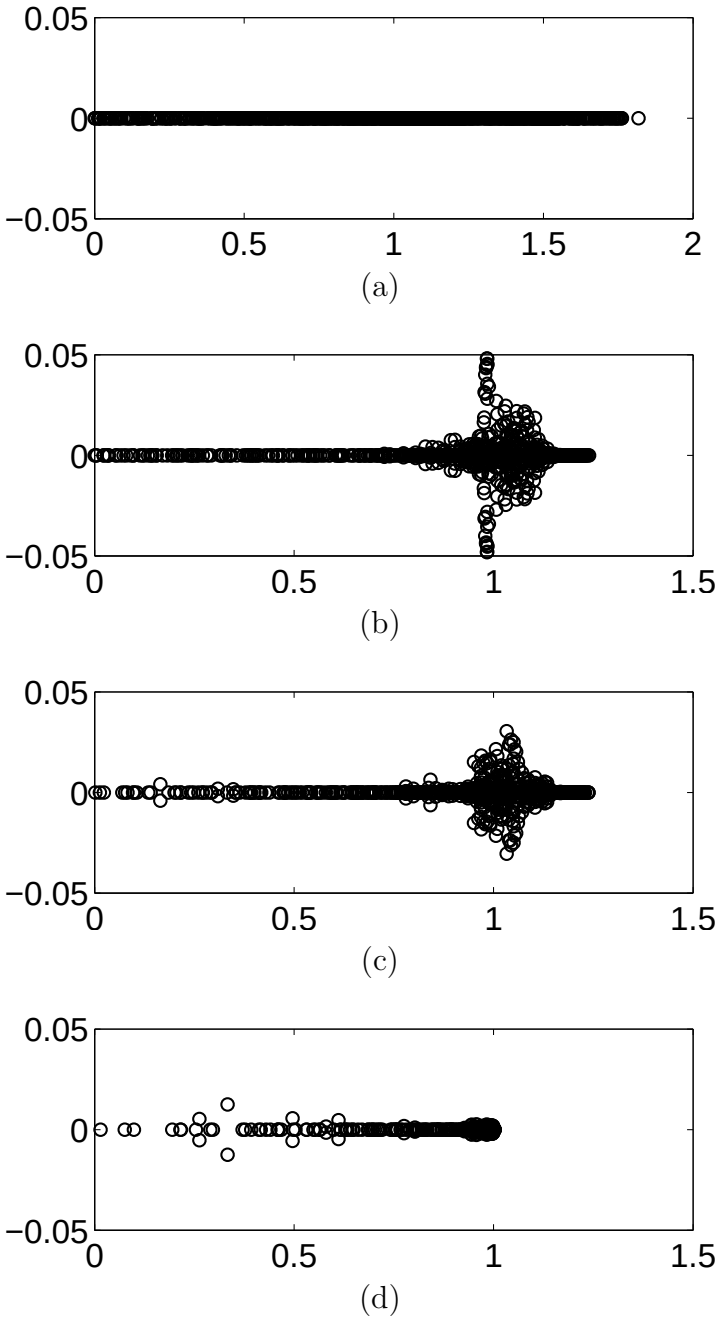


Figure 5.18: Spectra of the unpreconditioned and preconditioned system for a elasticity system plotted in the complex plane (the horizontal axes correspond to the real numbers and the vertical axes correspond to the imaginary numbers). (a) spectrum of the rescaled system; (b) spectrum of the rescaled system preconditioned by SPAI; (c) spectrum of the rescaled system preconditioned by MSPAI with one coarse level correction and no iterative refinement at the same level; (d) spectrum of the rescaled system preconditioned by MSPAI with two coarse level corrections and one step of iterative refinement at the same level.

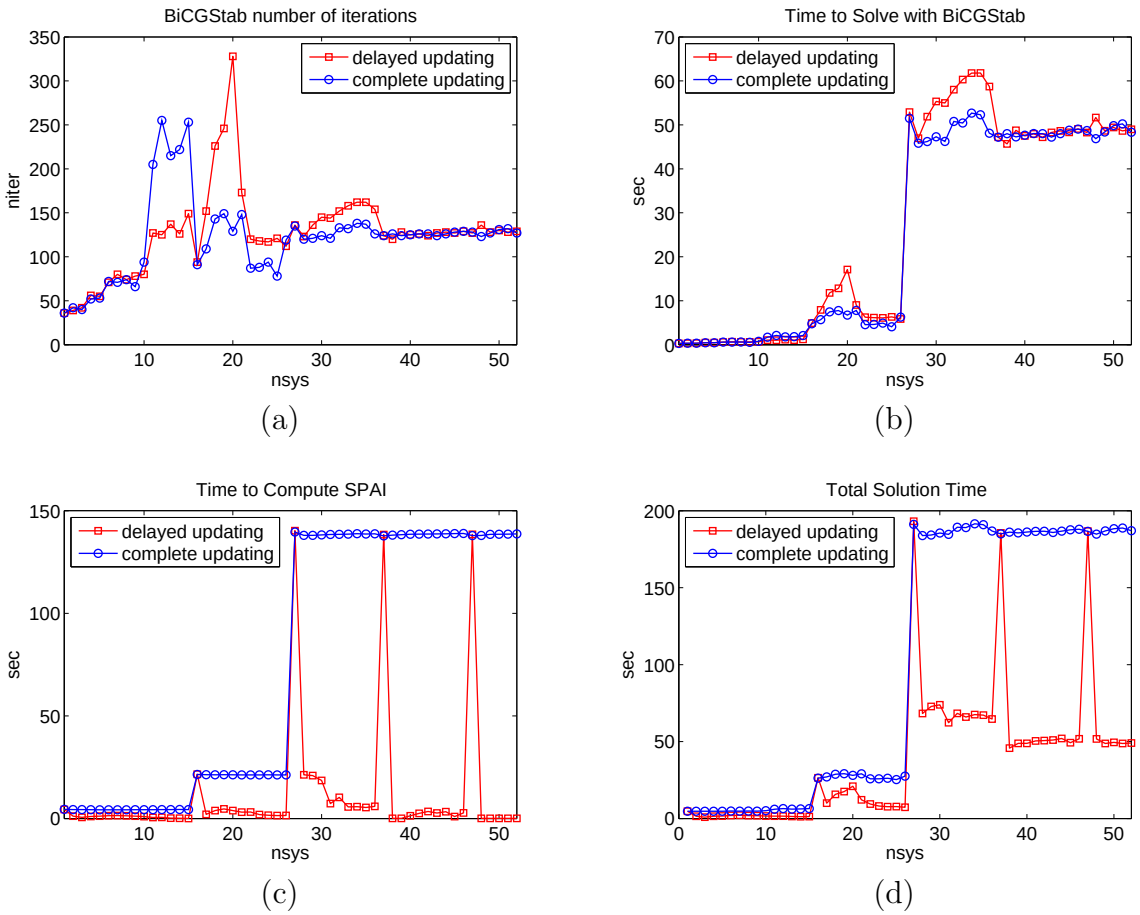


Figure 5.19: Delayed update of SPAI: (a) convergence of BiCGStab with MSPAI (number of iterations); (b) time to solve the linear system (seconds); (c) time to update/compute SPAIs on all levels (seconds); (d) total time (pre-solve and solve).

inverses at the time of mesh refinement.

Chapter 6

Conclusions

This thesis focused on the computational aspects of topology optimization. In the following section, we review the major contributions, many of which are not limited to topology optimization, but are also useful for other simulation and PDE problems. In Section 6.2, we discuss future work.

6.1 Contributions

Recycling MINRES method We adapted the Krylov subspace recycling idea with the MINRES method to solve sequences of symmetric systems. We preserved the short-term recurrence and made the subspace selection significantly cheaper by exploiting symmetry and choosing a proper subspace to recycle from. The recycling MINRES method greatly improves the convergence rate and reduces the overall computational cost compared to MINRES, which is optimal for a single symmetric system.

Multilevel sparse approximate inverse preconditioner For simulations on a dynamic mesh, the linear system changes constantly due to the change in the mesh. We proposed a multilevel sparse approximate inverse preconditioner (MSPAI) that is inexpensive to update when the mesh or the system change only locally. Experiments demonstrate that this preconditioner results in level independent convergence rate for convection-diffusion types of problems.

Extensions of MSPAI for topology optimization We adapted the MSPAI preconditioner for elasticity problems. We allow iterative refinement and multiple coarse level corrections in order to obtain a better preconditioner. We also incorporate the rescaling into the multilevel scheme. Experiments show that with these extensions our MSPAI preconditioner achieves nearly level-independent convergence rate for the elasticity problems arising in topology optimization. Therefore it scales much better in terms of convergence rate and total computational cost compared with the standard incomplete Cholesky preconditioner. We also proposed delayed update for topology optimization to reduce the cost of computing the preconditioner. The update of the preconditioner remains inexpensive for the constant but slow change of the design variables.

AMR schemes for topology optimization We proposed a new AMR scheme for topology optimization that makes mesh refinement truly dynamic and adaptive. Our AMR scheme provides room for the design to change and prevents the final design from being confined by the mesh resulting from designs on coarser meshes.

Analysis of scaling issues in topology optimization We analyzed the conditioning of the linear system in topology optimization and showed that the extreme ill-conditioning is largely due to the bad scaling from the solid/void density ratio. We further demonstrated that a proper rescaling reduces the huge condition numbers typical in topology optimization to roughly those arising for problems with homogeneous densities.

Implementation of RMINRES solver in `PETSc` We implemented and integrated the recycling MINRES solver in `PETSc`. Through collaboration with the `PETSc` development team, this RMINRES code can be made available in `PETSc`. In fact, several research groups have requested our `PETSc` RMINRES solver and started to use it in their applications.

6.2 Future Work

More efficient AMR scheme As the density distribution inside the structures is largely uniform, coarser elements can be used to represent the interior of the structures. This requires an appropriate error estimator for the finite element solutions [55] and proper handling of the sensitivities, but it may further reduce the computational cost.

Recycling CG and BiCGStab methods The recycling MINRES method provides an insight to a potential recycling CG method. In principle, we can keep the short-term recurrence in CG with recycling in a similar fashion as in RMINRES. For sequences of nonsymmetric systems, a variation of BiCGStab can be made very efficient if we develop a way to recycle in a bi-orthogonal recurrence.

Recycling for adaptive mesh We presented analysis and preliminary ideas for Krylov subspace recycling on adaptive meshes in Section 4.6. We investigate different approaches for transforming the recycle space on the old mesh to the new mesh. We notice that although the transformed recycle vectors on the old mesh resemble the eigenvectors of the preconditioned system on the new mesh in terms of the angles between them, the transformed recycle vectors have large residuals as eigenvectors. Further investigation and thoughts are needed. For example, a few iterations of the Jacobi-Davidson method may make the transformed recycle vectors better approximations to the eigenvectors of the preconditioned system.

6.3 Closing Remarks

As topology optimization has become a popular research area, the study in this thesis provides positive contributions to make it a truly effective tool for designing large structures and complex materials. This work has brought new perspectives and ideas to the study of topology optimization, in particular the adaptive mesh refinement strategy, and the linear

solvers and preconditioners. On the other hand, some of the methods proposed and presented in this thesis are not limited to topology optimization problems. For example, the recycling method is generally useful for sequences of systems arising from optimization or nonlinear problems, and the multilevel SPAI preconditioner has been demonstrated to be effective for convection-diffusion types of equations. In fact, the work in this thesis has gained interest from researchers in physics and engineering fields.

References

[1] Altair Engineering Web page. http://www.altair.com/.

[2] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, **9**:17–29, 1951.

[3] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Value Problems: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, 2001. ISBN 0-89871-499-0.

[4] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.

[5] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Web page, 2001. http://www.mcs.anl.gov/petsc.

[6] M. P. Bendsøe. Optimal shape design as a material distribution problem. *Structural Optimization*, **1**(4):193–202, 1989.

[7] M. P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, **71**(2):197–224, 1988.

[8] M. P. Bendsøe and O. Sigmund. Material interpolation schemes in topology optimization. *Archives of Applied Mechanics*, **69**(9–10):635–654, 1999.

[9] M. P. Bendsøe and O. Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer-Verlag, Berlin, 2003. ISBN 3-540-42992-1.

[10] M. Benzi, J. K. Cullum, and M. T. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM Journal on Scientific Computing*, **22**(4):1318–1332, 2000.

[11] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, **17**(5):1135–1149, 1996.

[12] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, **19**(3):968–994, 1998.

[13] M. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, **82**(1):64–84, 1989.

[14] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, **53**:484–512, 1984.

[15] M. Bollhöfer and V. Mehrmann. A new approach to algebraic multilevel methods based on sparse approximate inverses. Technical Report Preprint SFB393/99-22, Department of Mathematics, Technische Universität Chemnitz, Germany, 1999.

[16] M. Bollhöfer and V. Mehrmann. Algebraic multilevel methods and sparse approximate inverses. *SIAM Journal on Matrix Analysis and Applications*, **24**(1):191–218, 2002.

[17] T. Borrvall and J. Petersson. Large-scale topology optimization in 3D using parallel computing. *Computer Methods in Applied Mechanics and Engineering*, **190**(46–47):6201–6229, 2001.

[18] R. Bracewell. *The Fourier Transform and Its Applications*. 3rd edition. McGraw-Hill, New York, 1999. ISBN 0-073-03938-1.

[19] R. Bridson and W.-P. Tang. Multiresolution approximate inverse preconditioners. *SIAM Journal on Scientific Computing*, **23**(2):463–479, 2002.

[20] T. F. Chan and K. Chen. Two-stage preconditioners using wavelet band splitting and sparse approximation. Technical Report CAM 00-26, Department of Mathematics, University of California, Los Angeles, 2000.

[21] T. F. Chan, W.-P. Tang, and W. L. Wan. Wavelet sparse approximate inverse preconditioners. *BIT Numerical Mathematics*, **37**(3):644–660, 1997.

[22] E. Chow. A priori sparsity patterns for parallel sparse approximate inverse preconditioners. *SIAM Journal on Scientific Computing*, **21**(5):1804–1822, 2000.

[23] E. Chow and Y. Saad. Approximate inverse techniques for block-partitioned matrices. *SIAM Journal on Scientific Computing*, **18**(6):1657–1675, 1997.

[24] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, **19**(3):995–1023, 1998.

[25] J. C. A. Costa Jr. and M. K. Alves. Layout optimization with h-adaptivity of structures. *International Journal for Numerical Methods in Engineering*, **58**(1):83–102, 2003.

[26] E. de Sturler. IBLU preconditioners for massively parallel computers. *Contemporary Mathematics*, **180**:395–400, 1995. D. E. Keyes and J. Xu (editors): *Domain Decomposition Methods in Scientific and Engineering Computing* (*Proceedings of the Seventh International Conference on Domain Decomposition*, October 27–30, 1993, The Pennsylvania State University).

[27] E. de Sturler. Incomplete block LU preconditioners on slightly overlapping subdomains for a massively parallel computer. *Applied Numerical Mathematics*, **19**(1–2):129–146, 1995.

[28] E. de Sturler. Nested Krylov methods based on GCR. *Journal of Computational and Applied Mathematics*, **67**(1):15–41, 1996.

[29] E. de Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM Journal on Numerical Analysis*, **36**(3):864–889, 1999.

[30] E. de Sturler, C. H. Le, S. Wang, and G. H. Paulino. Large scale topology optimization using preconditioned Krylov subspace recycling and continuous approximation of material distribution. In *Proceedings of the Multiscale and Functionally Graded Materials Conference (M&FGM2006)*. 2007.

[31] A. Gersborg-Hansen, O. Sigmund, and R. B. Haber. Topology optimization of channel flow problems. *Journal for Structural and Multidisciplinary Optimization*, **30**(3):181–192, 2005.

[32] G. H. Golub and C. F. Van Loan. *Matrix Computations*. 3rd edition. The Johns Hopkins University Press, Baltimore, 1996. ISBN 0-8018-5414-8.

[33] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, 1997. ISBN 0-89871-396-X.

[34] M. J. Grote and M. Hagemann. *SPAI - SParse Approximate Inverse Preconditioner*. Universität Basel, March 2006. http://www.computational.unibas.ch/software/spai.

[35] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, **18**(3):838–853, 1997.

[36] J. K. Guest, J. H. Prévost, and T. Belytschko. Achieving minimum length scale in topology optimization using nodal design variables and projection functions. *International Journal for Numerical Methods in Engineering*, **61**(2):238–254, 2004.

[37] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, **49**(6):409–436, 1952.

[38] R. D. Hornung, A. M. Wissink, and S. R. Kohn. Managing complex data and geometry in parallel structured amr applications. *Engineering with Computers*, **22**(3–4):181–195, 2006.

[39] M. E. Kilmer and E. de Sturler. Recycling subspace information for diffuse optical tomography. *SIAM Journal on Scientific Computing*, **27**(6):2140–2166, 2006.

[40] T. S. Kim, J. E. Kim, and Y. Y. Kim. Parallelized structural topology optimization for eigenvalue problems. *International Journal of Solids and Structures*, **41**(9–10):2623–2641, 2004.

[41] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. `libMesh`: a C++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, **22**(3):237–254, 2006.

[42] L. Y. Kolotilina and A. Y. Yeremin. Factorized sparse approximate inverse preconditionings. *SIAM Journal on Matrix Analysis and Applications*, **14**:45–58, 1993.

[43] L. Krog, A. Tucker, M. Kemp, and R. Boyd. Topology optimization of aircraft wing box ribs. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. 2004.

[44] U. D. Larsen, O. Sigmund, and S. Bouwstra. Design and fabrication of compliant micromechanisms and structures with negative Poisson's ratio. *IEEE Journal of Microelectromechanical Systems*, **6**(2):99–106, 1997.

[45] P. MacNeice, K. Olson, C. Mobarry, R. de Fainchtein, and C. Packer. PARAMESH: a parallel adaptive mesh refinement community toolkit. Technical Report NASA/CR–1999-209483, Goddard Space Flight Center, NASA, 1999.

[46] K. Matsui and K. Terada. Continuous approximation of material distribution for topology optimization. *International Journal for Numerical Methods in Engineering*, **59**(14):1925–1944, 2004.

[47] J. Meijerink and H. A. van der Vorst. An iterative solution method for linear equations systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, **31**:148–162, 1977.

[48] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, New York, 1995. ISBN 0-521-42922-6.

[49] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numerical Linear Algebra with Applications*, **2**(2):115–133, 1995.

[50] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, **12**:617–629, 1975.

[51] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM Journal on Scientific Computing*, **28**(5):1651–1674, 2006.

[52] G. H. Paulino. Editorial: Fracture of functionally graded materials. *Engineering Fracture Mechanics*, **69**(14–16):1519–1520, 2002.

[53] G. H. Paulino. Guest editorial: modeling of functionally graded materials. *International Journal of Computational Engineering Science*, **5**(4), 2004.

[54] G. H. Paulino and C. H. Le. A modified Q4/Q4 element for topology optimization. Submitted to *Journal for Structural and Multidisciplinary Optimization* in 2007.

[55] G. H. Paulino, I. F. M. Menezes, J. B. C. Neto, and L. F. Martha. A methodology for adaptive finite element analysis: towards an integrated computational environment. *Computational Mechanics*, **23**(5–6):362–388, 1999.

[56] G. H. Paulino and E. C. N. Silva. Design of functionally graded structures using topology optimization. *Materials Science Forum*, **492–493**:435–440, 2005.

[57] K. E. Petersen. Silicon as a mechanical material. *Proc. IEEE*, **70**(5):420–457, 1982.

[58] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd edition. Society for Industrial and Applied Mathematics, Philadelphia, 2003. ISBN 0-89871-534-2.

[59] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, **7**(3):856–869, 1986.

[60] O. Sigmund. *Design of Material Structures Using Topology Optimization*. Ph.D. thesis, Department of Solid Mechanics, Technical University of Denmark, 1994.

[61] O. Sigmund. On the design of compliant mechanisms using topology optimization. *Mechanics of Structures and Machines*, **25**(4):495–526, 1997.

[62] O. Sigmund. Topology optimization: a tool for the tailoring of structures and materials. *A special issue of the Philosophical Transactions of the Royal Society: Science into the next Millennium (Issue III, Mathematics, Physics and Engineering)*, **358**(1765):211–227, 2000.

[63] O. Sigmund and J. S. Jensen. Systematic design of phononic band gap materials and structures. *Philosophical Transactions of the Royal Society London, Series A (Mathematical, Physical and Engineering Sciences)*, **361**:1001–1019, 2003.

[64] O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Journal for Structural and Multidisciplinary Optimization*, **16**(1):68–75, 1998.

[65] O. Sigmund and S. Torquato. Design of materials with extreme thermal expansion using a three-phase topology optimization method. *Journal of the Mechanics and Physics of Solids*, **45**(6):1037–1067, 1997.

[66] G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, **17**(2):401–425, 1996.

[67] R. Stainko. An adaptive multilevel approach to the minimal compliance problem in topology optimization. *Communications in Numerical Methods in Engineering*, **22**(2):109–118, 2006.

[68] G. W. Stewart. *Matrix Algorithms Volume II: Eigensystems*. Society for Industrial and Applied Mathematics, Philadelphia, 2001. ISBN 0-89871-503-2.

[69] K. Svanberg. The method of moving asymptotes: a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, **24**(2):359–373, 1987.

[70] W.-P. Tang and W. L. Wan. Sparse approximate inverse smoother for multigrid. *SIAM Journal on Matrix Analysis and Applications*, **21**(4):1236–1252, 2000.

[71] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, **13**(2):631–644, 1992.

[72] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, 2003. ISBN 0-521-81828-1.

[73] K. Vemaganti and W. E. Lawrence. Parallel methods for optimality criteria-based topology optimization. *Computer Methods in Applied Mechanics and Engineering*, **194**(34–35):3637–3667, 2005.

[74] C. Vuik and J. Frank. Coarse grid acceleration of a parallel block preconditioner. *Future Generation Computer Systems*, **17**(8):933–940, 2001.

[75] K. Wang, J. Zhang, and C. Shen. Parallel multilevel sparse approximate inverse preconditioners in large sparse matrix computations. In *Proceedings of the ACM/IEEE SC2003 Conference*. 2003.

[76] S. Wang and E. de Sturler. Multilevel sparse approximate inverse preconditioners with dynamic meshes. In preparation.

[77] S. Wang, E. de Sturler, and G. H. Paulino. Large-scale topology optimization using preconditioned Krylov subspace methods with recycling. *International Journal for Numerical Methods in Engineering*, **69**(12):2441–2468, 2007.

Author's Biography

Shun Wang was born in Wuhan, China, on November 2, 1979. He earned his bachelor's degree in computer science from the Tsinghua University in 2002. He began his Ph.D. studies in the Department of Computer Science at the University of Illinois at Urbana-Champaign in the fall of 2002.