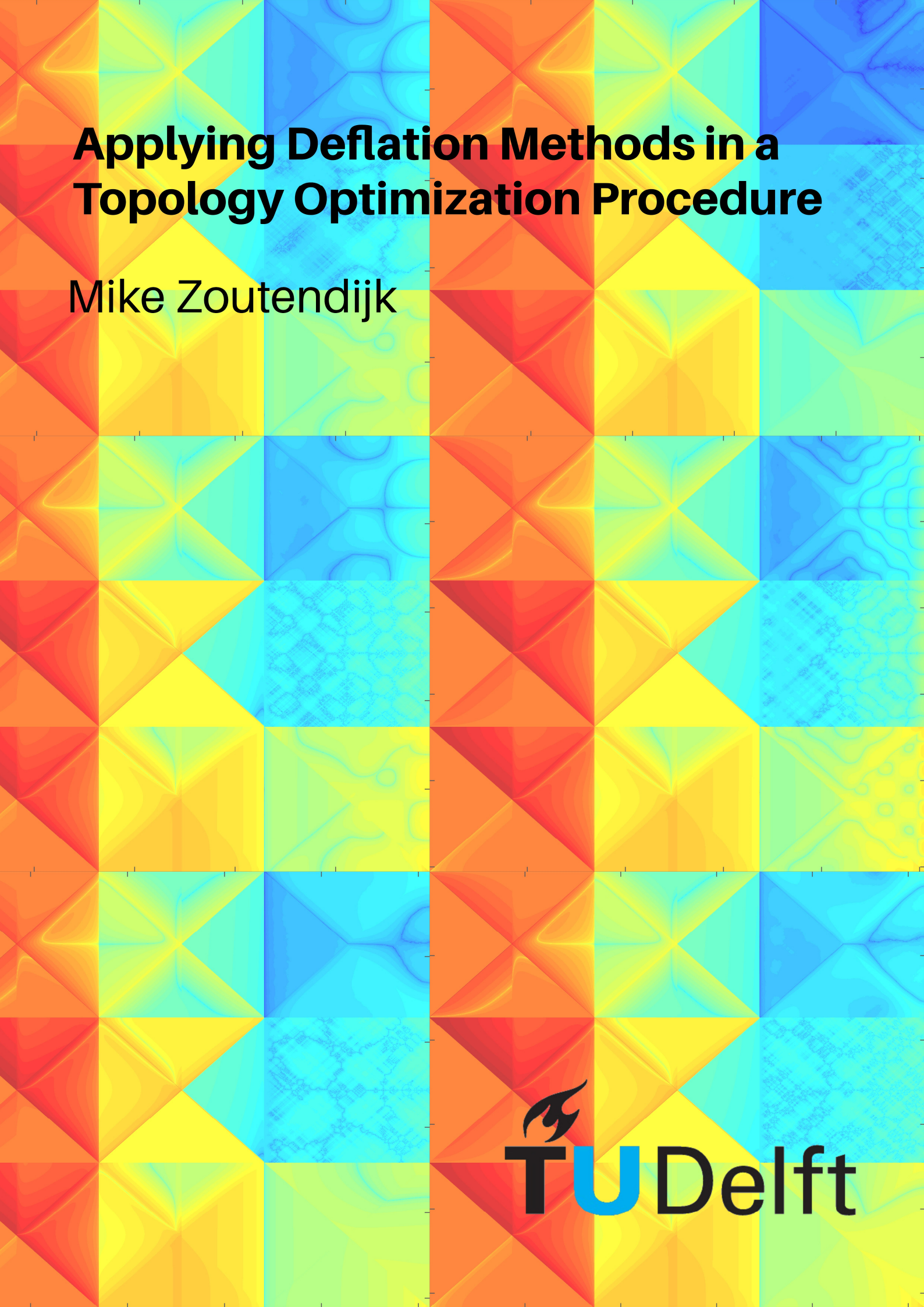


Applying Deflation Methods in a Topology Optimization Procedure

Mike Zoutendijk

TU Delft

Applying Deflation Methods in a Topology Optimization Procedure

by

Mike Zoutendijk

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Tuesday November 19, 2019 at 10:00 AM

Student number:	4355393	
Project duration:	January 7, 2019 – November 19, 2019	
Thesis committee:	Dr. ir. M. B. van Gijzen,	TU Delft, supervisor
	Dr. ir. R. Astudillo Rengifo,	TU Delft, supervisor
	Dr. ir. M. Langelaar,	TU Delft
	Dr. ir. W. T. van Horssen,	TU Delft

An electronic version of this thesis is available at https://repository.tudelft.nl/.

Abstract

Structure Optimization has been an important subject with many applications for centuries. In the last sixty years, numerical optimization has facilitated large advancements in this field. One of the areas in Structure Optimization is Topology Optimization, which is used for Additive Manufacturing purposes. In this thesis we explore Static and Dynamic Topology Optimization. In the optimization problems matrix equations of the type $\boldsymbol{Ax} = \boldsymbol{b}$, with $\boldsymbol{A}$ sparse and badly conditioned, are accelerated using deflation techniques in addition to preconditioning. We have applied several iterative methods, preconditioners, and deflation types to the topology optimization problems.

The static problem concerns compliance optimization of a two-dimensional MBB-beam. The deflation type that reduced the number of iterations the most without introducing large overhead costs was rigid body modes deflation, divided over element squares. It was found that dividing the rigid body modes vectors over density based regions did not reduce the number of iterations.

The dynamic problem concerns eigenvalue optimization of a three-dimensional moving wafer stage that is used for laser-printing computer chips. The optimization formulation contains a shifted eigenvalue problem that is solved using model order reduction. In the computations of bases for the reduction matrix equations appear, to which deflation techniques were applied.

All deflation types reduced the number of iterations and needed time to solve the matrix equations. The best deflation type was using rigid body modes (RBM) divided over element cubes combined with eigenvectors from the previous iteration. The next best was the same combination without the division over cubes. Using eigenvectors or RBM separately were the least effective deflation types. There is an optimal amount of element cubes to use when dividing the RBM. The tests on a few grid sizes suggested a quadrupling of the amount when the grid size doubles, but more research is needed to really identify a relation between the grid size and optimal amount.

When increasing the grid size to a level where parallel computing on a cluster was required, the deflation type using element cubes could not be used due to complications with parallel implementation. Using the deflation type RBM and eigenvectors, reductions by a factor of 1.60 and 1.75 in the total needed time for 150 optimization iterations were achieved for grid sizes $120 \times 120 \times 20$ and $180 \times 180 \times 30$, respectively. In the first case the objective function of the optimized design converged further in the same amount of iterations when using deflation. Future research could include using the promising deflation type RBM in cubes + eigenvectors in parallel computing to obtain an even larger time reduction in the optimization of the wafer stage with large grid sizes.

Acknowledgements

I would like to thank a few people, without whom it would not have been possible to complete this work.

Firstly, I would like to thank my supervisors Martin and Reinaldo, for their excellent guidance during the past ten months. During the countless meetings I got expert feedback and new ideas, and learned a lot about deflation and topology optimization. I want to thank Reinaldo for taking the time for the programming sessions and teaching me the wonders of PETSc, and for implementing the IDR method in it. I'd like to thank the rest of the participants in the Topology Optimization/IMSYS project, Matthijs, Arnoud, Fred and Ruben, for their input in the meetings that provided me with more insight in the other aspects of the TO project.

Next, I would like to thank my study friends Carlos, Marco, Massimo, Pieter, Rick and Roel, for making the time at the university a pleasant one, full of pooling, playing cards and talking about courses and each others projects. I'd like to thank Pieter, Marco and Jenny for proofreading my thesis and their tips.

Lastly, I would like to thank my parents and brother for their unconditional love and support over the years, and my beloved Jenny for being there for me in the past year and listening to my thesis struggles.

Mike Zoutendijk
Rijswijk, November 2019

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Structure design is a task that has been performed by humans for centuries. There are many constraints and demands for a structure that can be formulated, making it a complex task. In 1960, Schmit [21] first introduced numerical optimization by combining finite element analysis and nonlinear optimization. Since then, a lot more is possible in structure design. More recently, the use of material printing techniques has allowed for additive manufacturing, i.e. starting from an empty design space, and adding material only where it is necessary. In this field, Topology Optimization is very useful.

Topology Optimization

Topology Optimization is an optimization method that optimizes the design of a structure, subject to certain constraints. There are several approaches, but the one that is used in our application is called SIMP (Solid Isotropic Material with Penalization). It uses the finite elements method on a grid of elements. Every element is assigned a density, and these densities define the design. We will first consider Static Topology Optimization, following Sigmund [24] and Andreassen et al. [2]. The compliance of a static structure is optimized. After this we will investigate Dynamic Topology Optimization, where a moving structure is optimized. Our structure of interest is a wafer stage, on which computer chips are printed using lasers. This is done very precisely, calling for exact controllability of the wafer stage. Optimizing the structure of this stage with a fine enough spacial discretization grid in manageable time is the goal of the Topology Optimization project. In this project, TU Delft researchers in the areas of Precision and Microsystems Engineering, and Numerical Analysis work on this problem, financially supported by companies active in the field of Additive Manufacturing, and Photolithography.

The structure is optimized by maximizing the smallest m eigenvalues of the system. The optimization formulation contains a shifted eigenvalue problem. This eigenvalue problem is solved using model order reduction. Model order reduction is a technique that reduces the needed computation time by approximating the eigenvalue problem with one that has far fewer degrees of freedom than the original, in such a way that the transfer function of the problem is approximated as accurately as possible [8], [20], [32], [6]. This can be done by constructing bases for certain Krylov subspaces, and for this several matrix equations need to be solved. These computations are taking a large percentage of the total needed optimization time needed. In an effort to reduce this, we will apply deflation techniques to these matrix equations.

Deflation Methods

We aim to accelerate the solving of matrix equations of the type

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

with $\boldsymbol{A}$ a sparse matrix with a large condition number, $\boldsymbol{x}$ the solution vector, and $\boldsymbol{b}$ the right-hand-side vector. They are often solved using preconditioned iterative methods and we want to accelerate this with deflation.

Deflation, first introduced by Nicolaides [16], is a technique that can be applied to iterative methods and improves the effective condition number of the system matrix $\boldsymbol{A}$ by essentially removing the lowest eigenvalues. The solution is split in a part that is immediately solvable and a part that is solved using the iterative method but has better effective condition number, and thus can be solved faster. Deflation requires a set of deflation vectors, and there is currently no optimal method to choose these deflation

vectors. For every application it has to be investigated whether certain deflation vectors reduce the number of iterations needed.

Deflation has been the subject of a lot of research. For example, Vuik et al. [31] have applied deflation on oil pressure calculations in reservoirs, which used the Conjugate Gradient (CG) method, an Incomplete Cholesky (IC) preconditioner, and as deflation vectors the eigenvectors corresponding to the smallest eigenvalues of the problem, and variants of that. Frank and Vuik [7] used subdomain deflation with CG and Relaxed IC on several PDE's and provided guidelines for selecting subdomains. Tang et al. [28] compared projection methods based on deflation, multigrid methods and domain decomposition. Tang [27] applied several two-levels methods, including deflation methods, to the Poisson equation describing bubbly flow. Jönsthövel et al. [12] used rigid body modes deflation with CG on a stiffness matrix equation describing the interaction between the materials in asphalt. This is the first time rigid body modes were used as deflation vectors to our knowledge. Sheikh [22] applied deflation to the Helmholtz equation.

In this thesis, we will try several deflation types on the Topology Optimization problems. We will also use several iterative methods. Firstly, the Conjugate Gradient method [9], which is suited for symmetric positive definite (SPD) matrices. The system matrix will not always be SPD, so we will also introduce methods suitable for non-SPD matrices, namely MINRES [17], GMRES [19], BiCGSTAB [30] and IDR [26]. We will compare three different preconditioners: Jacobi, GAMG (Geometric Algebraic Multigrid Preconditioner) and GASM (Generalized Additive Schwarz Method).

Research question

Now that we have been introduced to Topology Optimization and deflation methods, it is time to formulate the research question that this thesis centres on:

How do the number of Krylov iterations and the needed time improve with respect to the preconditioned method if we apply deflation methods to the methods solving the matrix equations used in the 3D dynamic topology optimization problem that describes the optimization of the wafer stage?

Thesis outline

We will start in Chapter 1 with an introduction to the Conjugate Gradient method and with illustrating how an iterative method is improved by using preconditioning and, in Chapter 2, deflation. Several deflation methods will be presented there, and the chapter concludes with a one-dimensional example of an application of deflation. In Chapter 3 we will introduce an example of a two-dimensional Static Topology Optimization problem, and apply several deflation methods on it, using MATLAB. Several of these are then selected to be used in the Dynamic Topology Optimization problem, that will be presented in Chapter 4. The problem that optimizes the wafer stage is presented there, and the theory describing the use of model order reduction is explained. In Chapter 5 the deflation methods are applied to the problem. This is implemented in the programming language C using the library PETSc (Portable, Extensible Toolkit for Scientific Computation). The implementation is done using increasing grid size and an increasing number of matrix equations, and results are presented. Finally, in Chapter 6 the conclusions following all results are given, accompanied by ideas for future research. At the end, two appendices can be found, containing the notation and symbols used in the thesis, some definitions of notions from linear algebra and iterative methods, and residual norm graphs belonging to the results in Chapter 5.

This master thesis is written in partial fulfilment of the requirements for the master Applied Mathematics of the TU Delft.

Chapter 1

The Conjugate Gradient Method

In this thesis, we aim to solve large sparse linear systems with large differences in coefficients in the governing equations. In general, sparse matrices arise in discretized problems that are governed by differential equations. The problems addressed in this thesis fall into this category. When the system is symmetric positive definite (SPD) the Conjugate Gradient method (CG) is very suitable. Later we will also consider problems that are not SPD, and introduce other methods that are suitable for them. In this chapter, the problems will be SPD.

A large difference in coefficients often implies a large difference between the smallest and largest eigenvalues of the matrix. The smallest eigenvalues in a problem correspond to the slowest converging components of the solution, and the convergence rate of CG depends on the ratio of the largest and smallest eigenvalues of the matrix [31].

In order to deal with the large differences in coefficients, we will use deflation on the CG method (in the SPD case), and therefore we will first introduce the CG method and its background before using it to explain deflation. We will use concepts from e.g. linear algebra and iterative methods. For a list of notation, definitions and symbols, see Appendix A.

1.1 Projection Methods

We consider the linear system of equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{1.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is an SPD matrix, $\boldsymbol{b} \in \mathbb{R}^n$ is the right-hand side vector, and both are known. The solution we want to find is $\boldsymbol{x} \in \mathbb{R}^n$. For SPD matrices convergence in the $\boldsymbol{A}$-norm is guaranteed.

This is the type of system that we aim to solve in Chapter 2 and 3. In Chapter 4 and 5 we will also solve systems that are not SPD.

Systems like 1.1 can be solved using direct methods, such as the Gaussian elimination method, or Cholesky decomposition when $\boldsymbol{A}$ is SPD. However, when the direct computation of a solution takes too much time, one can resort to iterative methods, which start with an inital guess to the solution, and adapt it in every iteration until a certain convergence is reached.

1.1.1 General projection methods

Many of the well-known iterative methods in numerical analysis are based on the framework of projection methods. In general, a projection method is a way to obtain an approximation to the solution of a linear system from a subspace of $\mathbb{R}^n$ [18]. This subspace is denoted by $\mathcal{K}$. If it has dimension m, we need m orthogonal constraints to obtain the approximation. In fact, the residual vector $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ has to be orthogonal to m independent vectors. These vectors span an m-dimensional subspace $\mathcal{L}$.

There are two classes of projection methods: orthogonal and oblique. In oblique projection methods $\mathcal{L}$ and $\mathcal{K}$ are different, but in orthogonal projection methods $\mathcal{L} = \mathcal{K}$. CG is an orthogonal projection method. We will now proceed to only use $\mathcal{K}$, having dimension m.

An orthogonal projection process can thus be stated as:

$$\text{Find } \tilde{\boldsymbol{x}} \in \boldsymbol{x}_0 + \mathcal{K}, \qquad \text{such that } \boldsymbol{b} - \boldsymbol{A}\tilde{\boldsymbol{x}} \perp \mathcal{K}. \tag{1.2}$$

Here $\tilde{\boldsymbol{x}}$ is the approximation to the solution $\boldsymbol{x}$ and $\boldsymbol{x}_0$ is the initial guess.

If one writes the approximation as $\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 + \boldsymbol{p}$, with $\boldsymbol{p}$ the difference vector between the approximation and the initial guess, and define $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, the residual, we can derive:

$$\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{p} \perp \mathcal{K}, \tag{1.3}$$

meaning that Eq. 1.2 can be rearranged to:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}_0 + \boldsymbol{p}, \quad \boldsymbol{p} \in \mathcal{K} \tag{1.4}$$

$$(\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{p}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in \mathcal{K}, \tag{1.5}$$

where $(\cdot, \cdot)$ denotes an inproduct. This is a representation of one general projection step. Every step the subspace $\mathcal{K}$ is updated. The vector $\boldsymbol{p}$ is called a search direction.

A general optimality result for orthogonal projection methods is:

Theorem 1.1. *Assume that A is SPD. Then a vector $\tilde{\boldsymbol{x}}$ is the result of an orthogonal projection method onto $\mathcal{K}$ with the starting vector $\boldsymbol{x}_0$ if an only if it minimizes the $\boldsymbol{A}$-norm of the error over $\boldsymbol{x}_0 + \mathcal{K}$, i.e., if and only if:*

$$E(\tilde{\boldsymbol{x}}) = \min_{\boldsymbol{x} \in \boldsymbol{x}_0 + \mathcal{K}} E(\boldsymbol{x}), \tag{1.6}$$

where

$$E(\boldsymbol{x}) \equiv \|\boldsymbol{x}_{ex} - \boldsymbol{x}\|_{\boldsymbol{A}}. \tag{1.7}$$

[Adapted after Saad [18]].

In this theorem $\boldsymbol{x}_{\text{ex}}$ is the exact solution to the linear system. The theorem is stated and proven by Saad [18]. It states that if $\boldsymbol{x}_{\text{ex}} - \tilde{\boldsymbol{x}}$ is $\boldsymbol{A}$-orthogonal to $\mathcal{K}$, $\tilde{\boldsymbol{x}}$ minimizes E and is therefore optimal.

1.1.2 Method of steepest descent

Consider a general one-dimensional orthogonal projection process:

$$\mathcal{K} = \text{span}\{\boldsymbol{v}\}, \tag{1.8}$$

where $\boldsymbol{v} \in \mathcal{K}$ is a vector. The solution is updated as $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha\boldsymbol{v}$, and we need to find α. Condition 1.5 yields:

$$\alpha = \frac{(\boldsymbol{r}, \boldsymbol{v})}{(\boldsymbol{A}\boldsymbol{p}, \boldsymbol{v})}. \tag{1.9}$$

A specific example of a one-dimensional orthogonal projection process is the method of steepest descent. It is used for SPD matrices. Take $\boldsymbol{p} = \boldsymbol{r}$ and $\boldsymbol{v} = \boldsymbol{r}$. This yields the following update lines for every iteration:

$$\begin{aligned} \boldsymbol{r} &\leftarrow \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}, \\ \alpha &\leftarrow \frac{(\boldsymbol{r}, \boldsymbol{r})}{(\boldsymbol{A}\boldsymbol{r}, \boldsymbol{r})}, \\ \boldsymbol{x} &\leftarrow \boldsymbol{x} + \alpha\boldsymbol{r}. \end{aligned}$$

To save computation time, one can rewrite this to only contain one matrix-vector product: instead of $\boldsymbol{p} = \boldsymbol{r}$, we take $\boldsymbol{p} = \boldsymbol{A}\boldsymbol{r}$, yielding Algorithm 1:

Algorithm 1 Method of Steepest Descent

Choose $\boldsymbol{x}_0$ and ε
Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ and $\boldsymbol{p}_0 = \boldsymbol{A}\boldsymbol{r}_0$
while $\|\boldsymbol{r}_j\|/\|\boldsymbol{b}\|/n > \varepsilon$ **do**
    $\alpha_j := \frac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(\boldsymbol{p}_j, \boldsymbol{r}_j)}$
    $\boldsymbol{x}_{j+1} := \boldsymbol{x}_j + \alpha_j \boldsymbol{r}_j$
    $\boldsymbol{r}_{j+1} := \boldsymbol{r}_j - \alpha_j \boldsymbol{p}_j$
    $\boldsymbol{p}_{j+1} := \boldsymbol{A}\boldsymbol{r}_{j+1}$
    $j := j + 1$
end while

In every iteration step

$$f(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}_{\text{ex}}\|_{\boldsymbol{A}}^2 = (\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_{\text{ex}}), (\boldsymbol{x} - \boldsymbol{x}_{\text{ex}})), \tag{1.10}$$

is minimized over all vectors of the form $\boldsymbol{x} + \alpha \boldsymbol{x}_\delta$, where $\boldsymbol{x}_\delta = -\nabla f$. The latter is the direction that locally yields the fastest rate of decrease in f. Hence the name steepest descent method. Saad [18] states this algorithm and proves that convergence is guaranteed when $\boldsymbol{A}$ is SPD, and Shewchuk [23] states the following for the convergence at iteration $j + 1$:

$$\|\boldsymbol{x} - \boldsymbol{x}_{j+1}\|_A \leq \|\boldsymbol{x} - \boldsymbol{x}_0\|_A \left(\frac{\kappa(\boldsymbol{A}) - 1}{\kappa(\boldsymbol{A}) + 1} \right)^{j+1}, \tag{1.11}$$

where $\kappa(\boldsymbol{A})$ is the condition number of matrix $\boldsymbol{A}$.

In Fig. 1.1 an example is shown of convergence using the method of steepest descent.

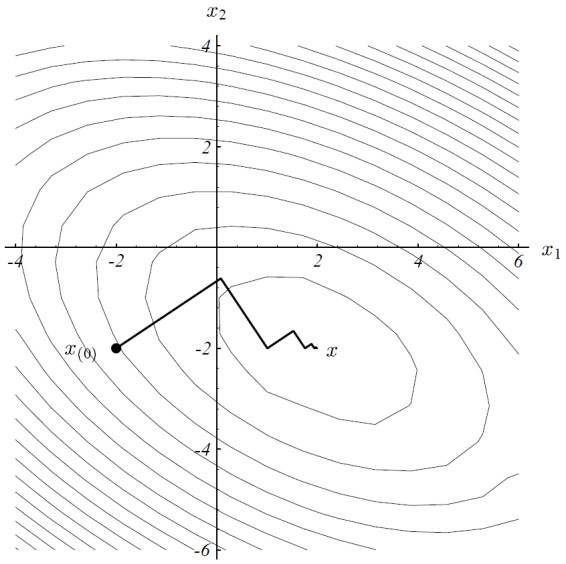


Figure 1.1: A 2D example of converging with the method of steepest descent. The starting point is $[-2, -2]$. The closed loops are contour lines related to Eq. 1.10. x is the minimum one would like to find. Taken from Shewchuk [23].

1.1.3 Method of conjugate directions

In every new iteration of the method of steepest descent the search direction is orthogonal to the previous search direction. This often causes zigzag patterns, or other patterns in which search directions are repeated [23], as can be seen in Fig. 1.1.

It would be best if in addition to the constraint that a direction is orthogonal to the last one, all search directions are orthogonal to each other: the set $\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_{m-1}$ is orthogonal. Then, every search direction will be used once, there will be m orthogonal search directions, and m steps.

However, the conditions for the α_j's would then need the errors $\boldsymbol{e}_j = \boldsymbol{x}_j - \boldsymbol{x}_{\text{ex}}$, which are of course unknown. A solution is to make the search directions conjugate instead of orthogonal: instead of taking the set of search directions to be orthogonal, we now require that $\boldsymbol{e}_{j+1}$ is $\boldsymbol{A}$-orthogonal to $\boldsymbol{p}_j$. This is in fact the same requirement as in the method of Steepest Descent: the optimal point along search direction $\boldsymbol{p}_j$ is found. This yields a condition for α_j that does not contain unknowns, namely:

$$\alpha_j = \frac{(\boldsymbol{p}_j, \boldsymbol{r}_j)}{(\boldsymbol{A}\boldsymbol{p}_j, \boldsymbol{p}_j)}. \tag{1.12}$$

Note that if $\boldsymbol{p}_j = \boldsymbol{r}_j$, we obtain the method of Steepest Descent.

The set of $\boldsymbol{A}$-orthogonal search directions can be generated from any set of m linearly independent vectors using the conjugate Gram-Schmidt process [23]. However, this process requires the storage of all old search vectors to calculate a new one. Therefore, this so-called method of Conjugate Directions is not practical as it is.

1.1.4 Krylov subspace methods

Before we apply the knowledge of the previous subsections to construct the Conjugate Gradient method, it is useful to introduce the Krylov subspace.

We have seen in the previous subsections that a general projection method solves system 1.1 with a solution in a subspace $\boldsymbol{x}_0 + \mathcal{K}$ of dimension m using condition 1.5. A Krylov subspace method is a projection method where $\mathcal{K}$ is a Krylov subspace:

Definition 1.2. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be an SPD matrix, and let $\boldsymbol{x}_0$ be the initial guess. The Krylov subspace of degree m for $\boldsymbol{A}$ and $\boldsymbol{x}_0$ is defined as:*

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) = \mathit{span}\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, ..., \boldsymbol{A}^{m-1}\boldsymbol{r}_0\}, \tag{1.13}$$

where $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$.

$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ is often denoted by $\mathcal{K}_m$. The solution to system 1.1 is in the Krylov subspace, and finding the solution can be described as approximating the inverse of $\boldsymbol{A}$ by a polynomial in $\boldsymbol{A}$, in the case that $\boldsymbol{x}_0 = \boldsymbol{0}$:

$$\boldsymbol{A}^{-1}\boldsymbol{b} \approx q_{m-1}(\boldsymbol{A})\boldsymbol{r}_0 = q_{m-1}(\boldsymbol{A})\boldsymbol{b}, \tag{1.14}$$

where q_{m-1} is a polynomial in $\boldsymbol{A}$ of degree $m-1$. So we see that the Krylov subspace $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$ is the subspace of all vectors $\boldsymbol{x} \in \mathbb{R}^n$ which can be written as $\boldsymbol{x} = q_{m-1}(\boldsymbol{A})\boldsymbol{r}_0$.

1.2 Conjugate Gradient Method

We now arrive at the Conjugate Gradient method. It is an iterative method for solving sparse linear SPD systems. It is suitable for sparse systems because the matrix $\boldsymbol{A}$ is not altered in such a way that the sparsity is lost, i.e. during the algorithm, this advantage, fewer necessary operations, is kept.

We have seen that the method of Conjugate Directions uses a set of m $\boldsymbol{A}$-orthogonal search directions $\boldsymbol{d}_j$, constructed from another set of m linearly independent vectors, to approach the solution $\boldsymbol{x}$. The Conjugate Gradient method is in fact the method of Conjugate Directions, where the set of independent vectors are the residuals. In other words: the search directions are constructed by conjugation of the residuals. The advantages of using the residuals are that they can be proven to be orthogonal to previous search directions and previous residuals and that the subspace of the search directions ends up to be the Krylov subspace [23].

We know from Sec. 1.1.3 that getting the search directions for the Conjugate Directions method takes many operations. Shewchuk [23] shows that when using the residuals, one does not need to store all search vectors to construct the new vector, and the number of operations is drastically reduced.

Now everything can be pieced together. In every iteration, the $(j+1)$-th iterate is updated with search direction $\boldsymbol{p}_j$ as:

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j, \tag{1.15}$$

and the residual, initially defined as $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, is updated as:

$$\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j \boldsymbol{A}\boldsymbol{p}_j. \tag{1.16}$$

The new iterand $\boldsymbol{x}_{j+1}$ is found in search direction $\boldsymbol{p}_j$ at a distance of coefficient α_j, which is found from the previous residual and search direction. The new search direction $\boldsymbol{p}_{j+1}$ is $\boldsymbol{A}$-orthogonal to the previous and found from the updated residual and the previous search direction with a factor β_j.

The CG method is described by the following algorithm, first given by Hestenes and Stiefel [9]:

Algorithm 2 Conjugate Gradient

1: Choose $\boldsymbol{x}_0$ and ε
2: Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
3: Set $\boldsymbol{p}_0 \coloneqq \boldsymbol{r}_0$ and $j \coloneqq 0$
4: **while** $\|\boldsymbol{r}_j\|/\|\boldsymbol{b}\|/n > \varepsilon$ **do**
5: $\quad \boldsymbol{w}_j \coloneqq \boldsymbol{A}\boldsymbol{p}_j$
6: $\quad \alpha_j \coloneqq \frac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(\boldsymbol{w}_j, \boldsymbol{p}_j)}$
7: $\quad \boldsymbol{x}_{j+1} \coloneqq \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j$
8: $\quad \boldsymbol{r}_{j+1} \coloneqq \boldsymbol{r}_j - \alpha_j \boldsymbol{w}_j$
9: $\quad \beta_j \coloneqq \frac{(\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1})}{(\boldsymbol{r}_j, \boldsymbol{r}_j)}$
10: $\quad \boldsymbol{p}_{j+1} \coloneqq \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{p}_j$
11: $\quad j \coloneqq j + 1$
12: **end while**
13: $\boldsymbol{x}_s = \boldsymbol{x}_{j+1}$

In Alg. 2 $\boldsymbol{x}_0$ is the initial guess for the solution, ε is the termination criterion, and $\boldsymbol{x}_s$ is the numerically obtained solution from this algorithm.

As seen earlier, every residual is orthogonal to previous residuals and every search direction is conjugate to previous search directions. This makes sure that the residuals form an orthogonal basis for the m-dimensional Krylov space $\mathcal{K}_m$, which in turn ensures that the CG method minimizes the error in $\boldsymbol{x}_s$ over the Krylov subspace in the $\boldsymbol{A}$-semi-norm. Because $\boldsymbol{A}$ is SPD this is a regular $\boldsymbol{A}$-norm. CG will thus always convergence in exact arithmetic within n iterations, because then there is one vector in the Krylov subspace for every element in the solution. The convergence of CG can be described by [23],[27]:

$$\|\boldsymbol{x} - \boldsymbol{x}_{j+1}\|_A \leq 2\|\boldsymbol{x} - \boldsymbol{x}_0\|_A \left(\frac{\sqrt{\kappa(\boldsymbol{A})} - 1}{\sqrt{\kappa(\boldsymbol{A})} + 1} \right)^{j+1}, \tag{1.17}$$

We see that the larger the condition number, the slower the convergence, and the closer the condition number is to 1, the faster the convergence. In discretized problems with large differences in coefficients, we see that the condition number can become very large. We will see later, that nonexact arithmetic combined with large changes in coefficients can cause CG to take many more than n iterations.

In Fig. 1.2 we return to the example shown in Fig. 1.1, but now solved using CG. Note that convergence now takes $n = 2$ iterations, instead of many more, as in the zig-zag pattern seen with the method of steepest descent.

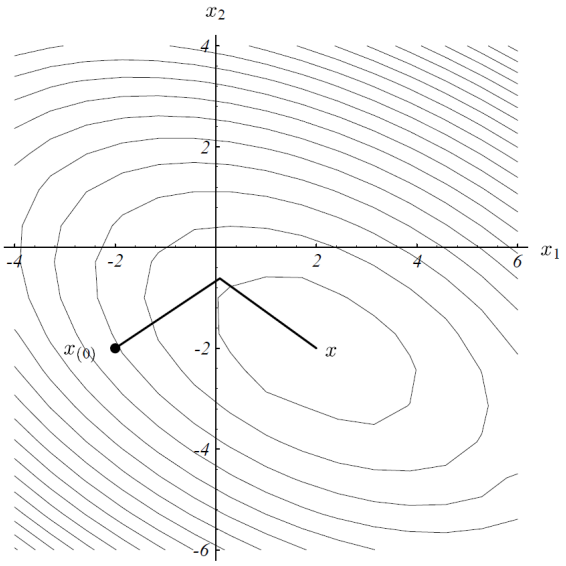


Figure 1.2: A 2D example of converging with the conjugate gradient method. The starting point is $[-2, -2]$. The closed loops are contour lines related to Eq. 1.10. x is the minimum one would like to find. Taken from Shewchuk [23].

For a more detailed description of the CG method, see for example Hestenes and Stiefel [9], Saad [18], Shewchuk [23], or Tang [27].

1.3 Preconditioned Conjugate Gradient Method

In the previous section we have seen, using the convergence bound Eq. 1.17, that a large condition number can slow down the CG method. One way to improve the condition number is to use a preconditioner. A preconditioner is the inverse of a matrix $\boldsymbol{M}$, where $\boldsymbol{M}$ is SPD. Multiplying $\boldsymbol{A}$ with a preconditioner in the symmetric way means the result is always SPD [27]. The advantage is that it has better condition number and eigenvalue distribution: the eigenvalue are clustered, reducing the condition number. A requirement on the preconditioner is that the inverse exists and that construction of and operations with $\boldsymbol{M}^{-1}$ take relatively few operations. This is because they will be performed in every iteration of the algorithm. $\boldsymbol{M}$ itself does not need to be calculated explicitly.

We define:

$$\widetilde{\boldsymbol{A}} \equiv \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{M}^{-\frac{1}{2}}, \quad \tilde{\boldsymbol{x}} \equiv \boldsymbol{M}^{\frac{1}{2}}\boldsymbol{x}, \quad \tilde{\boldsymbol{b}} \equiv \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{b}, \tag{1.18}$$

so that

$$\begin{aligned} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \iff & \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{b} \\ \iff & \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{M}^{\frac{1}{2}}\boldsymbol{x} = \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{b} \\ \iff & \widetilde{\boldsymbol{A}}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \end{aligned} \tag{1.19}$$

and we see that the transformed system equations hold when the original system equations hold. We can apply this to Alg. 2, and then substitute the preconditioned matrix and vectors from 1.18 back into the original notation. This yields Alg. 3 [27]:

Algorithm 3 Preconditioned Conjugate Gradient

Choose $\boldsymbol{x}_0$ and ε
Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ and solve $\boldsymbol{M}\boldsymbol{y}_0 = \boldsymbol{r}_0$
Set $\boldsymbol{p}_0 \coloneqq \boldsymbol{y}_0$ and $j \coloneqq 0$
while $\|\boldsymbol{r}_j\|/\|\boldsymbol{b}\|/n > \varepsilon$ **do**
    $\boldsymbol{w}_j \coloneqq \boldsymbol{A}\boldsymbol{p}_j$
    $\alpha_j \coloneqq \frac{(\boldsymbol{r}_j, \boldsymbol{y}_j)}{(\boldsymbol{p}_j, \boldsymbol{w}_j)}$
    $\boldsymbol{x}_{j+1} \coloneqq \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j$
    $\boldsymbol{r}_{j+1} \coloneqq \boldsymbol{r}_j - \alpha_j \boldsymbol{w}_j$
    Solve $\boldsymbol{M}\boldsymbol{y}_{j+1} = \boldsymbol{r}_{j+1}$
    $\beta_j \coloneqq \frac{(\boldsymbol{r}_{j+1}, \boldsymbol{y}_{j+1})}{(\boldsymbol{r}_j, \boldsymbol{y}_j)}$
    $\boldsymbol{p}_{j+1} \coloneqq \boldsymbol{y}_{j+1} + \beta_j \boldsymbol{p}_j$
    $j \coloneqq j + 1$
end while
$\boldsymbol{x}_s = \boldsymbol{x}_{j+1}$

System 1.19 is often denoted as:

$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}. \tag{1.20}$$

This notation corresponds to a slightly different way of preconditioning: left-preconditioning instead of symmetric preconditioning. In the case of left-preconditioning the $\boldsymbol{M}$-inproduct has to be used in the algorithm.

Note that in Alg. 3 only $\boldsymbol{M}^{-1}$ itself needs to be computed, and $\boldsymbol{M}^{\frac{1}{2}}$ does not. The convergence result for Preconditioned Conjugate Gradient (PCG) follows logically from Eq. 1.17 [27]:

$$\|\boldsymbol{x} - \boldsymbol{x}_{j+1}\|_A \leq 2\|\boldsymbol{x} - \boldsymbol{x}_0\|_A \left(\frac{\sqrt{\kappa(\boldsymbol{M}^{-1}\boldsymbol{A})} - 1}{\sqrt{\kappa(\boldsymbol{M}^{-1}\boldsymbol{A})} + 1} \right)^{j+1}. \tag{1.21}$$

The question remains which $\boldsymbol{M}$ is the best choice. A large amount of research into preconditioners has been done and is still being done. In the coming chapters we will use several different preconditioners. For the one- and two-dimensional problems that will be investigated first, we will use the most simple option: the diagonal Jacobi preconditioner. This preconditioner is based on taking the diagonal elements of the system matrix:

$$\boldsymbol{M} = \text{diag}(\boldsymbol{A}).$$

In this chapter, we have explained the Conjugate Gradient methods and illustrated its origin, starting from general projection methods. We have also introduced the principle of preconditioning. Now, we can move on to the principle and application of deflation.

Chapter 2

Deflation Methods

In the previous chapter, we have been introduced to the Conjugate Gradient method and its preconditioned version. In this chapter deflation will be introduced and we will see why and how it can be used to improve CG.

The deflation method was first introduced by Nicolaides [16], and has been applied in various situations since then. For example, Vuik et al. [31] have applied deflation with an Incomplete Cholesky (IC) preconditioner on CG for oil reservoirs in sandstone and scale layers. Tang [27] has applied several two-level PCG methods of which some use deflation to 2D and 3D bubbly flow problems. Jönsthövel [11] has applied deflation to composite materials, in particular to asphalt.

2.1 Principle of deflation

We are interested in improving the convergence and speed of iterative methods applied to linear systems with large condition numbers. In particular these linear systems arise from the Topology Optimization process that will be explained in later chapters. We have seen that the PCG method is an improvement on the CG method for such systems because it clusters the eigenvalues when replacing $\boldsymbol{A}$ with $\boldsymbol{M}^{-1}\boldsymbol{A}$, which improves the condition number. Another method to improve the convergence is to look at the extreme eigenvalues of $\boldsymbol{A}$ directly. The goal is to set some of the smallest eigenvalues to zero. For a matrix which has zero eigenvalues, the effective condition number is used. The effective condition number is equal to the ratio of the largest and smallest non-zero eigenvalues in absolute value. This means that setting the smallest eigenvalues to zero would reduce the effective condition number and improve the convergence of the CG method.

The principle of deflation is to split the solution $\boldsymbol{x}$ in a part that is immediately solvable and a part that is found from a different linear system. In this system, the small eigenvalues have been projected to zero, effectively increasing the smallest eigenvalue.

We will first introduce the necessary notation and definitions, before applying deflation to both CG and PCG.

Definition 2.1. *Let $\boldsymbol{A}$ be an SPD matrix as in Eq. 1.1, with d zero eigenvalues. Suppose $\boldsymbol{Z} \in \mathbb{R}^{n\times k}$, with full rank and $k < n - d$, is given. Then, we define the invertible Galerkin matrix, $\boldsymbol{E} \in \mathbb{R}^{k\times k}$, the correction matrix, $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$, and the deflation matrix, $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, as follows:*

$$\boldsymbol{P} \equiv \boldsymbol{I} - \boldsymbol{A}\boldsymbol{Q}, \quad \boldsymbol{Q} \equiv \boldsymbol{Z}\boldsymbol{E}^{-1}\boldsymbol{Z}^T, \quad \boldsymbol{E} \equiv \boldsymbol{Z}^T\boldsymbol{A}\boldsymbol{Z}. \tag{2.1}$$

The k columns of the deflation-subspace matrix $\boldsymbol{Z}$ are the deflation vectors. Determining the deflation vectors is the key part of the deflation method. We will investigate this in the next section.

One requirement for the deflation vectors is that they must be chosen in such a way that $\boldsymbol{E}$ is nonsingular. This is satisfied when $\mathcal{N}(\boldsymbol{A}) \cup \mathcal{R}(\boldsymbol{Z}) = \boldsymbol{0}$, i.e. that a nonzero vector is never both a linear combination of deflation vectors and in the null space of $\boldsymbol{A}$. For nonsingular $\boldsymbol{A}$, this condition is automatically satisfied, because then $\mathcal{N}(\boldsymbol{A}) = \boldsymbol{0}$.

In the following we will need some results using the matrices of Definition 2.1, which are stated in the following lemma:

Lemma 2.2. *Let $\boldsymbol{A}, \boldsymbol{Z}, \boldsymbol{E}, \boldsymbol{Q}$ and $\boldsymbol{P}$ be as in Def. 2.1, $\boldsymbol{x}$ and $\boldsymbol{b}$ as in Eq. 1.1, and let $\boldsymbol{Z}$ satisfy $\mathcal{N}(\boldsymbol{A}) \not\subseteq \mathcal{R}(\boldsymbol{Z})$. Then, the following equalities hold:*

a) $\boldsymbol{A}\boldsymbol{P}^T = \boldsymbol{P}\boldsymbol{A}$

b) $(\boldsymbol{I} - \boldsymbol{P}^T)\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{b}$

Proof

a) $\boldsymbol{A}\boldsymbol{P}^T = \boldsymbol{A}(\boldsymbol{I} - \boldsymbol{Q}\boldsymbol{A}) = \boldsymbol{A} - \boldsymbol{A}\boldsymbol{Q}\boldsymbol{A} = (\boldsymbol{I} - \boldsymbol{A}\boldsymbol{Q})\boldsymbol{A} = \boldsymbol{P}\boldsymbol{A}$

b) $(\boldsymbol{I} - \boldsymbol{P}^T)\boldsymbol{x} = (\boldsymbol{I} - (\boldsymbol{I}^T - \boldsymbol{A}^T\boldsymbol{Q}^T))\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{b}$

Recall that the goal is to solve Eq. 1.1. We will now split the solution $\boldsymbol{x}$ as explained above:

$$\boldsymbol{x} = (\boldsymbol{I} - \boldsymbol{P}^T)\boldsymbol{x} + \boldsymbol{P}^T\boldsymbol{x}. \tag{2.2}$$

The first part can be immediately calculated using Lemma 2.2a). This does not take many operations, as $\boldsymbol{E}$ and $\boldsymbol{Z}$ are smaller than $\boldsymbol{A}$. This leaves us with:

$$\begin{aligned} & \boldsymbol{x} = \boldsymbol{Q}\boldsymbol{b} + \boldsymbol{P}^T\boldsymbol{x} \\ \iff & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{Q}\boldsymbol{b} + \boldsymbol{A}\boldsymbol{P}^T\boldsymbol{x} \\ \iff & \boldsymbol{b} = \boldsymbol{A}\boldsymbol{Q}\boldsymbol{b} + \boldsymbol{P}\boldsymbol{A}\boldsymbol{x} \\ \iff & \boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = (\boldsymbol{I} - \boldsymbol{A}\boldsymbol{Q})\,\boldsymbol{b} \\ \iff & \boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{P}\boldsymbol{b}, \end{aligned} \tag{2.3}$$

where we have used Lemma 2.2b). The solution of this last system contains components of $\mathcal{N}(\boldsymbol{P}\boldsymbol{A})$, and is therefore not the actual solution, but a deflated solution, which we call $\hat{\boldsymbol{x}}$. The system we have is the deflated version of Eq. 1.1, the deflated system:

$$\boldsymbol{P}\boldsymbol{A}\hat{\boldsymbol{x}} = \boldsymbol{P}\boldsymbol{b}. \tag{2.4}$$

It can be proven that the original solution $\boldsymbol{x}$ can be found from the deflated solution $\hat{\boldsymbol{x}}$ (see for example [27]) using:

$$\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{b} + \boldsymbol{P}^T\hat{\boldsymbol{x}}. \tag{2.5}$$

2.2 Deflation vectors

As explained in the previous section, the success of the deflation methods depends on the choice of the deflation vectors. Generally speaking, one wants to take deflation vectors that span the space that is spanned by the eigenvectors corresponding to (some of) the very small eigenvalues of $\boldsymbol{A}$. This causes several of the small eigenvalues to be set to zero, so that the new smallest nonzero eigenvalue is higher than before, meaning that the effective condition number improves.

The part of the solution that corresponds to this space is then calculated in the first part of the right hand side of Eq. 2.2. The rest is calculated in the second part, where the iterative method is used. In each iteration the solution is projected onto the orthogonal complement of the space spanned by the eigenvectors corresponding to the very small eigenvalues of the (preconditioned) matrix [31], i.e., the deflation-subspace matrix $\boldsymbol{Z}$.

In this thesis, we distinguish several types of deflation vectors, which are explained below. As stated before, deflation vectors must be constructed such that the Galerkin matrix $\boldsymbol{E}$ is nonsingular.

2.2.1 Eigenvector deflation

The most straightforward way of taking deflation vectors is directly calculating the eigenvectors corresponding to the eigenvalues that one would like to set to zero. These eigenvectors then form the deflation-subspace matrix $\boldsymbol{Z}$. This type of deflation is the most effective in reducing the number of iterations, because instead of approximating the space spanned by the eigenvectors corresponding to

the smallest eigenvalues, one actually takes the exact eigenvectors. Note that for Preconditioned Deflated Conjugate Gradient, which will be introduced in Sec. 2.4, we need to take the eigenvectors of the preconditioned matrix, $\boldsymbol{M}^{-1}\boldsymbol{A}$, and not of $\boldsymbol{A}$.

Since the direct calculation of eigenvectors takes so much time that the beneficial effects of deflation are often nullified, eigenvector deflation is only suitable as a reference case for other types of deflation vectors. That is, the number of iterations needed to solve the linear systems is compared, while disregarding the needed computing time.

2.2.2 Subdomain deflation on physical regions

Within the domain, regions can be distinguished using difference in values of a physical variable. These regions will be referred to as physical regions. In the case of topology optimization, we will consider regions with similar density. There will be regions representing near-void, and regions representing solid material. In general we could take any set of regions that consist of adjacent elements. However, in order to benefit from deflation, the regions are often elements grouped by magnitude of coefficients. Large differences in coefficients are thus dealt with by treating these groups separately.

When using subdomain deflation, every region corresponds to one deflation vector. The deflation vectors are constructed by putting a 1 in the regions deflation vector at every index that corresponds to a node that is in that region. One node has to be in at most one region. Nodes that are on a boundary between regions have to be assigned to the region with the largest coefficients [11]. In the case of topology optimization, boundaries between near-void and solid material will be assigned to the solid material. This assignment is illustrated in Fig. 2.1. In general cases, the boundary nodes will be assigned to the region with the higher coefficient.

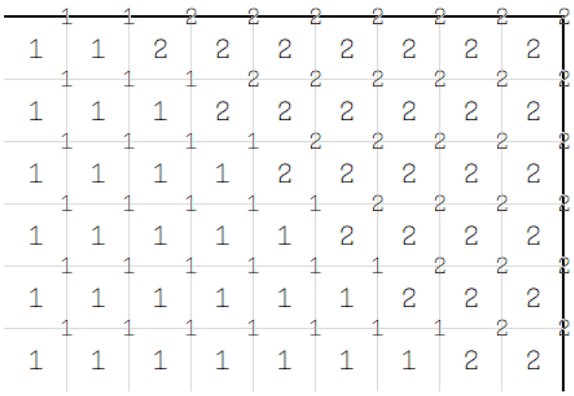


Figure 2.1: A schematic example of (part of) a 2D grid, illustrating the assignment of nodes to regions. In this example region 1 is a high-density region and region 2 is a low-density region. The numbers within a square (an element) represent the region that element belongs to, based on its value of the physical variable, and the numbers on a crossing (a node) represent the region that node is assigned to.

2.2.3 Rigid body modes deflation on physical regions

Rigid bodies are subsets of unconstrained connected elements in a finite-element setting. It is possible to construct the sub-stiffness matrix of a rigid body by assembling the corresponding element stiffness matrices. The regions from the previous subsection can also be considered as rigid bodies. There are not real rigid bodies, but a region with very small density effectively 'free-floating' in a surrounding region with density nearing 1 can be treated as a free-floating object that has rigid body modes.

Rigid body modes are the directions in which the unconstrained body can move: translations and rotations. In two dimensions, there are two possible translations and one rotation, see Fig. 2.2. In three dimensions, there are three translations and three rotations. For a two-dimensional problem, the three vectors representing these modes for a certain region will be used as deflation vectors. This is done for every region, so the number of deflation vectors forming the deflation-subspace matrix $\boldsymbol{Z}$ will be three times the number of regions.

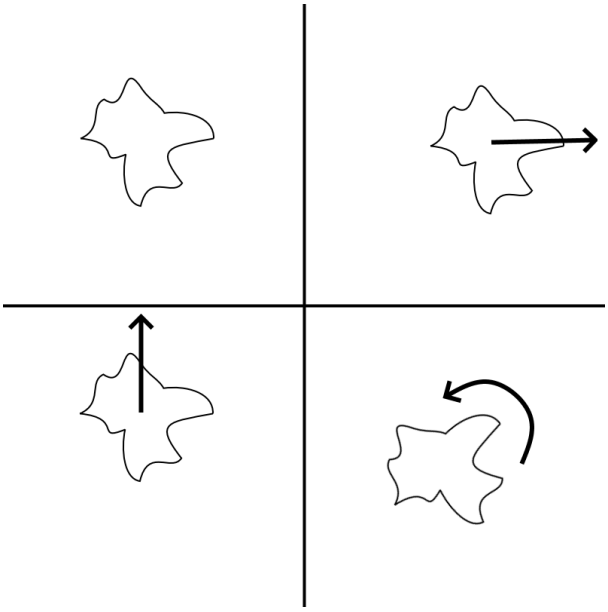


Figure 2.2: A representation of the rigid body modes of a two-dimensional rigid body. Top left: a rigid body. Top right: the translation in the x-direction. Bottom left: the translation in the y-direction. Bottom right: the rotation.

The rigid body modes of a region are constructed from the rigid body modes of its elements [5], [11]. Assume we have bilinear square elements. Then the coordinate vector of an element consists of eight entries: the x and y indices for the four adjacent nodes:

$$\begin{bmatrix} x_1 & y_1 & x_2 & y_2 & x_3 & y_3 & x_4 & y_4 \end{bmatrix}^T. \tag{2.6}$$

The straightforward way to define the translations is to take the x and y direction, which yields the translation vectors:

$$\begin{aligned} &\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}^T, \\ &\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T. \end{aligned} \tag{2.7}$$

These vectors can easily be extended to all elements in a region, by putting the ones correctly at every node in the region.

The rotation vector is expressed in the circular coordinate system, using r and ϕ:

$$\begin{bmatrix} r_1\cos(\phi_1) & r_1\sin(\phi_1) & r_2\cos(\phi_2) & r_2\sin(\phi_2) & r_3\cos(\phi_3) & r_3\sin(\phi_3) & r_4\cos(\phi_4) & r_4\sin(\phi_4) \end{bmatrix}^T, \tag{2.8}$$

$$\phi_j = \arctan\left(\frac{y_j}{x_j}\right). \tag{2.9}$$

When filling in ϕ_j, and using $r_j = \sqrt{x_j^2 + y_j^2}$, this reduces to

$$\begin{bmatrix} x_1 & y_1 & x_2 & y_2 & x_3 & y_3 & x_4 & y_4 \end{bmatrix}^T. \tag{2.10}$$

So the elements of the rotation vector are the (cartesian) coordinates of the elements with respect to an origin.

For example, let the upper left node of the element be the origin, and the four nodes be ordered with vertical lexicographic ordering. Then the rotation vector for the element becomes:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}^T. \tag{2.11}$$

Because of the translation vectors, the origin can be put anywhere: adding the translation vectors a certain amount of times to the rotation vector effectively moves the origin to wherever one would like it

to be. This means that when calculating the rotation vector for all elements in a region, we do not take a new origin for every element, but instead take one origin for all. We put the origin at the top left of the entire domain. The values in the rotation vector are then calculated with ease: the x-value for an element that is the n-th from the left is n, and the y-value for an element that is the m-th from the top is m.

We can apply this deflation type to the physical regions described in Sec. 2.2.2, as done for example by Lingen et al. [14].

2.2.4 Rigid body modes deflation on the entire domain

Now that we know how rigid body modes are calculated, we can also investigate the effect of using rigid body modes on the entire domain at once, without using regions, or knowledge about density. This case can give insight in the effect of rigid body modes separate from other effects.

With this deflation type we obtain three deflation vectors, containing all the nodes.

2.2.5 Rigid body modes deflation on square regions

Another variant of RBM deflation is to use regions that have been constructed without using the physical properties of the solution design: the easiest example is to use squares. Splitting up the domain in regular regions is specifically suited for parallel computations.

In this deflation type we treat each region separately as a rigid body (even though they do not necessarily share similar size of coefficients). We calculate the rigid body modes in similar way as for actual rigid body modes deflation, but now for square regions. These rigid body modes form the deflation vectors. There are three times as many deflation vectors as there are square regions. Note that the actual calculations will not be in parallel in this application.

2.2.6 Rigid body modes deflation on physical regions divided into smaller subregions

This deflation type combines the use of physical regions with the division of the domain. The physical regions are interlaced with the square regions. All subregions still consist of nodes with similar coefficients, because they are part of a normal physically defined region. Furthermore, no subregion can become larger than the squares in the previous deflation type, which preserves the advantage of splitting the domain in smaller regions.

We have explained the principle of deflation and introduced several types of deflation vectors. In the following sections, we provide algorithms that apply deflation to normal and preconditioned systems.

2.3 Deflated Conjugate Gradient

We can now apply deflation to the Conjugate Gradient method. It can be proven using properties of $\boldsymbol{P}$ and $\boldsymbol{A}$ that $\boldsymbol{PA}$ is SPSD if $\boldsymbol{A}$ is SPD [27], a necessity for CG to be used with the deflated system, Eq. 2.4. Furthermore, the deflated system is singular, since it contains zero eigenvalues, as discussed before. Kaasschieter [13] showed that CG and PCG can only be used on singular matrices if they are consistent, i.e. if there is a solution to the system. In this case, since $\boldsymbol{b} \in \mathcal{R}(\boldsymbol{A})$, we have $\boldsymbol{Pb} \in \mathcal{R}(\boldsymbol{PA})$ as well, showing that there is a solution. So we see that CG can be used on the deflated system.

We can now adapt Alg. 2 to get an algorithm for the Deflated Conjugate Gradient method (DCG). All vectors in the algorithm become deflated versions of the vectors in Alg. 2, but we will adhere to the convention of Tang [27] to only use the hat notation for $\boldsymbol{r}, \boldsymbol{w}$ and $\boldsymbol{x}$. An example: $\hat{\boldsymbol{r}}_j = \boldsymbol{P}\boldsymbol{r}_j = \boldsymbol{P}(\boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}_j)$.

Thus we obtain Alg. 4:

Algorithm 4 Deflated Conjugate Gradient

Choose $\boldsymbol{x}_0$ and ε
Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
Set $\hat{\boldsymbol{r}_0} \coloneqq \boldsymbol{P}\boldsymbol{r}_0, \boldsymbol{p}_0 \coloneqq \hat{\boldsymbol{r}}_0$ and $j \coloneqq 0$
while $\|\hat{\boldsymbol{r}}_j\|/\|\boldsymbol{b}\|/n > \varepsilon$ **do**
    $\hat{\boldsymbol{w}}_j \coloneqq \boldsymbol{P}\boldsymbol{A}\boldsymbol{p}_j$
    $\alpha_j \coloneqq \frac{(\hat{\boldsymbol{r}}_j, \hat{\boldsymbol{r}}_j)}{(\hat{\boldsymbol{w}}_j, \boldsymbol{p}_j)}$
    $\hat{\boldsymbol{x}}_{j+1} \coloneqq \hat{\boldsymbol{x}}_j + \alpha_j \boldsymbol{p}_j$
    $\hat{\boldsymbol{r}}_{j+1} \coloneqq \hat{\boldsymbol{r}}_j - \alpha_j \hat{\boldsymbol{w}}_j$
    $\beta_j \coloneqq \frac{(\hat{\boldsymbol{r}}_{j+1}, \hat{\boldsymbol{r}}_{j+1})}{(\hat{\boldsymbol{r}}_j, \hat{\boldsymbol{r}}_j)}$
    $\boldsymbol{p}_{j+1} \coloneqq \hat{\boldsymbol{r}}_{j+1} + \beta_j \boldsymbol{p}_j$
    $j \coloneqq j + 1$
end while
$\boldsymbol{x}_s = \boldsymbol{Q}\boldsymbol{b} + \boldsymbol{P}^T \hat{\boldsymbol{x}}_{j+1}$

In practice, DCG is not directly used for problems with large differences in the coefficients, because, as we will see shortly, it is essential to precondition these problems. DCG is an improvement on CG but using only deflation and no preconditioning likely results in many more than n iterations. Therefore, we will proceed with Preconditioned Deflated Conjugate Gradient, where both deflation and preconditioning are applied.

2.4 Preconditioned Deflated Conjugate Gradient

One can combine deflation and preconditioning by preconditioning a deflated system. If one preconditions the deflated system 2.4 one obtains:

$$\boldsymbol{M}^{-1}\boldsymbol{P}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{P}\boldsymbol{b}. \tag{2.12}$$

There is another variant which is deflating a preconditioned system. This leads to the following formulation:

$$\boldsymbol{P}^T\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{P}^T\boldsymbol{M}^{-1}\boldsymbol{b}. \tag{2.13}$$

The latter can be named Deflated Preconditioned Conjugate Gradient (DPCG), and it uses a slightly different algorithm than the former, Preconditioned Deflated Conjugate Gradient (PDCG). More information about both variants can be found in [7] and [27]. In addition to the transformations in Eq. 1.18 we now have the following preconditioned matrices:

$$\begin{aligned} \widetilde{\boldsymbol{Z}} &= \boldsymbol{M}^{\frac{1}{2}}\boldsymbol{Z}, \\ \widetilde{\boldsymbol{P}} &= \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{P}\boldsymbol{M}^{\frac{1}{2}}. \end{aligned} \tag{2.14}$$

PDCG can be described by the following algorithm:

Algorithm 5 Preconditioned Deflated Conjugate Gradient

Choose $\boldsymbol{x}_0$ and ε
Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$ and $\hat{\boldsymbol{r}}_0 = \boldsymbol{P}\boldsymbol{r}_0$, solve $\boldsymbol{M}\boldsymbol{y}_0 = \hat{\boldsymbol{r}}_0$
Set $\boldsymbol{p}_0 \coloneqq \boldsymbol{y}_0$ and $j \coloneqq 0$
while $\|\hat{\boldsymbol{r}}_j\|/\|\boldsymbol{b}\|/n > \varepsilon$ **do**
    $\hat{\boldsymbol{w}}_j \coloneqq \boldsymbol{P}\boldsymbol{A}\boldsymbol{p}_j$
    $\alpha_j \coloneqq \frac{(\hat{\boldsymbol{r}}_j, \boldsymbol{y}_j)}{(\boldsymbol{p}_j, \hat{\boldsymbol{w}}_j)}$
    $\hat{\boldsymbol{x}}_{j+1} \coloneqq \hat{\boldsymbol{x}}_j + \alpha_j \boldsymbol{p}_j$
    $\hat{\boldsymbol{r}}_{j+1} \coloneqq \hat{\boldsymbol{r}}_j - \alpha_j \hat{\boldsymbol{w}}_j$
    Solve $\boldsymbol{M}\boldsymbol{y}_{j+1} = \hat{\boldsymbol{r}}_{j+1}$
    $\beta_j \coloneqq \frac{(\hat{\boldsymbol{r}}_{j+1}, \boldsymbol{y}_{j+1})}{(\hat{\boldsymbol{r}}_j, \boldsymbol{y}_j)}$
    $\boldsymbol{p}_{j+1} \coloneqq \boldsymbol{y}_{j+1} + \beta_j \boldsymbol{p}_j$
    $j \coloneqq j + 1$
end while
$\boldsymbol{x}_s = \boldsymbol{Q}\boldsymbol{b} + \boldsymbol{P}^T \hat{\boldsymbol{x}}_{j+1}$

The method adheres to the following convergence result [27], which follows logically from the previous (Eq. 1.21):

$$\|\hat{\boldsymbol{x}} - \hat{\boldsymbol{x}}_{j+1}\|_A \leq 2\|\hat{\boldsymbol{x}} - \hat{\boldsymbol{x}}_0\|_A \left(\frac{\sqrt{\kappa_{\text{eff}}(\boldsymbol{M}^{-1}\boldsymbol{P}\boldsymbol{A})} - 1}{\sqrt{\kappa_{\text{eff}}(\boldsymbol{M}^{-1}\boldsymbol{P}\boldsymbol{A})} + 1} \right)^{j+1}. \tag{2.15}$$

2.5 Application of methods to a one-dimensional example

Now that we have seen the four variations of the Conjugate Gradient Method that one can construct using preconditioning and deflation, we can apply these to a meaningful example in order to investigate their performance and behaviour. We do this in Matlab. The example is a one-dimensional Poisson problem, with a Dirichlet boundary condition on the left and a Neumann boundary condition on the right. The example is taken from Jönsthövel [11]:

$$\begin{aligned} -\frac{d}{dx}\left(c(x)\frac{du(x)}{dx}\right) &= f(x), \quad x \in [0, \ell] \\ u(0) = 0, \quad \frac{du}{dx}(\ell) &= 0. \end{aligned} \tag{2.16}$$

Here $u(x)$ is the unknown displacement, $f(x)$ the given source term and $c(x)$ is piecewise constant over the line $[0, \ell]$. The line is divided in three equally large regions, between which the $c(x)$ differs by several orders of magnitude. So we have the large difference in coefficients in this example: the case that we wish to accelerate using deflation. Eq. 2.16 is discretized using linear first-order shape functions and equally spaced elements of size h. This yields the following finite element stencil:

$$\begin{pmatrix} c(x_i) & -c(x_{i+1}) \\ -c(x_i) & c(x_{i+1}) \end{pmatrix} \tag{2.17}$$

On the left side the boundary is eliminated, so $x_1 = h$, and on the right side it is not, so $x_n = \ell$. We will use $c_1 = 1, c_2 = 10^4$ and $c_3 = 10^8$ for $c(x_i)$, corresponding to the three regions. The discretization yields the following:

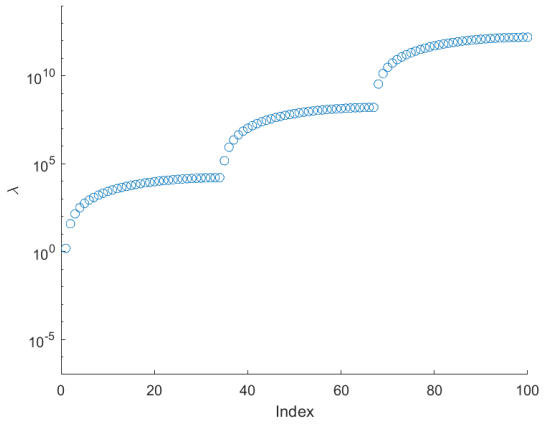
$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}(\boldsymbol{x}), \tag{2.18}$$

where $\boldsymbol{u} = [u_1, ..., u_n]^T, \boldsymbol{x} = [x_1, ..., x_n]^T$ and

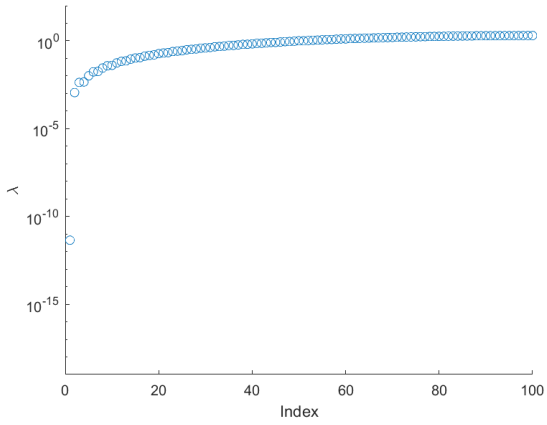
$$\boldsymbol{K} = \frac{1}{h^2} \begin{pmatrix} 2c_1 & c_1 & & & & & & & \\ -c_1 & \ddots & \ddots & & & & & & \\ & \ddots & 2c_1 & -c_1 & & & & & \\ & & -c_1 & c_1 + c_2 & -c_2 & & & & \\ & & & -c_2 & \ddots & \ddots & & & \\ & & & & \ddots & 2c_2 & -c_2 & & \\ & & & & & -c_2 & c_2 + c_3 & -c_3 & \\ & & & & & & -c_3 & \ddots & \ddots \\ & \emptyset & & & & & & \ddots & 2c_3 & -c_3 \\ & & & & & & & & -c_3 & c_3 \end{pmatrix}$$

Matrix $\boldsymbol{K}$ is the matrix of interest (which was called $\boldsymbol{A}$ in the theory above). When applying preconditioning, we will use Jacobi, i.e. $\boldsymbol{M} = \text{diag}(\boldsymbol{K})$, as discussed before. When applying deflation, we will use subdomain deflation as in Sec. 2.2.2 with the three regions we have seen. In the following we will use $n = 100, \ell = \frac{\pi}{2}, \varepsilon = 10^{-6}$ and $f = [1, ..., 1]^T$.

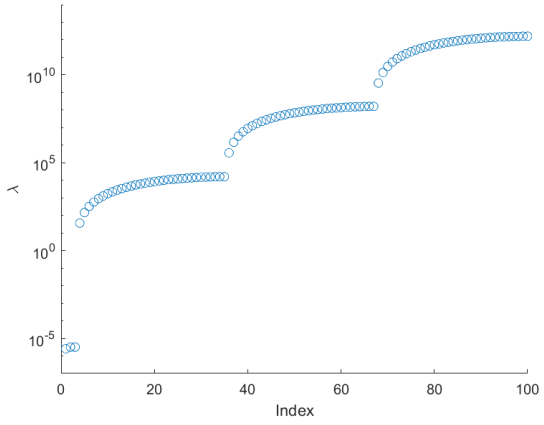
Before continuing to the results, we can inspect the spectra of the matrices corresponding to CG, PCG, DCG and PDCG. These are shown in Fig. 2.3.



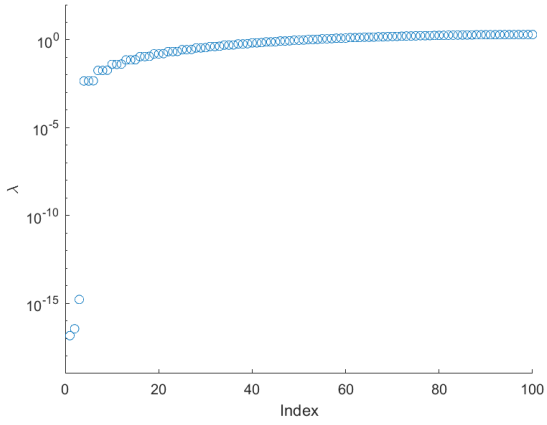
(a) The spectrum of $\boldsymbol{K}$, to be used for the CG method.



(b) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{K}$, to be used for the PCG method.



(c) The spectrum of $\boldsymbol{PK}$, to be used for the DCG method.



(d) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{PK}$, to be used for the PDCG method.

Figure 2.3: Visualization of the spectrum of the matrices used for solving Eq. 2.18 with CG, PCG, DCG and PDCG, respectively. Every circle denotes an eigenvalue. Eigenvalues have been sorted by magnitude.

All eigenvalues are calculated and sorted by magnitude. These spectra show that preconditioning clusters the eigenvalues around 1 and deflation puts the smallest eigenvalues to a value close to zero. As the calculation are performed to machine precision, this does not happen exactly. The original spectrum clearly shows the influence of the large difference in coefficients, and preconditioning and deflation improve the spectrum. The best spectrum is that of $\boldsymbol{M}^{-1}\boldsymbol{PK}$. We see that in both deflated cases, the 'zero' eigenvalues relate to the maximal eigenvalue by a factor of about the machine precision (10^{-16}). The original condition number was $\kappa(\boldsymbol{K}) = 1.06 \cdot 10^{12}$. Deflation alone yields a condition number of $\kappa_{\text{eff}}(\boldsymbol{PK}) = 4.41 \cdot 10^{11}$. Preconditioning alone yields a condition number of $\kappa(\boldsymbol{M}^{-1}\boldsymbol{K}) = 1.77 \cdot 10^{3}$ (when disregarding the outlier). When using both the condition number is reduced to $\kappa_{\text{eff}}(\boldsymbol{M}^{-1}\boldsymbol{PK}) = 4.42 \cdot 10^{2}$, which is an enormous reduction compared to the original.

Algorithms 2 to 5 have been implemented in Matlab for this example. All methods converge, but in very different ways. The solution to this system is shown in Fig. 2.4.

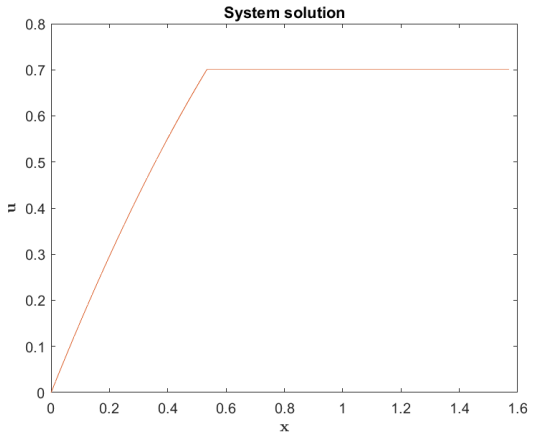
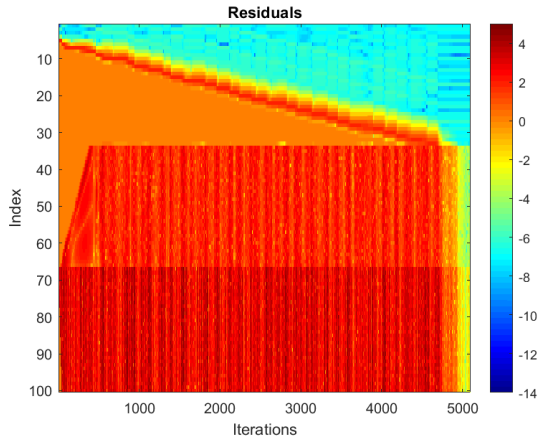
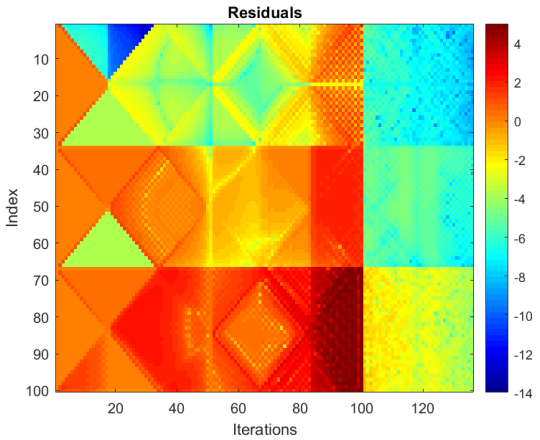


Figure 2.4: The solution to system 2.18.

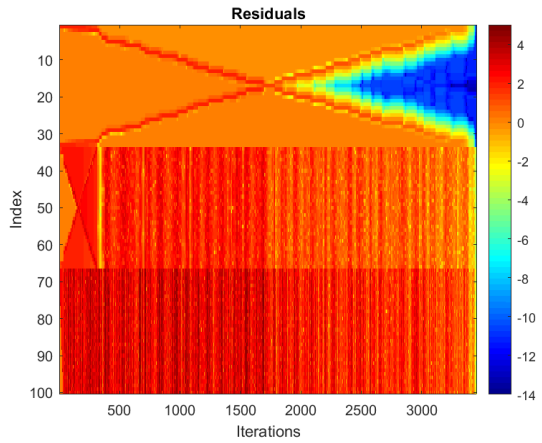
What is needed to reach this solution for each of the methods is shown in Fig. 2.5:



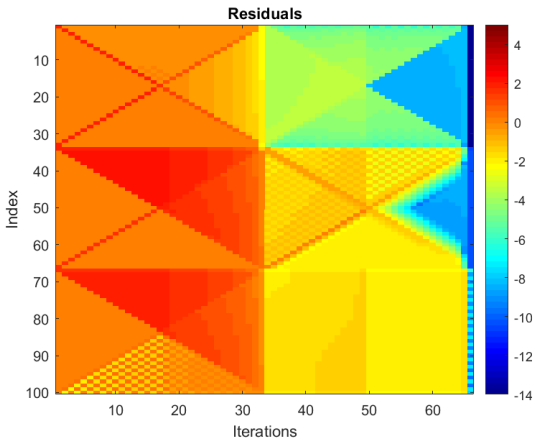
(a) Residuals in all iterations of the CG method.



(b) Residuals in all iterations of the PCG method.



(c) Residuals in all iterations of the DCG method.



(d) Residuals in all iterations of the PDCG method.

Figure 2.5: Visualization of the behaviour of the residuals when solving the one-dimensional example of size $n = 100$ with CG, PCG, DCG and PDCG. Every column corresponds to the residual vector from the iteration on the x-axis. The index on the y-axis is the index in each residual vector. The colour of a value shows the decimal logarithm of the residual value.

From these figures we can draw several conclusions:

- In all figures the influence of the division in three regions on the residuals is clear. In general, the residuals vanish faster in the region with the smaller coefficient ($c_1 = 1$) and slower in the region with the higher coefficient ($c_3 = 10^8$).
- As mentioned before, for matrices with large condition numbers, the CG method can break down, i.e. take more than n iterations. This is what we see in Fig. 2.5a. Applying deflation helps to reduce the number of iterations, but the number is still extraordinarily large (Fig. 2.5c).
- Preconditioning makes a very large difference in the number of iterations, as expected. In the case of PCG, Fig. 2.5b, it can be seen that after $n = 100$ iterations, there is a jump in the residual values. At this point the 'natural' accuracy is reached. Recall that in exact arithmetic, CG converges to the exact value. However, numerical calculations do not use exact arithmetic. A qualitative estimation of the expected accuracy is the product of the machine precision and the matrix condition number. In this case this is in the order of $10^{-16+12} = 10^{-4}$. The extra iterations seen in the figure bring the residual norm from 10^{-4} to 10^{-6}. However, during these iterations the change in the actual solution $\boldsymbol{u}$ is negligible. For the PCG method, it is therefore wise to adapt the termination criterion to this estimate of 10^{-4}, in order to prevent wasting computation power.
- In Fig. 2.5d, we see that less than n iterations are necessary to obtain the required accuracy. The values of the residuals are more evenly distributed over the domain than without deflation. Where PCG treats the entire domain at once, PDCG with region deflation reduces the residuals within the regions parallel to each other. It also has three 'stages' of converging, i.e. there are jumps in residual values for every region, instead of one at $n = 100$: at the 33th and 67th iteration. In Fig. 2.6 the final accuracy of PDCG is illustrated: after 100 iterations the best result is reached. The residual norm there is of the order of 10^{-10}. In conclusion: in this case, the residuals vanish faster using PDCG and in as many iteration groups as there are regions.

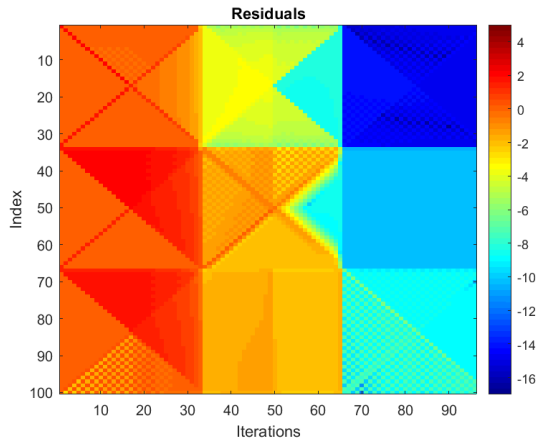


Figure 2.6: Residuals in all iterations of the PDCG method, using $\varepsilon = 10^{-10}$. Note the difference in limits of the colour axis compared to Fig. 2.5.

Chapter 3

Static Topology Optimization

Topology optimization is the process that finds the optimal shape of an object that is to be designed, by finding the extremum of a certain objective function, subject to certain conditions. The conditions can apply to the shape directly, or for example to the total volume. Before we proceed to the problem that is the focus of this thesis, which is a three-dimensional dynamic problem, we will focus on a two-dimensional static problem. We use a classical problem in topology optimization, the MBB-beam (Messerschmidt-Bolkow-Blohm) [24]. An example of a solution to this problem is shown in Fig. 3.1. Because it is symmetric, only half of the domain has to be optimized.

There are several techniques to perform topology optimization, with differing objective functions and different ways of quantifying the shape. There are three (classes of) techniques for topology optimization [10], [24]:

- Evolutionary Structural Optimization (ESO). In the initial state all elements have density one. At every iteration the solution is used to determine the effectiveness of every element. The least effective elements which are below a rejection criterion are removed. This is done until an optimal solution is found. One of the variations is an algorithm where elements can also be put back into the design (bi-directional algorithm).
- Level Set Method (LSM). This method uses moving material boundaries constructed by summing so-called level set functions. The boundaries are between the material and the void. These boundaries can form complicated shapes without having to be defined as functions. They are not related to the mesh anymore.
- Solid Isotropic Material with Penalization (SIMP). Also called "power-law approach". Every element in the mesh is assigned a density between zero and one, and in every iteration the densities are updated using an optimization procedure.

In this chapter we will be using the SIMP method. We will follow the explanation and use the code of Sigmund [24], which was later followed by Andreassen et al. [2]. They provide insight in the two-dimensional topology optimization problem. Applying deflation to this problem provides an indication of the degree of improvement deflation will be to the actual problem, and it is a test area for different types of deflation vectors. After this, we proceed to the dynamic topology optimization of the wafer stage.

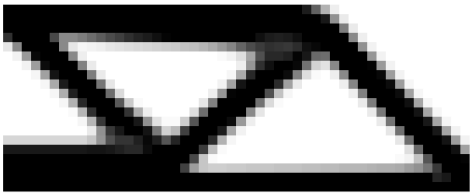


Figure 3.1: An example of a solution to the MBB half-beam problem using the 88 line code from Andreassen et al. [2]. White corresponds to void, black to density one. The problem is the MBB half-beam and the problem size is 50×20.

In this chapter, some of the symbols from the previous chapter do not have the same meaning as before. For a list of symbols for this chapter, see Appendix A.3

3.1 The 2D Topology Optimization problem

The papers mentioned above apply the SIMP technique to the MBB half-beam optimization problem in 2D, and give and explain a 99-line and 88-line code, respectively. The MBB half-beam problem is a classical problem in topology optimization. It is depicted in Fig. 3.2. Neumann conditions are applied where the beam is cut in half. A support is located in the lower right, forming a Dirichlet boundary condition there. A force is located on the top left (in the middle of the full domain).

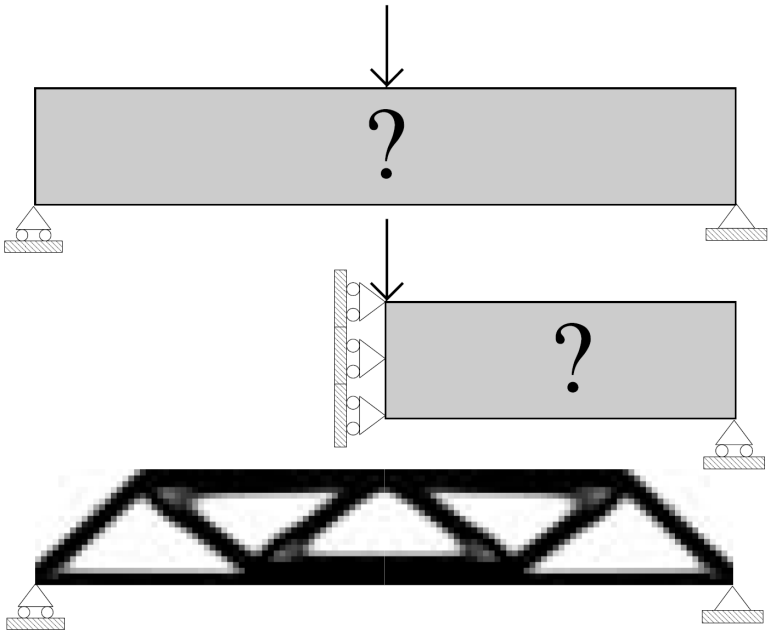


Figure 3.2: Schematic representation of the MBB-problem. Top: full design domain. Middle: half design domain with symmetry boundary conditions. Bottom: resulting optimized full beam. Taken from Sigmund [24].

In the SIMP technique, the domain is discretized into elements, in which material properties are assumed

to be constant. The known parameters are the stiffness (elasticity) of the material, the sizes of the grid, and the boundary conditions. The variables are the element densities. The densities can vary from 0 (void) to 1 (solid material). At the end of the optimization routine, most elements will have a non-integer density value, which will have to be rounded in order to know whether there will or will not be material in that element in the final design. The most simple rule is to keep density $\frac{1}{2}$ as the border value, and round to either 0 or 1. Note that this will not automatically yield a printable design, because the structure could still contain disconnected elements.

In these papers, a 2D rectangular problem with square finite elements, using columnwise element and node numbering, is assumed (see Fig. 3.3). The optimization problem using the SIMP technique that can be formulated for this is given by:

$$\begin{aligned} \min_{x}: \quad & c(\boldsymbol{x}) = \boldsymbol{u}^T \boldsymbol{K}(\boldsymbol{x}) \boldsymbol{u} = \sum_{e=1}^{N} E_e(x_e) \boldsymbol{u}_e^T \boldsymbol{k}_0 \boldsymbol{u}_e \\ \text{subject to:} \quad & \frac{V(\boldsymbol{x})}{V_0} = f_V \\ & \boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} \\ & \boldsymbol{0} < \boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{1}, \end{aligned} \tag{3.1}$$

where

$$E_e(x_e) = E_{\min} + x_e^p (E_0 - E_{\min}), \tag{3.2}$$

and where c is the compliance, $x_e \in [0, 1]$ is the element density, $\boldsymbol{u}$ and $\boldsymbol{f}$ are the global displacement and force vectors, respectively, $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is the global stiffness matrix, n is the number of degrees of freedom (two times the number of nodes), $\boldsymbol{u}_e$ is the e-th element displacement vector, $\boldsymbol{k}_0$ is the element stiffness matrix for an element with a unit Young's modulus, $\boldsymbol{x}$ is the vector of design variables (i.e. of element densities), N is the number of elements in the design domain, $V(\boldsymbol{x})$ and V_0 are the material volume and the (constant) design domain volume, respectively, f_V is the prescribed volume fraction, E, E_0 and $E_{\min}$ are elasticity moduli and p is the penalization power [2].

The compliance is a parameter that measures the inverse of stiffness. One would like to minimize the compliance in order to maximize the structural integrity. The displacement vector $\boldsymbol{u}$ shows how much an element can move away from its rest position. The force vector $\boldsymbol{f}$ contains the forces put on the object and the boundary conditions on the displacement of some of the elements. The element stiffness matrix $\boldsymbol{k}_0$ is calculated from the finite element techniques using a square bilinear element. The penalization power often has the value 3 and helps to move the density values to either 0 or 1 and to avoid checkerboard patterns. The minimum stiffness $E_{\min}$ is enforced to prevent singularities in the stiffness matrix.

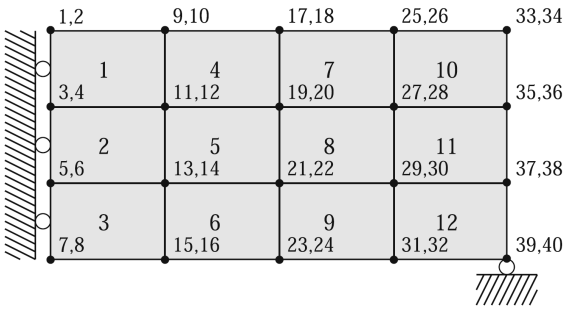


Figure 3.3: An example of the MBB half-beam, discretized into 12 elements, with corresponding element and node numbers. Taken from Andreassen et al. [2].

In order to solve this optimization problem, the compliance and density values are updated in every optimization iteration. The iteration terminates when the change in design is smaller than a threshold value. In every iteration the following steps occur:

- The global stiffness matrix $\boldsymbol{K}$ is assembled from the 8×8 square bilinear 4-node element stiffness matrices.
- The displacement vector $\boldsymbol{u}$ is solved from the global stiffness matrix $\boldsymbol{K}$ and the force vector $\boldsymbol{f}$.

- The compliance is calculated as above, using the stiffness matrix, the displacement vector and the densities.
- The sensitivities (i.e. derivatives) of the compliance and the volume with respect to the element densities are calculated.
- The densities are updated using the old densities and the sensitivies.

These steps are visualized in a graph in Fig. 3.4.

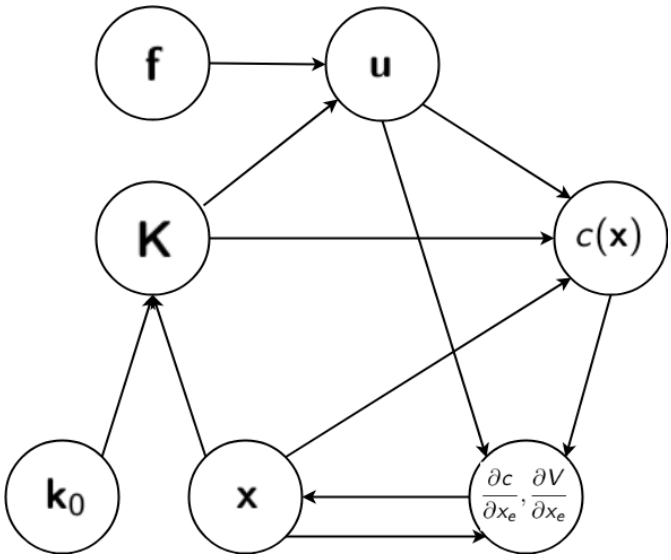


Figure 3.4: A graph showing the steps taken in the optimization iteration of the 2D half-beam problem. The graph is followed clockwise.

For more details, see Sigmund [24] or Andreassen et al. [2].

3.2 Applying deflation to the problem

When solving for the displacement vector $\boldsymbol{u}$, an equation of the type of Eq. 1.1 has to be solved. The global stiffness matrix $\boldsymbol{K}$ has size $n \times n$. It is sparse: since every element only interacts with their nearest neighbours, the (full) bandwidth of $\boldsymbol{K}$ has to be $\mathcal{O}(2 \cdot \max(n_x, n_y))$, when n_x and n_y are the number of nodes in the x- and y-direction, respectively. There is also a large difference in coefficients which springs from the difference between near-void regions and regions with density 1. This causes a very large condition number. Lastly, $\boldsymbol{K}$ is SPD. This is because it consists of the element matrices, which are SPSD. A physical argument is that the stiffness is always positive, so $\boldsymbol{K}$ will always have positive eigenvalues.

We see that all ingredients are present to apply deflation to this problem. We use the code provided by [2], but replace the backward solve for $\boldsymbol{u}$ by a function that applies PCG and one that applies PDCG. The latter can apply either of the first four types of deflation vectors explained in section 2.2. Both functions use a Jacobi preconditioner.

For eigenvector deflation, we take the eigenvectors corresponding to the smallest eigenvalues of $\boldsymbol{M}^{-1}\boldsymbol{K}$. In order to compare correctly to rigid body modes deflation, the amount of eigenvalues taken will be three times the number of near-void regions, in every optimization iteration. Then, the number of deflation vectors is the same is with rigid body modes deflation, because there we have the three rigid body modes for every region.

The regions in this case are density-based. They are determined by a region-growing algorithm: this algorithm starts with an element with a density below or above the threshold of $x_e = 10^{-3}$ and appends everything to it that is adjacent and also at the same side of the threshold, recursively. Only interior region nodes, i.e. those nodes that have all four neighbouring elements in the same region, are assigned

values for the rigid body modes. This means for example that regions that consist of a line of elements are disregarded fully, because they do not contain interior nodes.

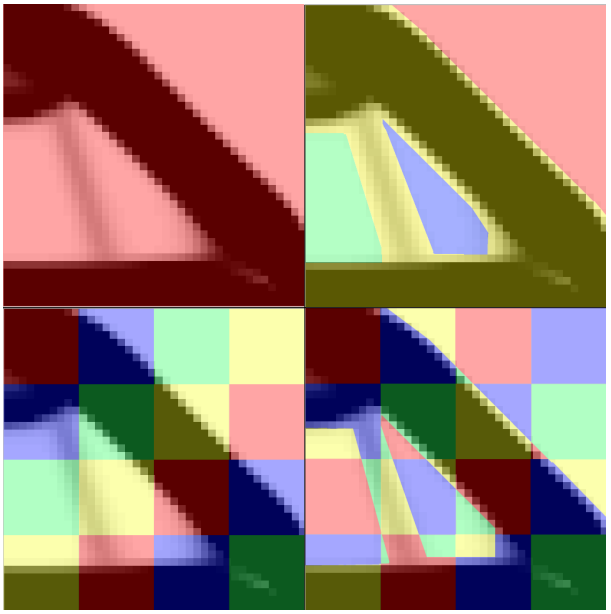
A region can also contain only one element. This is undesirable, because the deflation vectors corresponding to this region will then be linearly dependent: the rotation vector can be constructed from the translation vectors. In this case we disregard the region as well.

As explained in Sec. 2.2.3, we take the left top of the region as the origin for the rotation vectors.

For deflation with square regions we divide the domain in blocks of 10×10 elements, and calculate the rigid body modes of these regions, as explained in Sec. 2.2.5. This cannot be done for the entire domain, since there are fixed degrees of freedom that are not in matrix $\boldsymbol{K}$. If there are fixed degrees of freedom in a region, they cannot be assigned a value in a deflation vector because there simply is no index corresponding to them. The rigid body is therefore not always the full region.

In Fig. 3.5 an example is shown of the regions used in every type of rigid body modes deflation for this two-dimensional topology optimization problem.

(a) Solution of the 40×40 half-beam problem at optimization iteration 12.



(b) Top left: the region when using the full domain.
Top right: the regions when using density-based regions.
Bottom left: the regions when using square regions.
Bottom right: the regions when using divided density-based regions.

Figure 3.5: An example of the regions made for the different types of rigid body modes deflation. The solution shown is from the 40x40 half-beam problem at optimization iteration 12.

3.3 Results and preliminary conclusions

In the previous section we have seen how the 2D topology optimization code is structured and how we can compare the PDCG method to the PCG method in this case. In Table 3.1 and Table 3.2 we see this comparison, for all the deflation types we have seen in Sec. 2.2. The calculations were done using Matlab.

The average number of Krylov iterations per optimization iteration and the time taken in seconds for the full optimization process are shown in Table 3.1. Several grid sizes are investigated to see the influence of scaling on performance of the different deflation types.

Table 3.1: Table of average number of Krylov iterations per optimization iteration and timing in seconds (in parentheses) for PCG and PDCG for several sizes of the MBB half-beam.

		50×20	40×40	60×60	80×80
Method	Defl. type	$n = 2142$	$n = 3362$	$n = 7442$	$n = 13122$
PCG	N/A	414 (1.7)	320 (5.0)	518 (13)	669 (69)
PDCG	Eigenvectors	74 (2.8)	60 (9.6)	64 (47)	74 (56)
PDCG	Subdomain on physical regions	405 (4.1)	296 (14)	469 (39)	574 (127)
PDCG	RBM on full domain	342 (1.8)	240 (5.6)	357 (13)	430 (90)
PDCG	RBM on physical regions	343 (3.9)	240 (13)	357 (27)	430 (158)
PDCG	RBM on square regions	168 (1.5)	98 (5.1)	100 (28)	91 (65)
PDCG	RBM on divided physical regions	168 (2.5)	98 (10)	100 (56)	91 (132)

In Table 3.2 the effective condition number at optimization iteration 7 for a grid size of 40×40 is shown. The reason iteration 7 is taken is because at the first few iterations the regions have not yet started to form, which is necessary to correctly compare the deflation types.

Fig. 3.6 and Fig. 3.7 correspond to this table. They display the spectrum of the PCG method and the spectra of the different PDCG methods, respectively. The eigenvalues of the applicable matrix were calculated and sorted. Again, in all cases, the grid size is 40×40 and the optimization iteration number is 7.

Table 3.2: Table of average number of Krylov iterations per optimization iteration and effective condition numbers for PCG and PDCG at the seventh optimization iteration, for the MBB half-beam with size $40 \times 40, n = 3362$.

Method	Defl. type	Krylov it per opt. it.	$\kappa_{\text{eff}}(\boldsymbol{M}^{-1}\boldsymbol{P}\boldsymbol{K})$ at it. 7
PCG	Not Applicable	320	$5.017 \cdot 10^4$
PDCG	Eigenvectors	60	$9.932 \cdot 10^2$
PDCG	Subdomain on physical regions	296	$2.894 \cdot 10^4$
PDCG	RBM on full domain	240	$1.883 \cdot 10^4$
PDCG	RBM on physical regions	240	$1.833 \cdot 10^4$
PDCG	RBM on square regions	98	$2.974 \cdot 10^3$
PDCG	RBM on divided physical regions	98	$2.970 \cdot 10^3$

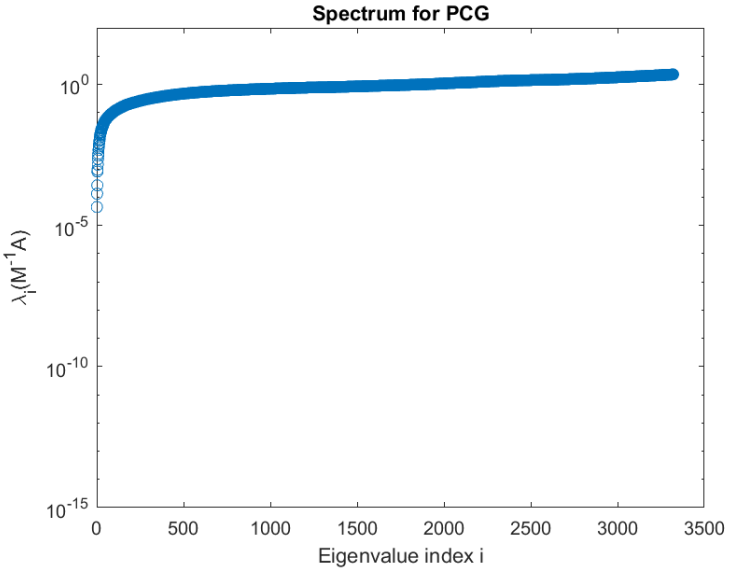
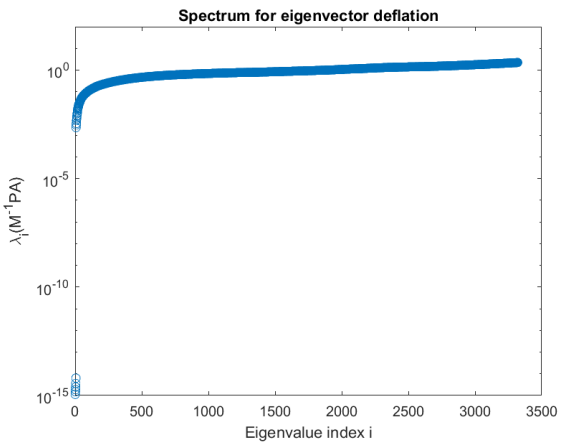
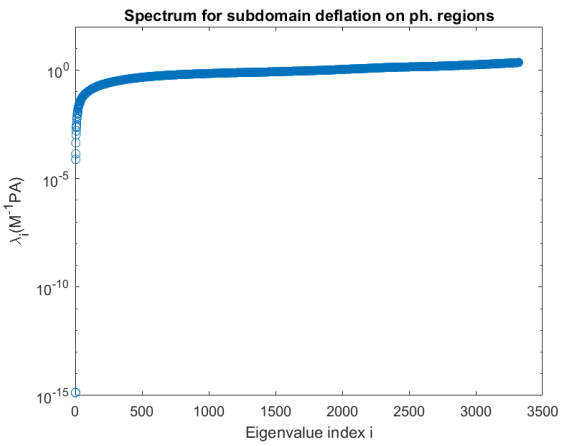


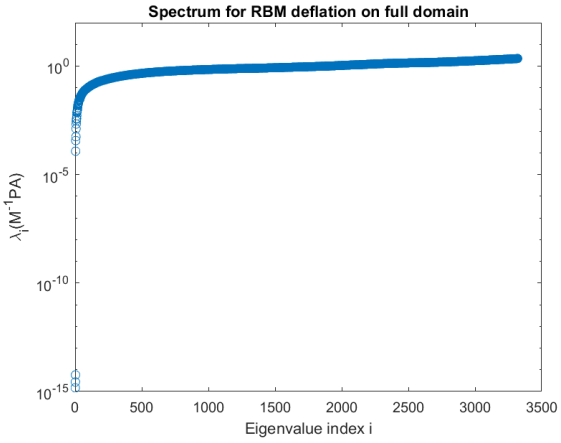
Figure 3.6: The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{A}$ at the seventh optimization iteration for a grid size of 40×40, where the PCG method is used. The eigenvalues are sorted by ascending magnitude.



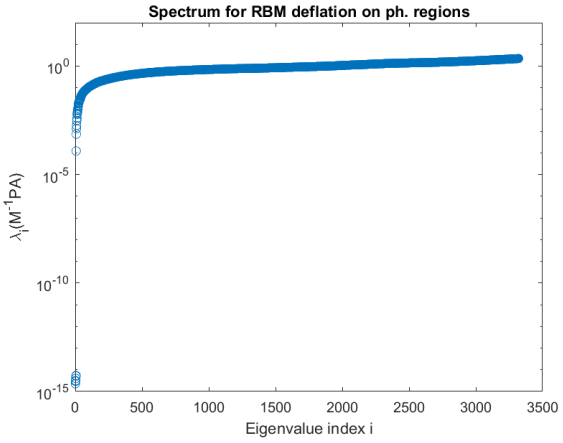
(a) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{PA}$ using eigenvector deflation.



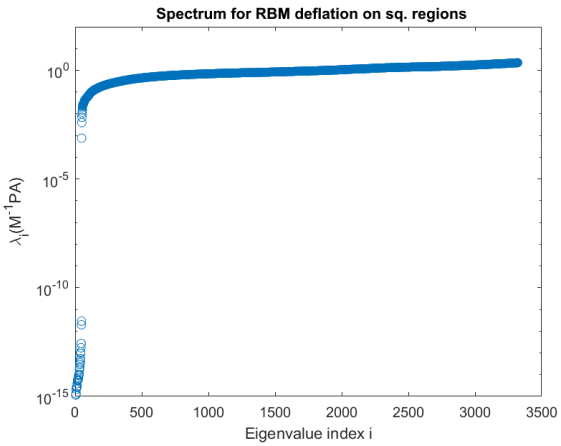
(b) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{PA}$ using subdomain deflation on physical regions.



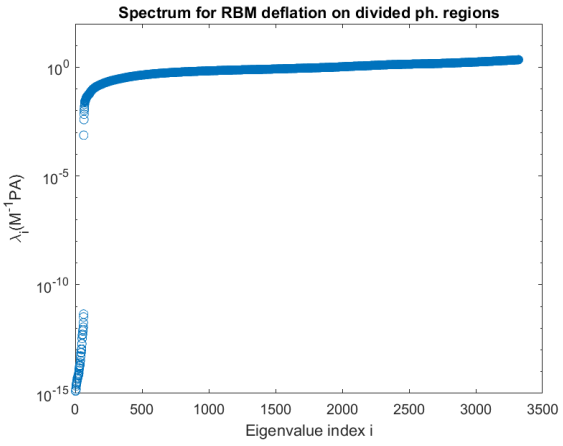
(c) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{PA}$ using rigid body modes deflation on the entire domain.



(d) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{PA}$ using rigid body modes deflation on physical regions.



(e) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{PA}$ using rigid body modes deflation on square regions.



(f) The spectrum of $\boldsymbol{M}^{-1}\boldsymbol{PA}$ using rigid body modes deflation on physical regions divided into smaller subregions.

Figure 3.7: The spectra of $\boldsymbol{M}^{-1}\boldsymbol{PA}$ at the seventh optimization iteration for a grid size of 40×40 and for different deflation types. The eigenvalues are sorted by ascending magnitude.

From these tables a few preliminary conclusions can be drawn:

- We see that for all sizes the PDCG method needs less Krylov iterations per optimization iteration to solve the linear system than the PCG method. The best deflation vectors are those using eigenvector deflation, which is in line with what we expect from Sec. 2.2.1. The worst are those using subdomain deflation. This is because there are less deflation vectors than for rigid body modes, and because the properties of the rigid bodies are not exploited.
- In between these extremes, we see the variants of rigid body modes deflation. It appears that by far the best variants are those using square regions. We see that for rigid body modes on density-based regions, the difference between PCG and PDCG is not as striking as one might expect from looking at the result for eigenvector deflation or other results: applying deflation regularly yields more than a 50% drop in the number of iterations, for example in Vuik et al. [31] or Tang [27]. The result for squares is in line with this.
- We see in Table 3.1 that there is little to no difference between deflation types that differ only in their use of physical regions, i.e. full domain and physical regions are similar, and square regions and divided physical regions are similar. It appears that using the information about the local density does little to nothing to improve the quality of the deflation vectors in this case. For this reason, we will not be using it in the 3D case.
- We see that the effective condition number of the stiffness matrix is improved with all deflation types, in a similar way as with the iteration count. Deflation types with lower Krylov iteration count per optimization iteration also have a lower effective condition number, which was expected.

We have seen that the deflation types treated in this chapter all improve on the Krylov iteration count to a reasonable extent. The problem we have considered in this chapter is a standard problem in topology optimization. In the next chapters we will consider the application we are really interested in: the 3D dynamic topology optimization of the design of a wafer stage. Based on our findings in this chapter, we will consider the following deflation types:

1. Eigenvector deflation
2. Rigid body modes deflation on the entire domain
3. Rigid body modes deflation on cubic regions

In the next chapter, the theory behind the 3D dynamic topology optimization problem will be reviewed, and after this deflation will be applied to matrix equations appearing in the optimization procedure.

Chapter 4

Dynamic Topology Optimization

The problem of interest in this thesis is the 3D wafer stage topology optimization problem. This is an optimization problem which maximizes the smallest eigenfrequencies of the structure. It is a dynamic problem instead of a static problem, i.e. it concerns a moving structure.

In the dynamic topology optimization process several matrix equations with very many degrees of freedom have to be solved. We will start by introducing the optimization problem and the reduced ordel mode, so t hat we can identify where these matrix equations originate from and acquaint ourselves with the efforts that have already been made to solve them in reasonable time, before introducing deflation to them.

In this chapter, some of the symbols from the previous chapter do not have the same meaning as before. For a list of symbols for this chapter, see Appendix A.3

4.1 The wafer stage problem

The wafer stage is a free-floating stage on which computer chips are printed with a laser. This is done by moving the stage with the chip on it, underneath the laser. The stage is moved by magnetic actuators. Because the chips have to be printed with great accuracy, the movement of the stage has to be very precise, and it is therefore necessary to design it in such a way that no unwanted resonances or instabilities occur. This is called precision motion system design [29]. The optimization process uses knowledge from several fields, such as structural mechanics, control engineering and thermal analysis, and is considerably more complicated than for example the optimization of the MBB-beam.

The optimization is again performed by using cartesian finite elements. Since this is a real-world, three-dimensional problem, the elements are now cubes. To reach the required accuracy, the elements have to become smaller and smaller. The dimensions of the wafer stage have a ratio of 6 to 6 to 1. The stage will have to have sizes 40 by 40 by 6.7 cm. Every element introduces three degrees of freedom. If one would like millimetre-accuracy in the design, this would lead to more than 32 million degrees of freedom. The current model performs the eigenvalue optimization with roughly 3 million degrees of freedom in a few days.

A representation of the wafer stage can be found in Fig. 4.1.

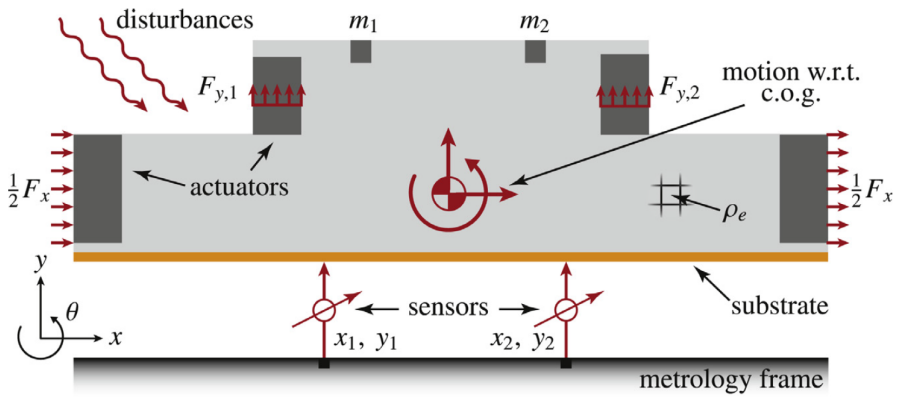


Figure 4.1: A representation of the free-floating wafer stage. The light gray area may be modified using topology optimization. The dark gray areas are the actuators. Taken from Van der Veen et al. [29].

The wafer stage can be controlled in three degrees of freedom (x, y and θ) by the actuator forces $F_x, F_{y,1}$ and $F_{y,2}$ in Fig. 4.1. The contactless sensors measure the displacements in the x- and y-direction and this information is used by the actuators to position the stage.

The stage design setup that we use for optimization contains only the F_x-actuators. An example of an optimized wafer stage is shown in Fig. 4.2.

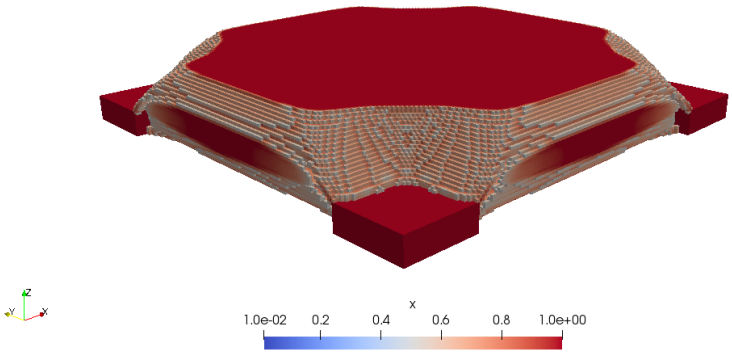


Figure 4.2: An example of a solution to the 3D dynamic wafer stage optimization problem. The four corners are actuators and are fixed at density one. Note that the axis labels are different from Fig. 4.1.

4.1.1 Optimization formulation and dynamical problem

The optimization for this structure is based on eigenvalue maximization and its formulation is given by:

$$\begin{aligned} \min_{\boldsymbol{x}}: \quad & g(\boldsymbol{x}) = \left(\sum_{i=1}^{m} \frac{1}{\lambda_i} \right)^{-1} \\ \text{subject to:} \quad & \frac{V(\boldsymbol{x})}{V_0} = V, \\ & \boldsymbol{K}(\boldsymbol{x})\boldsymbol{z}_i = \lambda_i \boldsymbol{M}(\boldsymbol{x})\boldsymbol{z}_i, \\ & \lambda_i \neq 0, \\ & \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}, \\ & \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n, \end{aligned} \tag{4.1}$$

where $\boldsymbol{M}(\boldsymbol{x}) \in \mathbb{R}^{n \times n}$ and $\boldsymbol{K}(\boldsymbol{x}) \in \mathbb{R}^{n \times n}$ are the mass and stiffness matrices, both SPD, n is the number of degrees of freedom (three times the number of nodes), $\boldsymbol{x}$ is the vector of design variables (i.e. of

element densities), λ_i is the i-th eigenvalue of the eigenvalue problem $\boldsymbol{K}(\boldsymbol{x})\boldsymbol{z}_i = \lambda_i \boldsymbol{M}(\boldsymbol{x})\boldsymbol{z}_i$, $\boldsymbol{z}_i$ is the i-th eigenvector (or eigenmode) of the eigenvalue problem, $V(\boldsymbol{x})$ and V_0 are the material volume and the (constant) design domain volume, respectively, V is a prescribed volume fraction and m is the amount of eigenvalues that will be maximized (often three).

There are no imposed boundary conditions to the eigenvalue problem, since the stage is free-floating. Apart from minimizing $g(\boldsymbol{x})$, the goal is to reach a design that can be constructed. This means e.g. that it has to consist of one piece and be robust. Such conditions are satisfied because of mechanisms within the optimization routine. As before, the initial condition is a design where every element has density of 0.5.

The dynamical problem can be formulated mathematically as [29]:

$$\begin{aligned} \boldsymbol{M}\ddot{\boldsymbol{d}} + \boldsymbol{K}\boldsymbol{d} &= \bar{\boldsymbol{b}}\boldsymbol{u}, \\ \boldsymbol{y} &= \bar{\boldsymbol{c}}^T\boldsymbol{d}, \end{aligned} \tag{4.2}$$

where $\boldsymbol{d}$ is the vector of nodal displacements (previously $\boldsymbol{u}$), $\bar{\boldsymbol{b}}$ is a collection of three normalized actuation vectors corresponding to one unit of actuator force, $\boldsymbol{u}$ contains the inputs (forces), $\bar{\boldsymbol{c}}^T$ is a collection of three normalized sensing vectors and $\boldsymbol{y}$ is a vector of outputs.

The aim is to solve the eigenvalue problem in Eq. 4.1, and then maximize the m smallest eigenvalues.

4.2 The reduced order model

As stated previously, the matrices $\boldsymbol{K}$ and $\boldsymbol{M}$ are very large. A method to effectively reduce the number of unknowns, is by approximating the system with a system with a lower order. This is called model order reduction. Instead of using the normal system, all the matrices and vectors are transformed to smaller versions. The eigenvalue problem can then be solved relatively easily because the number of degrees of freedom is reduced drastically.

The relation between the input and output variables is characterised by the transfer function of the system. We want the transfer function of the reduced system to match the actual transfer function in the best way possible, because then the behaviour of the low-order model approximates the behaviour of the high-order model. This is done using moment matching. It is in the construction of the reduced system that Krylov methods are used.

4.2.1 Transfer function and moments

In order to see how this all works, we first need to write Eq. 4.2 as a first order system. Eq. 4.2 can be written as:

$$\begin{aligned} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{d}} \\ \ddot{\boldsymbol{d}} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{K} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{d} \\ \dot{\boldsymbol{d}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ \bar{\boldsymbol{b}} \end{bmatrix} \boldsymbol{u}, \\ \boldsymbol{y} &= \begin{bmatrix} \bar{\boldsymbol{c}}^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{d} \\ \dot{\boldsymbol{d}} \end{bmatrix}. \end{aligned} \tag{4.3}$$

This is still a second order system, but now it contains $2n$ unknowns. The system can be written as a first-order system by introducing matrices $\boldsymbol{E}, \boldsymbol{A}$ and vectors $\boldsymbol{x}, \boldsymbol{b}, \boldsymbol{c}$. This is called the state-space formulation [20]:

$$\begin{aligned} \boldsymbol{E}\dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\boldsymbol{u}, \\ \boldsymbol{y} &= \boldsymbol{c}^T\boldsymbol{x}. \end{aligned} \tag{4.4}$$

The transfer function and moments are defined using this formulation. The transfer function $H(s)$ is the dependence between the input and output of the system:

$$H(s) = \boldsymbol{c}^T(s\boldsymbol{E} - \boldsymbol{A})^{-1}\boldsymbol{b}, \tag{4.5}$$

with $s \in \mathbb{C}$ arbitrary in the Laplace domain. The transfer function can be rewritten in a polynomial form expressed in its moments M_i. This is called moment expansion:

$$H(s) = \sum_{i=0}^{\infty} (-1)^i M_i(s_0)(s - s_0)^i, \tag{4.6}$$

with $s_0 \in \mathbb{C}$ arbitrary and $M_i(s_0)$ the i-th moment of the transfer function expanded around s_0 [6]:

$$M_i(s_0) = \boldsymbol{c}^T (s_0 \boldsymbol{E} - \boldsymbol{A})^{-1} \boldsymbol{E})^i (s_0 \boldsymbol{E} - \boldsymbol{A})^{-1} \boldsymbol{b}. \tag{4.7}$$

The goal is to calculate a reduced-order system with transfer function $\hat{H}(s)$:

$$\hat{H}(s) = \sum_{i=0}^{\infty} (-1)^i \hat{M}_i(s_0)(s - s_0)^i, \tag{4.8}$$

where

$$\hat{M}_i(s_0) = M_i(s_0) \text{ for } i = 1, ..., k, \tag{4.9}$$

i.e. the first k moments are matched.

For now, we will consider moment expansion around $s_0 = 0$, meaning Eq. 4.7 reduces to:

$$M_i(0) = \boldsymbol{c}^T (\boldsymbol{A}^{-1} \boldsymbol{E})^i \boldsymbol{A}^{-1} \boldsymbol{b}. \tag{4.10}$$

We can now revert to the original system. The transfer function Eq. 4.5 becomes:

$$\begin{aligned} H(s) &= \begin{bmatrix} \bar{\boldsymbol{c}}^T & \boldsymbol{0} \end{bmatrix} \left(s \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M} \end{bmatrix} - \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{K} & \boldsymbol{0} \end{bmatrix} \right)^{-1} \begin{bmatrix} \boldsymbol{0} \\ \bar{\boldsymbol{b}} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\boldsymbol{c}}^T & \boldsymbol{0} \end{bmatrix} \left(\begin{bmatrix} s\boldsymbol{I} & -\boldsymbol{I} \\ \boldsymbol{K} & s\boldsymbol{M} \end{bmatrix} \right)^{-1} \begin{bmatrix} \boldsymbol{0} \\ \bar{\boldsymbol{b}} \end{bmatrix} \\ &= \bar{\boldsymbol{c}}^T \left(-(s\boldsymbol{I})^{-1}(-\boldsymbol{I})(s\boldsymbol{M} - \boldsymbol{K}(sI)^{-1}(-\boldsymbol{I}))^{-1} \right) \bar{\boldsymbol{b}} \\ &= \bar{\boldsymbol{c}}^T (\boldsymbol{K} + s^2 \boldsymbol{M})^{-1} \bar{\boldsymbol{b}}, \end{aligned} \tag{4.11}$$

where we have used a formula for block matrix inversion from Lu and Shiou [15], which is allowed because $\boldsymbol{K}$ and $\boldsymbol{M}$ are SPD.

The moment around 0, Eq. 4.10 becomes:

$$\begin{aligned} M_i(0) &= \begin{bmatrix} \bar{\boldsymbol{c}}^T & \boldsymbol{0} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{K} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M} \end{bmatrix} \right)^i \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{K} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{0} \\ \bar{\boldsymbol{b}} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\boldsymbol{c}}^T & \boldsymbol{0} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{0} & -\boldsymbol{K}^{-1}\boldsymbol{M} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}^{-1} \right)^i \begin{bmatrix} -\boldsymbol{K}^{-1}\bar{\boldsymbol{b}} \\ \boldsymbol{0} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\boldsymbol{c}}^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{M}^{-1}\boldsymbol{K} & \boldsymbol{0} \end{bmatrix}^i \begin{bmatrix} -\boldsymbol{K}^{-1}\bar{\boldsymbol{b}} \\ \boldsymbol{0} \end{bmatrix}. \end{aligned} \tag{4.12}$$

The first moments are for example:

$$\begin{aligned} M_0(0) &= -\bar{\boldsymbol{c}}^T \boldsymbol{K}^{-1} \bar{\boldsymbol{b}}, \qquad M_1(0) = 0 \\ M_2(0) &= \bar{\boldsymbol{c}}^T \boldsymbol{M}^{-1} \bar{\boldsymbol{b}}, \qquad M_3(0) = 0 \\ M_4(0) &= -\bar{\boldsymbol{c}}^T \boldsymbol{M}^{-1} \boldsymbol{K} \boldsymbol{M}^{-1} \bar{\boldsymbol{b}}. \end{aligned} \tag{4.13}$$

Direct calculation of the moments takes many matrix inversions and products, therefore we compute the moments in a different way. This involves the usage of the input and output Krylov subspaces of the system and a basis for them.

4.2.2 Input and output Krylov subspaces

In order to compute the moments of the reduced system $\hat{M}_i(0)$ we introduce the input and output Krylov subspaces for the system.

For system 4.4 the input and output Krylov subspaces are given by

$$\mathcal{K}_k(\boldsymbol{A}^{-1}\boldsymbol{E}, \boldsymbol{A}^{-1}\boldsymbol{b}) \quad \text{and} \quad \mathcal{K}_k(\boldsymbol{A}^{-T}\boldsymbol{E}^T, \boldsymbol{A}^{-T}\boldsymbol{c}), \tag{4.14}$$

respectively. Now we can write these in terms of the original matrices $\boldsymbol{K}$ and $\boldsymbol{M}$:

$$\begin{aligned} &\mathcal{K}_k(\boldsymbol{A}^{-1}\boldsymbol{E}, \boldsymbol{A}^{-1}\boldsymbol{b}) \\ &= \mathcal{K}_k\left(\begin{bmatrix} \boldsymbol{0} & -\boldsymbol{K}^{-1}\boldsymbol{M} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{0} & -\boldsymbol{K}^{-1} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{0} \\ \bar{\boldsymbol{b}} \end{bmatrix}\right) \\ &= \mathcal{K}_k\left(\begin{bmatrix} \boldsymbol{0} & -\boldsymbol{K}^{-1}\boldsymbol{M} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}, \begin{bmatrix} -\boldsymbol{K}^{-1}\bar{\boldsymbol{b}} \\ \boldsymbol{0} \end{bmatrix}\right), \end{aligned} \tag{4.15}$$

and

$$\begin{aligned} &\mathcal{K}_k(\boldsymbol{A}^{-T}\boldsymbol{E}^T, \boldsymbol{A}^{-T}\boldsymbol{c}) \\ &= \mathcal{K}_k\left(\begin{bmatrix} \boldsymbol{0} & -\boldsymbol{K}^{-T} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M}^T \end{bmatrix}, \begin{bmatrix} \boldsymbol{0} & -\boldsymbol{K}^{-T} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \bar{\boldsymbol{c}}^T \\ \boldsymbol{0} \end{bmatrix}\right) \\ &= \mathcal{K}_k\left(\begin{bmatrix} \boldsymbol{0} & \boldsymbol{M}^T \\ -\boldsymbol{K}^{-T} & \boldsymbol{0} \end{bmatrix}, \begin{bmatrix} \bar{\boldsymbol{c}}^T \\ \boldsymbol{0} \end{bmatrix}\right). \end{aligned} \tag{4.16}$$

and it can be shown that these reduce to $\mathcal{K}_k(-\boldsymbol{K}^{-1}\boldsymbol{M}, -\boldsymbol{K}^{-1}\bar{\boldsymbol{b}})$ and $\mathcal{K}_k(-\boldsymbol{K}^{-T}\boldsymbol{M}^T, -\boldsymbol{K}^{-T}\bar{\boldsymbol{c}})$ for problem Eq. 4.2 [20].

4.2.3 Reduction matrices

In order to use the in- and output subspaces, we need bases that span them. To this end we introduce the reduction matrices $\boldsymbol{V}$ and $\boldsymbol{W}$. Their columns will be bases for the input and output subspaces, respectively. For now we will assume that $\boldsymbol{V}$ and $\boldsymbol{W}$ are known. The reduction works in the following way. First, the displacements are projected using $\boldsymbol{V}$

$$\boldsymbol{d} = \boldsymbol{V}\boldsymbol{d}_r, \tag{4.17}$$

and the state equation is premultiplied by $\boldsymbol{W}^T$. Here $\boldsymbol{d}_r$ is the reduced version of $\boldsymbol{d}$, i.e. $\boldsymbol{d} \in \mathbb{R}^n, \boldsymbol{d}_r \in \mathbb{R}^k, \boldsymbol{V} \in \mathbb{R}^{n\times k}$ and $\boldsymbol{W} \in \mathbb{R}^{n\times k}$. This yields the following reduced model of order $2k$:

$$\begin{aligned} &\boldsymbol{W}^T\boldsymbol{M}\boldsymbol{V}\ddot{\boldsymbol{d}}_r + \boldsymbol{W}^T\boldsymbol{K}\boldsymbol{V}\boldsymbol{d}_r = \boldsymbol{W}^T\bar{\boldsymbol{b}}\boldsymbol{u}, \\ &\boldsymbol{y} = \bar{\boldsymbol{c}}^T\boldsymbol{V}\boldsymbol{d}_r. \end{aligned} \tag{4.18}$$

If we now define the following reduced matrices:

$$\boldsymbol{M}_r = \boldsymbol{W}^T\boldsymbol{M}\boldsymbol{V}, \quad \boldsymbol{K}_r = \boldsymbol{W}^T\boldsymbol{K}\boldsymbol{V}, \quad \bar{\boldsymbol{b}}_r = \boldsymbol{W}^T\bar{\boldsymbol{b}}, \quad \bar{\boldsymbol{c}}_r^T = \bar{\boldsymbol{c}}^T\boldsymbol{V}, \tag{4.19}$$

we obtain the exact formulation as in Eq. 4.2 but with the reduced matrices.

The Krylov subspaces and the reduction matrices adhere to the following theorem. It is proven by Salimbahrami and Lohmann [20]:

Theorem 4.1. *Let $\boldsymbol{K}$ and $\boldsymbol{K}_r$ be invertible. If the columns of matrices $\boldsymbol{V}$ and $\boldsymbol{W}$ from Eq. 4.18 are bases of the Krylov subspaces $\mathcal{K}_k(-\boldsymbol{K}^{-1}\boldsymbol{M}, -\boldsymbol{K}^{-1}\bar{\boldsymbol{b}})$ and $\mathcal{K}_k(-\boldsymbol{K}^{-T}\boldsymbol{M}, -\boldsymbol{K}^{-T}\bar{\boldsymbol{c}})$, respectively, both with rank k, then the first $2k$ moments of the original and reduced-order system match. [Adapted after Salimbahrami and Lohmann [20]].*

This theorem tells us that we can achieve moment matching by constructing the Krylov subspaces with the reduction matrices. What remains is how to construct the $\boldsymbol{V}$ and $\boldsymbol{W}$. One could use the definition of the Krylov subspaces directly, but this is prone to errors and instability. Instead, the Arnoldi algorithm is used. This is a well-known algorithm that constructs an orthonormal basis for a Krylov subspace using the modified Gram-Schmidt process [3]. We will denote the basis vectors, i.e. the columns of $\boldsymbol{V}$ and $\boldsymbol{W}$, with $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$, respectively.

Alg. 6 describes the construction of matrix $\boldsymbol{V}$. The construction of $\boldsymbol{W}$ is similar.

Algorithm 6 Arnoldi algorithm for constructing reduction matrix $\boldsymbol{V}$.

1: Solve $\boldsymbol{v}_0 = \boldsymbol{K}^{-1}\bar{\boldsymbol{b}}$
2: Normalize $\boldsymbol{v}_0 \coloneqq \boldsymbol{v}_0/\|\boldsymbol{v}_0\|$
3: **for** $j = 1, ..., k-1$ **do**
4: $\quad \boldsymbol{v}_j \coloneqq \boldsymbol{K}^{-1}\boldsymbol{M}\boldsymbol{v}_{j-1}$
5: $\quad$ **for** $i = 1, ..., i$ **do**
6: $\quad\quad \alpha \coloneqq (\boldsymbol{v}_j, \boldsymbol{v}_i)$
7: $\quad\quad \boldsymbol{v}_j \coloneqq \boldsymbol{v}_j - \alpha\boldsymbol{v}_i$
8: $\quad$ **end for**
9: $\quad \boldsymbol{v}_j \coloneqq \boldsymbol{v}_j/\|\boldsymbol{v}_j\|$
10: **end for**

The solution of the k matrix equations in line 1 and 4 can be accelerated with deflation. Before we do this, we will make an adaptation.

4.3 Spectral Transformation

Recall that in the derivation above the approximation of the transfer function was expanded around $s_0 = 0$. However, in order to find the first m eigenvalues that have to be optimized in Eq. 4.1, the approximate transfer function $\hat{H}(s)$ has to be as accurate as possible near these eigenvalues, not near 0. So we should expand around the three optimized eigenvalues from the previous optimization iteration, since this is the closest guess one can get to the current eigenvalues. For this, the following theorem, proved by Grimme [8], is very useful:

Theorem 4.2. *Let $\boldsymbol{E}_r, \boldsymbol{A}_r, \boldsymbol{b}_r$ and $\boldsymbol{c}_r$ be the reduced order parameters of the state-space formulation Eq. 4.4. Let $\sigma^2 \in \mathbb{R}$ and let $\boldsymbol{A} - \sigma^2\boldsymbol{E}$ and its reduced counterpart be invertible.*
If the columns of matrices $\boldsymbol{V}$ and $\boldsymbol{W}$ from Eq. 4.18 are bases of the Krylov subspaces $\mathcal{K}_k\left((\boldsymbol{A}-\sigma^2\boldsymbol{E})^{-1}\boldsymbol{E}, (\boldsymbol{A}-\sigma^2\boldsymbol{E})^{-1}\boldsymbol{b}\right)$ and $\mathcal{K}_k\left((\boldsymbol{A}-\sigma^2\boldsymbol{E})^{-T}\boldsymbol{E}^T, (\boldsymbol{A}-\sigma^2\boldsymbol{E})^{-T}\boldsymbol{c}\right)$, respectively, then the moments of Eq. 4.4 satisfy

$$-\boldsymbol{c}^T\left((\boldsymbol{A}-\sigma^2\boldsymbol{E})^{-1}\boldsymbol{E}\right)^i(\boldsymbol{A}-\sigma^2\boldsymbol{E})^{-1}\boldsymbol{b} = -\boldsymbol{c}_r^T\left((\boldsymbol{A}_r-\sigma^2\boldsymbol{E}_r)^{-1}\boldsymbol{E}_r\right)^i(\boldsymbol{A}_r-\sigma^2\boldsymbol{E}_r)^{-1}\boldsymbol{b}_r \tag{4.20}$$

for $i = 1, ..., 2k$. In other words, the first $2k$ moments of the original and reduced-order system, expanded around $s_0 = \sigma^2$, match. [Adapted after Grimme [8].]

This theorem shows that by shifting the matrix $\boldsymbol{A}$ with the matrix $\boldsymbol{E}$ multiplied by a factor σ^2, we obtain moment matching around the same σ^2. By transforming to the second-order model as in Eq. 4.14 to 4.16 a similar statement to Thm. 4.1 can be made:

Corollary 4.3. *Let $\boldsymbol{K} - \sigma^2\boldsymbol{M}$ and its reduced counterpart be invertible. If the columns of matrices $\boldsymbol{V}$ and $\boldsymbol{W}$ from Eq. 4.18 are bases of the Krylov subspaces $\mathcal{K}_k((\boldsymbol{K}-\sigma^2\boldsymbol{M})^{-1}\boldsymbol{M}, (\boldsymbol{K}-\sigma^2\boldsymbol{M})^{-1}\bar{\boldsymbol{b}})$ and $\mathcal{K}_k((\boldsymbol{K}-\sigma^2\boldsymbol{M})^{-T}\boldsymbol{M}, (\boldsymbol{K}-\sigma^2\boldsymbol{M})^{-T}\bar{\boldsymbol{c}})$, respectively, both with rank k, then the first $2k$ moments around $s_0 = \sigma^2$ of the original and reduced-order system match.*

In addition to Thm. 4.2 Grimme [8] also proved a theorem using several values for σ^2, which can be summarized as: given $\sigma_1^2, ..., \sigma_l^2 \in \mathbb{R}$, if the columns of $\boldsymbol{V}$ and $\boldsymbol{W}$ are bases of the collection of the l input and output subspaces calculated with the different values for σ^2, respectively, the moments match around each value of σ^2.

This means that by setting σ_i^2 equal to the optimized eigenvalues of the previous optimization iteration, the corresponding Krylov subspaces as in Cor. 4.3 can be used to calculate the reduced order model

with a transfer function that matches the original transfer function perfectly in the optimized eigenvalues from the previous optimization iteration, and therefore nearly perfectly in the optimized eigenvalues of the current optimization iteration.

4.3.1 Eigenvalue shift

Recall that the optimization formulation Eq. 4.1 contained the following eigenvalue problem:

$$\boldsymbol{K}\boldsymbol{z}_i = \lambda_i \boldsymbol{M}\boldsymbol{z}_i, \tag{4.21}$$

Now we can introduce the shifting parameter σ^2 used in Thm. 4.2, and shift Eq. 4.21 with σ^2, i.e. subtract $\sigma^2\boldsymbol{M}$:

$$(\boldsymbol{K} - \sigma^2\boldsymbol{M})\boldsymbol{z}_i = (\lambda_i - \sigma^2)\boldsymbol{M}\boldsymbol{z}_i. \tag{4.22}$$

Since $\boldsymbol{M}$ is non-singular, the pencil $((\boldsymbol{K} - \sigma^2\boldsymbol{M}), \boldsymbol{M})$ is equivalent to the pencil $((\boldsymbol{K} - \sigma\boldsymbol{M})\boldsymbol{M}^{-1}, \boldsymbol{I})$, so this may be written as:

$$(\boldsymbol{K} - \sigma^2\boldsymbol{M})\boldsymbol{M}^{-1}\boldsymbol{z}_i = (\lambda_i - \sigma^2)\boldsymbol{z}_i. \tag{4.23}$$

This eigenvalue problem is equivalent to the original Eq. 4.21, but its eigenvalues are shifted by σ^2. In a 3D situation, the first six eigenvectors, $\boldsymbol{z}_1$ to $\boldsymbol{z}_6$ are the rigid body modes, and the first six eigenvalues are therefore close to zero. The next smallest eigenvalues are the ones that the optimization problem seeks to maximize.

In order to approximate the transfer function correctly, the parameter σ^2 is used as explained in the theorems. We take three different values for σ^2: σ_1^2, σ_2^2 and σ_3^2. They are equal to the first three smallest eigenvalues after the rigid body modes. Because the eigenvalues are not known when solving the equation, the three smallest eigenvalues after the RBM of the previous iteration are taken. This means $\lambda_i - \sigma^2$ will be close to 0, but not equal to it, since the eigenvalues change in every iteration.

Because of this shift, the rigid body modes now correspond to eigenvalues close to $-\sigma^2$ instead of close to zero. This means that the matrix $\boldsymbol{K} - \sigma^2\boldsymbol{M}$ can become indefinite. In that case, the Conjugate Gradient method is not usable. We will consider several other methods, which will be introduced shortly. We will later see that the rigid body modes can be deflated, so that theoretically no eigenvalues are below zero, meaning that the matrix would become SPD again. In that case, CG can be used again.

It can occur that two of the three eigenvalues are the same. If this happens, two of the σ_i^2's will be the same. That means that two identical sets of basis vectors will be added to the reduction matrices $\boldsymbol{V}$ and $\boldsymbol{W}$. In order to still get three different sets of vectors, the σ_i^2 are not calculated as exactly equal to the eigenvalue of the previous optimization iteration. Instead, the following is used:

$$\sigma_i^2 = (1 - \varepsilon)\lambda_{6+i} + \varepsilon\hat{\sigma}_i^2, \tag{4.24}$$

with $\hat{\sigma}_i^2$ the i-th sigma from the previous iteration, λ_{6+i} the eigenvalue of the previous optimization iteration and ε a small value, e.g. 0.01. In this way, we have for example that $\sigma_1^2 \neq \sigma_2^2$ even if $\lambda_7 = \lambda_8$.

4.3.2 Constructing the shifted subspaces

When constructing the subspaces using the shifted matrices, the calculations in the Arnoldi algorithm change accordingly. Alg. 6 now becomes:

Algorithm 7 Arnoldi algorithm for constructing reduction matrix $\boldsymbol{V}$, using shift with σ^2.

Solve $\boldsymbol{v}_0 = (\boldsymbol{K} - \sigma^2\boldsymbol{M})^{-1}\bar{\boldsymbol{b}}$
Normalize $\boldsymbol{v}_0 \coloneqq \boldsymbol{v}_0/\|\boldsymbol{v}_0\|$
for $j = 1, ..., k-1$ **do**
    $\boldsymbol{v}_j \coloneqq (\boldsymbol{K} - \sigma^2\boldsymbol{M})^{-1}\boldsymbol{M}\boldsymbol{v}_{j-1}$
    for $i = 1, ..., i$ **do**
        $\alpha \coloneqq (\boldsymbol{v}_j, \boldsymbol{v}_i)$
        $\boldsymbol{v}_j \coloneqq \boldsymbol{v}_j - \alpha\boldsymbol{v}_i$
    end for
    $\boldsymbol{v}_j \coloneqq \boldsymbol{v}_j/\|\boldsymbol{v}_j\|$
end for

Again the construction of $\boldsymbol{W}$ is similar. As stated before, we will apply deflation to the equations in line 1 and 4. In total, $2k$ matrix equations are solved. When solving an equation as in line 4, one first computes a vector, say $\boldsymbol{x}$, with the value $\boldsymbol{x} = \boldsymbol{M}\boldsymbol{v}_{j-1}$, and then solves the matrix equation $\boldsymbol{v}_j = (\boldsymbol{K} - \sigma^2\boldsymbol{M})^{-1}\boldsymbol{x}$.

4.4 Possible methods

As mentioned before we have to turn to other Krylov subspace methods that are suitable for indefinite problems. There are several methods with their own advantages and disadvantages and we will try a few of them.

4.4.1 MINRES

MINRES (Minimal Residual method) was developed by Paige and Saunders [17]. It is meant for symmetric indefinite linear systems. An advantage is that it is optimal in its class, it minimizes the residual norm (hence the name). A disadvantage is that it theoretically requires an SPD preconditioner. If the system matrix is not positive definite, the diagonal is not necessarily positive, i.e. a Jacobi preconditioner matrix might also be indefinite. We expect therefore that the combination of MINRES with Jacobi will perform poorly, so this combination will not be used.

4.4.2 GMRES

GMRES (Generalized Minimal Residual method) is the most-used method for solving indefinite problems. GMRES was developed by Saad and Schultz [19]. It is a generalization of MINRES and can be used on nonsymmetric systems of linear equations, so it is very general.

The method uses Arnoldi's method to find a vector in the Krylov subspace that approximates the solution. A downside of GMRES is that the cost of every next iteration is larger than the previous, because information from all iterations is necessary. A solution for this is restarting the method with the obtained solution as initial guess. This can sometimes lead to stagnation or instability. An advantage of GMRES is that if it does not stagnate or become unstable, it is always the method with the smallest number of iterations.

4.4.3 BiCGSTAB

BiCGSTAB (Biconjugate Gradient Stabilized method) was developed by Van der Vorst [30]. It is used for nonsymmetric linear systems and is a more stable variant of the Conjugate Gradients-Squared (CGS) method, which is a variant of the Bi-Conjugate Gradients (Bi-CG) method [25]. An advantage of BiCGSTAB is that it uses short recurrences, and therefore a limited, fixed amount of memory. The method therefore does not require restarts, as opposed to GMRES. Another advantage is that it does not require an SPD preconditioner. A disadvantage however, is that the convergence can be quite erratic, i.e. the residual norm jumps up and down. There also is no optimality property for general matrices.

4.4.4 IDR(s)

IDR(s) (Induced Dimension Reduction with parameter s) was developed by Sonneveld and Van Gijzen [26]. It is used for nonsymmetric systems. It is not based on Bi-CG, and uses short recurrence formulas, which reduces the computational time and required memory. For $s = 1$ it produces the same residuals as BiCGSTAB, but for $s > 1$ it is always at least as good or faster than BiCGSTAB [26], [4]. We will use $s = 4$.

4.4.5 Preconditioners

As mentioned before, the Jacobi preconditioner is a standard preconditioner, that can be used to test other preconditioners against. The preconditioner that is currently used in the Topology Optimization project is GAMG (Geometric Algebraic Multigrid Preconditioner). Another option is GASM (Generalized Additive Schwarz Method). Both of these preconditioners are not necessarily SPD. However, we will see shortly that they still perform well with MINRES to a certain extent. We will use and test these preconditioners, as well as Jacobi, in the simulations.

4.5 Possible deflation techniques

As with the static problem, the standard way of applying deflation is using the eigenvectors. In this case, we use the three eigenvectors that correspond to the eigenvalues that are used in the model order reduction, i.e. the eigenvectors that correspond to the smallest eigenvalues after the rigid body modes, but from the previous optimization iteration. This is advantageous because no extra computational costs have to be made to calculate the deflation vectors. Because of the shift with σ^2, the small eigenvalues that these eigenvectors belong with, have become very small, and it is therefore likely an improvement on the condition number of the matrix if they are deflated. However, bear in mind that the next smallest eigenvalues have also been shifted by σ^2, so after shifting and deflating the now smallest eigenvalue can still be relatively close to zero.

Another possibility is to use the rigid body modes that we have used in the static topology optimization. This time they are the real rigid body modes of the free-floating stage. There are now three translation vectors and three rotation vectors, so six deflation vectors. As seen before, after shifting with σ^2, the rigid body modes correspond to eigenvalues close to $-\sigma^2$. They are the only negative eigenvalues in most cases, so deflating them will accelerate the solving of matrix equations with $\boldsymbol{K} - \sigma^2\boldsymbol{M}$ and also make that matrix SPD again, allowing for the use of the Conjugate Gradient method.

A large advantage is that the rigid body modes are already available. This is because the current optimization code uses the GAMG preconditioner. The rigid body modes are calculated to be added to the bases $\boldsymbol{V}$ and $\boldsymbol{W}$ before any other vectors are added. They are also set as the near-null space of the Krylov matrix. The GAMG preconditioner in PETSc requires this to work correctly.

Even better is to use the rigid body modes and the eigenvectors together as deflation vectors. This combines the advantages of both cases that were just discussed. Fig. 4.3 illustrates what happens to the eigenvalues when deflating with rigid body modes or eigenvectors or both.

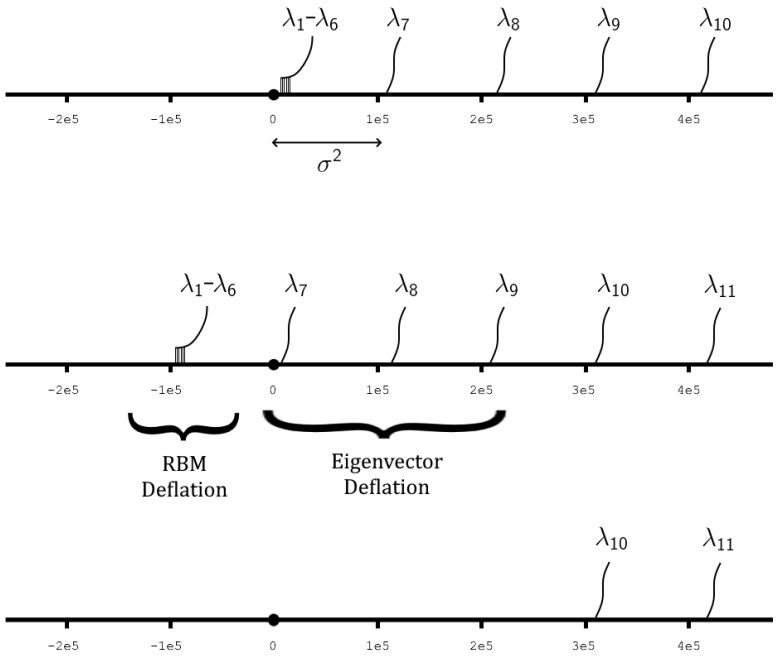


Figure 4.3: An illustration of the influence of deflation with rigid body modes and/or eigenvectors to an example of a spectrum. $\lambda_1 - \lambda_6$ are the eigenvalues corresponding to the rigid body modes. $\lambda_7 - \lambda_9$ are the three smallest eigenvalues after these. Top: The eigenvalues of $\boldsymbol{K}$. Middle: The eigenvalues of $\boldsymbol{K} - \sigma^2\boldsymbol{M}$. Bottom: The eigenvalues of the deflated matrix $\boldsymbol{P}(\boldsymbol{K} - \sigma^2\boldsymbol{M})$, deflated with RBM and eigenvectors.

For the static problem we observed that the largest decrease in the number of Krylov iterations was produced by using rigid body modes deflation on square regions. This can be done in the dynamic problem as well, but now with cubic regions. The eigenvectors are also added, because they are better when combined, as argued above. An important parameter is the size of the cubes. This decides the total number of cubes and therefore the number of deflation vectors. More deflation vectors means more matrix-vector products in the creation of the deflation subspace matrix but it also means the grid is divided into more cubes. This yields a larger deflation space, which is expected to reduce the number of iterations. There is a balance where the cube size is optimal. The optimal cube size will likely depend on the grid size, Krylov method and preconditioner, and we will attempt to find it.

If the number of nodes in one direction is not divisible by the cube size, then the last cube will have less nodes. Some cubes are therefore rectangular cuboids, but we will refer to them as cubes. If there is only one node in the last cube, it is added to the second-to-last. Recall from the 2D case that if one node is the only one in a region, the corresponding deflation vectors are linearly dependent, leading to a singular Galerkin matrix $\boldsymbol{E}$. In 3D, this would occur if in all three directions, the last node would be put alone in a last cube. This becomes impossible if we do not let the last node to be the only one in a cube.

For the first optimization iteration, there will not be any previous eigenvectors. This means that for the first optimization iteration, we can only use a solve without deflation or one with only the rigid body modes. As of the next iteration, any of the just described methods can be used.

Chapter 5

Implementation and results of the dynamical problem

In the previous chapter the theory behind the dynamical problem, the use of model order reduction, and the way deflation might be applied to the problem were explained. Now it is time to implement these ideas and see if and how they improve on the current situation.

For the implementation of the wafer stage problem PETSc is used. This is a library for C and Fortran. The Topology Optimization project from the TU Delft currently uses C code with PETSc for its calculations, as well as Aage et al. [1], for example. The Krylov methods and preconditioners introduced in the previous chapter are all available in PETSc, except for IDR, which was separately implemented for the purpose of the Topology Optimization project.

5.1 Implementation

The already available PETSc code from the Topology Optimization project solves the problem using model order reduction and moment matching, without using deflation. We would like to apply deflation to this problem, so that the computation time can be reduced.

Before we start on the full problem, some intermediate steps must be taken. For example, running the optimization of a 3D problem takes a very long computation time. This is because there are many more elements and nodes, and their amount now scales with the third order of the grid size. This means that in the testing phase, it is preferred to first work with small versions of the problem. Using small grid sizes we can investigate all the different methods and deflation techniques.

Furthermore, instead of solving all matrix equations in Alg. 7 and its counterpart that constructs $\boldsymbol{W}$, we start by solving only the matrix equation in line 1. This equation is solved once in every optimization iteration, next to the normal solve with model order reduction (i.e. the solutions are not used for further calculations).

After this, the grid sizes will be increased more and more and eventually the other matrix equations in the algorithm will also be solved using deflation.

In order to get this to work in PETSc, a matrix shell variable (MatShell) for $\boldsymbol{P}(\boldsymbol{K} - \sigma^2\boldsymbol{M})$ was created from the matrices $\boldsymbol{K}$ and $\boldsymbol{M}$, using the deflation subspace matrix $\boldsymbol{Z}$. The matrix $\boldsymbol{P}(\boldsymbol{K} - \sigma^2\boldsymbol{M})$ is used in the KSP (Krylov subspace) solver as the Krylov matrix. The matrix is never actually stored by the program, but is calculated when needed and then used. This is advantageous because $\boldsymbol{P}(\boldsymbol{K} - \sigma^2\boldsymbol{M})$ is not sparse, so storing it takes a lot of memory.

The matrix that is the source for the preconditioner is chosen to be just $(\boldsymbol{K} - \sigma^2\boldsymbol{M})$. Then the method and preconditioner can be selected in PETSc and they will work as a Preconditioned Deflated method, as required. Furthermore, functions are implemented to premultiply with $\boldsymbol{P}, \boldsymbol{Q}$ and $\boldsymbol{P}^T$, because we need $\boldsymbol{Pb}$ (Eq. 2.4) and $\boldsymbol{Qb}$ and $\boldsymbol{P}^T\hat{\boldsymbol{x}}$ for projecting the solution back as in Eq. 2.5.

Recall from Chapter 2 that the Galerkin matrix $\boldsymbol{E}$ has to be inverted when using deflation. Because it is small, this can be done using LU factorization.

Using deflation introduces some extra computational costs. They will be amply compensated by the reduction in Krylov iterations. We will provide some insight in these costs, and provide the number of flops (floating point operations) that are used for a certain computation. Functions that take only few flops are not counted (it is a qualitative estimate): inproducts, matrix-vector products, LU-factorizations and summations of multiple vectors are counted. Let n be the size of matrix $(\boldsymbol{K} - \sigma^2\boldsymbol{M})$, m be the number of deflation vectors and l be the number of Krylov iterations required for a certain optimization iteration.

There are calculations that are made only once for an optimization iteration: creating the deflation subspace matrix $\boldsymbol{Z}$, calculating $(m(2n^2-n))$ and storing the product $(\boldsymbol{K}-\sigma^2\boldsymbol{M})\boldsymbol{Z}$, creating $(m(2n-2))$ and factoring $(\frac{2}{3}m^3)$ the Galerkin matrix $\boldsymbol{E}$, and the deflation $(2n^2 + n(3m-1) + m(m-2))$ and back-projection $(2n^2+n(6m-1)+m(2m-4))$ of the right hand side vector. Other calculations are done for each Krylov iteration, namely the calculation of the product with $\boldsymbol{P}(\boldsymbol{K}-\sigma^2\boldsymbol{M})$: $l(2n^2+n(3m-1)+m(m-2))$. Note that these numbers are upper bounds because they don't assume any sparsity in the vectors and matrices. The total number of flops due to functions summed up above is then:

$$l(2n^2 + n(3m-1) + m(m-2)) + n^2(2m+4) + \frac{2}{3}m^3 + n(10m-2) + 3m^2 - 8m \tag{5.1}$$

The most time-consuming operations are the calculations of the product with $\boldsymbol{P}(\boldsymbol{K}-\sigma^2\boldsymbol{M})$, calculating $(\boldsymbol{K}-\sigma^2\boldsymbol{M})\boldsymbol{Z}$ and when m becomes large, factoring the Galerking matrix $\boldsymbol{E}$.

Performing the full eigenvalue optimization can only be done for small grid sizes, because it takes a lot of time. In order to test larger sizes, the eigenvalue optimization is done on the cluster, and the eigenvectors and the design are then saved to files. These are then imported again by the code that is run on the laptop. There the matrices $\boldsymbol{K}$ and $\boldsymbol{M}$ are constructed using the imported variables, and then line 1 of Alg. 7 is solved for every optimization iteration.

The final step is to choose the Krylov method and preconditioner that are currently used in the project and implement the deflation techniques in the actual computation of the reduction bases $\boldsymbol{V}$ and $\boldsymbol{W}$. Then the full eigenvalue optimization with model order reduction can be tested on the cluster using deflation on the large grid sizes that are currently the state-of-the-art, and these results can be compared to the non-deflated results.

The advantage of upscaling in this way is that many options can be tried on the small grid sizes and choices can be made for larger sizes based on the results from smaller sizes. In general, the result of upscaling on the performance of the combinations of methods, preconditioners and deflation types can be investigated.

The grid sizes used for solving line 1 of Alg. 7 next to the eigenvalue optimization will be $12 \times 12 \times 2$ and $30 \times 30 \times 5$. The grid size for solving line 1 of Alg. 7 while importing the eigenvectors and design will be $60 \times 60 \times 10$. The grid sizes for solving all matrix equations in Alg. 7 will be $120 \times 120 \times 20$ and $180 \times 180 \times 30$.

5.2 Introduction to the results

In this chapter the results for the different grid sizes using the different implementations are presented. In the tables below the number of Krylov iterations and the needed time per optimization iteration are shown for solving line 1 of Alg. 7. These are compared between using only preconditioning and using both the deflation techniques and preconditioning. This is done using the earlier mentioned 3D grid sizes and using the different Krylov subspace methods and preconditioners described in Sec. 4.4.

In all cases, 10 optimization iterations are performed, and the average number of Krylov iterations required is shown in the tables. If an entry shows 'DNC' it means the method did not converge within 10000 iterations to the required relative tolerance of 10^{-8}. Note: this does not mean that the method would not converge after this, it is just a way of truncation. In order to correctly compare BiCGSTAB to the other Krylov methods the number of iterations should be doubled because one BiCGSTAB iteration actually consists of two iterations. This is already done in the tables.

Note that the timing results can be very sensitive to other processes happening on a laptop, especially for the smaller grid sizes. They are meant as an indication for the improvement that deflation yields, rather than an exact timing measurement; for any combination of Krylov method and preconditioner the results were calculated directly after each other by table row.

The timing results do not include the construction of the deflation subspace matrix $\boldsymbol{Z}$, which consistently turned out to take a nonsignificant fraction of time. They do include the calculation of the shell matrix $\boldsymbol{P}(\boldsymbol{K} - \sigma^2\boldsymbol{M})$, which is used in every Krylov iteration.

The Krylov methods that have been described before are all used. The Conjugate Gradient method is also used, but it can only be used when it is deflated using rigid body modes, because then the negative eigenvalues are deflated. The first two columns of CG are therefore empty in the result tables.

5.3 Applying deflation separate from eigenvalue optimization

In order to test the functionality of the implementation of the functions related to deflation, they were used to solve line 1 of Alg. 7:

$$\boldsymbol{v}_0 = (\boldsymbol{K} - \sigma^2\boldsymbol{M})^{-1}\boldsymbol{b}. \tag{5.2}$$

The rest of the implementation that optimizes the eigenvalue problem was left as it is, and the rigid body modes and eigenvectors needed for the deflation were taken from the current and previous optimization iteration, respectively. Note that because this matrix solve is performed next to the actual eigenvalue optimization, the solution $\boldsymbol{v}_0$ is not used. Table 5.1 and Table 5.2 show the average number of iterations and time needed for the grid sizes $12 \times 12 \times 2$ and $30 \times 30 \times 5$, respectively.

Table 5.1: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $12 \times 12 \times 2$, $n = 1521$, with varying Krylov methods, preconditioners and deflation types. The time is in between parentheses and expressed in ms. The size of the cubes is 6.

Method	Prec.	Deflation type				
		None	Eigenvector	RBM	Both	Both, in cubes
GMRES	Jacobi	DNC (2250)	372 (247)	839 (554)	103 (97)	82 (90)
	GASM	194 (183)	64 (108)	40 (89)	20 (73)	17 (73)
	GAMG	24 (196)	17 (224)	18 (236)	11 (206)	9 (199)
MINRES	GASM	59 (105)	46 (121)	34 (106)	21 (91)	19 (90)
	GAMG	115 (368)	103 (458)	18 (162)	11 (143)	9 (136)
BiCGSTAB	Jacobi	1144 (260)	580 (223)	270 (124)	116 (74)	114 (76)
	GASM	214 (113)	134 (109)	48 (60)	26 (53)	24 (54)
	GAMG	56 (193)	30 (182)	28 (181)	14 (146)	10 (139)
IDR(4)	Jacobi	284 (145)	209 (158)	194 (158)	127 (116)	104 (105)
	GASM	74 (78)	57 (98)	41 (82)	27 (72)	23 (70)
	GAMG	30 (151)	22 (166)	22 (178)	14 (149)	12 (152)
CG	Jacobi			149 (100)	88 (71)	78 (72)
	GASM	Not	Not	33 (59)	20 (53)	18 (50)
	GAMG	Applicable	Applicable	18 (159)	11 (137)	9 (138)

Table 5.2: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $30 \times 30 \times 5$, $n = 17298$, with varying Krylov methods, preconditioners and deflation types. The time is in between parentheses and expressed in ms. The size of the cubes is 6.

Method	Prec.	Deflation type				
		None	Eigenvector	RBM	Both	Both, in cubes
GMRES	Jacobi	DNC (40759)	DNC (51535)	DNC (41872)	829 (5067)	145 (1246)
	GASM	3846 (31126)	514 (4969)	711 (7844)	111 (1500)	36 (866)
	GAMG	161 (6413)	54 (3394)	68 (4032)	34 (2841)	18 (2106)
MINRES	GASM	203 (1472)	171 (1549)	130 (1310)	90 (1123)	35 (743)
	GAMG	DNC (160675)	DNC (79325)	45 (3035)	32 (2691)	15 (2093)
BiCGSTAB	Jacobi	3984 (10715)	10742 (36605)	886 (3387)	564 (2553)	170 (1091)
	GASM	3496 (19926)	976 (6678)	206 (1791)	144 (1478)	44 (822)
	GAMG	132 (6686)	90 (5699)	62 (4610)	44 (3816)	18 (2512)
IDR(4)	Jacobi	1191 (4662)	852 (4185)	761 (4057)	483 (2841)	152 (1213)
	GASM	258 (2109)	207 (2058)	158 (1728)	105 (1355)	41 (830)
	GAMG	66 (3986)	50 (3800)	53 (4084)	38 (3387)	18 (2398)
CG	Jacobi			567 (2792)	396 (2181)	131 (1022)
	GASM	Not	Not	132 (1585)	93 (1358)	36 (815)
	GAMG	Applicable	Applicable	46 (3376)	33 (2795)	15 (2090)

The behaviour of the residual norm over time can also give insight in the convergence process. To enhance readability, all residual norm graphs are collected in Appendix B. The graphs corresponding to Tables 5.1 and 5.2 are Fig. B.1 through B.4.

From these results we can conclude the following:

- As expected, Jacobi is the worst preconditioner. GASM appears to be faster than GAMG, but GAMG uses less Krylov iterations. It is known that GAMG takes a lot of prepare time. We expect this effect to get smaller for larger grid sizes, since the larger the grid size, the smaller the influence of this prepare time will become.
- Using deflation accelerates the computation for all combinations of Krylov methods and preconditioners. The speed-up is greatest for the methods that are doing worse in the undeflated case, i.e. GMRES or BiCGSTAB with Jacobi or GASM, or MINRES with GAMG. For the best combinations the speed-up is less spectacular. This is because they were already performing quite good in the undeflated case.
- When comparing eigenvector deflation and rigid body modes deflation, neither of them is always better than the other. Using both is always better than using either. Using both with cubes is in turn better than using both without cubes.
- Deflation smoothens the convergence of especially BiCGSTAB and IDR(4), but also of the other methods. All deflation types yield better convergence than not using deflation. The residuals do not move up and down as much as in the non-deflated case. This effect is largest for the deflation cases with both rigid body modes and eigenvector deflation.

5.4 Importing eigenvectors and design

In the previous section we tested all combinations of Krylov methods, preconditioners and deflation types on small grid sizes. Increasing the grid size takes a lot of extra time when working on one processor, because the time needed by the eigenvalue optimization process increases greatly. So, in order to test larger grid sizes, we can run the actual optimization process on a computing cluster, using multiple processors, and then save the eigenvectors and design for every optimization iteration. These are then used in the code that solves Eq. 5.2. The rigid body modes and matrices $\boldsymbol{K}$ and $\boldsymbol{M}$ can be calculated using these, as well as the values for σ^2, since if $\boldsymbol{z}_i$ is the i-th eigenvector to Eq. 4.21, we have that:

$$\lambda_i = \frac{\boldsymbol{z}_i^T \boldsymbol{K} \boldsymbol{z}_i}{\boldsymbol{z}_i^T \boldsymbol{M} \boldsymbol{z}_i}. \tag{5.3}$$

Using this technique, solving Eq. 5.2 becomes the task that takes most of the time, instead of the eigenvalue optimization. This allows us to solve it using a grid size of $60 \times 60 \times 10$ on a laptop. The Jacobi preconditioner is not used anymore, since we have already seen how its performance relates to the other preconditioners in the previous tables. The same holds for the GAMG preconditioner when combined with MINRES. Here the convergence stalled when using no deflation. In Table 5.3 the iteration and timing results of the current grid size are collected.

Table 5.3: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $60 \times 60 \times 10$, $n = 122793$, with varying Krylov methods, preconditioners and deflation types. The time is in between parentheses and expressed in seconds. The size of the cubes is 6.

Method	Prec.	Deflation type				
		None	Eigenvector	RBM	Both	Both, in cubes
GMRES	GASM	DNC	1519 (148.61)	1628 (189.34)	211 (26.97)	29 (11.65)
	GAMG	258 (112.06)	71 (46.76)	104 (58.31)	44 (37.10)	9 (28.77)
MINRES	GASM	375 (30.76)	306 (29.42)	236 (24.61)	158 (20.21)	29 (11.14)
BiCGSTAB	GASM	DNC	3332 (254.59)	474 (41.85)	234 (24.18)	38 (11.13)
	GAMG	154 (52.41)	98 (40.79)	86 (38.53)	58 (32.61)	10 (20.15)
IDR(4)	GASM	490 (25.45)	366 (22.55)	311 (20.29)	187 (14.54)	35 (8.21)
	GAMG	83 (31.37)	61 (28.75)	65 (30.61)	46 (26.33)	10 (18.15)
CG	GASM	Not	Not	243 (15.73)	166 (12.74)	29 (7.50)
	GAMG	Applicable	Applicable	57 (26.15)	41 (22.78)	9 (16.39)

The residual norm graphs can be found in Fig. B.5 in Appendix B. From the tables and graphs we can conclude the following:

- The tendency that GAMG is slower than GASM but takes less iterations persists. However, the time difference is becoming smaller. For example, consider the rigid body modes deflation types (the three rightmost columns) combined with IDR(4) and CG. While in Table 5.2 we saw that GASM was between two and three times faster than GAMG, in this table we see that this factor has fallen to in between one and two.
- In the residual norm graphs of grid size $12 \times 12 \times 2$ the graphs of RBM + eigenvectors RBM in cubes + eigenvectors are close to each other. When looking at $30 \times 30 \times 5$ and $60 \times 60 \times 10$ this difference becomes greater, and the iteration count of RBM in cubes + eigenvectors decreases dramatically. Since the cube size is six in all cases, the number of cubes are $4, 25$ and 200, respectively. These numbers are apparently in the range where they are very beneficial to the number of iterations and the computing time, while also not introducing so many extra operations that this nullifies the time reduction.
- Again we see that in the two rightmost columns, the results are more similar between the Krylov methods than in the three leftmost columns. The Krylov methods that perform best in all cases are IDR(4) and CG. None of the other three methods perform significantly better than IDR(4) and CG even with the best deflation types. Therefore, we choose at this point to only use the methods IDR(4) and CG in the following.

5.5 Finding the optimal cube size

Up until this point, the size of the cubes in which the deflation vectors are divided has been six nodes. We have seen this in Table 5.1 through 5.3. The size of the cubes determines the number of deflation vectors. If the number of deflation vectors becomes too high, the Galerkin matrix $\boldsymbol{E}$ will become too large. In the creation and factorization of the Galerkin matrix and at the creation of the product $(\boldsymbol{K} - \sigma^2 \boldsymbol{M})\boldsymbol{Z}$ the number of matrix-vector products and inproducts can become too large and outweigh the benefits brought by the division of the rigid body modes. A similar type of balance was found by [7] when using subdomain deflation and increasing the number of subdomains.

In order to find out where the optimal balance lies between these effects, we can solve Eq. 5.2 deflated with RBM in cubes + eigenvectors several times, each with a different cube size. We do this with the

Krylov method-preconditioner combinations of IDR(4), CG and GASM, GAMG. In Table 5.4 and 5.5 the number of iterations and time results can be found for grid sizes $30 \times 30 \times 5$ and $60 \times 60 \times 10$, respectively.

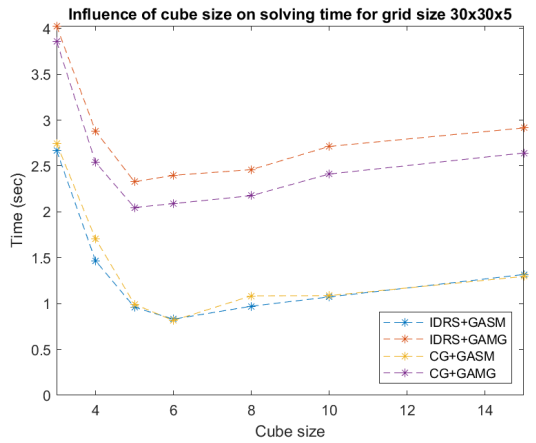
Table 5.4: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $30 \times 30 \times 5$, $n = 17298$, using the methods IDR(4) and CG and the preconditioners GASM and GAMG, deflated with RBM in cubes with varied size + eigenvectors. The time is in between parentheses and expressed in seconds.

Method	Prec.	Cube size						
		3	4	5	6	8	10	15
IDR(4)	GASM	27 (2.67)	34 (1.47)	42 (0.96)	41 (0.83)	48 (0.97)	62 (1.07)	85 (1.32)
	GAMG	9 (4.03)	9 (2.88)	15 (2.33)	18 (2.40)	20 (2.46)	26 (2.71)	33 (2.92)
CG	GASM	24 (2.75)	29 (1.71)	37 (0.99)	36 (0.82)	42 (1.08)	56 (1.09)	76 (1.30)
	GAMG	8 (3.86)	8 (2.54)	14 (2.05)	15 (2.09)	18 (2.18)	23 (2.41)	29 (2.64)

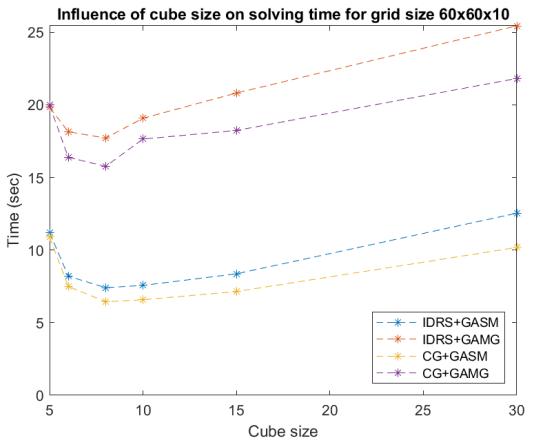
Table 5.5: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $60 \times 60 \times 10$, $n = 122793$, using the methods IDR(4) and CG and the preconditioners GASM and GAMG, deflated with RBM in cubes with varied size + eigenvectors. The time is in between parentheses and expressed in seconds.

Method	Prec.	Cube size						
		4	5	6	8	10	15	30
IDR(4)	GASM	27 (70.59)	33 (11.22)	35 (8.21)	45 (7.41)	60 (7.57)	73 (8.37)	142 (12.54)
	GAMG	10 (89.16)	9 (19.84)	10 (18.15)	14 (17.71)	21 (19.08)	28 (20.82)	42 (25.45)
CG	GASM	23 (86.00)	28 (10.90)	29 (7.50)	38 (6.45)	51 (6.58)	66 (7.14)	124 (10.18)
	GAMG	9 (82.33)	8 (19.99)	9 (16.39)	13 (15.77)	18 (17.65)	25 (18.24)	38 (21.82)

The tables show that the number of iterations is monotonously decreasing when the cube size decreases. The needed time is not monotonous and has a minimum. This is visualized in Figs. 5.1 and 5.2, which show the needed time as plotted against the cube size and the number of cubes, respectively.

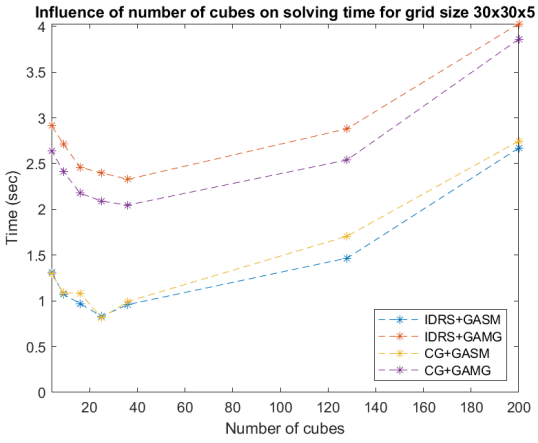


(a) Graph of node cube size against average needed time for solving a matrix equation for grid size $30 \times 30 \times 5$.

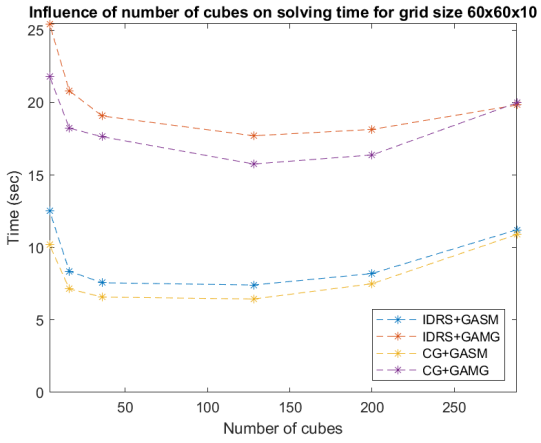


(b) Graph of node cube size against average needed time for solving a matrix equation for grid size $60 \times 60 \times 10$.

Figure 5.1: The influence of size of the node cubes over which the rigid body modes are distributed on the needed time to solve Eq. 5.2 using Krylov methods IDR(4) and CG, preconditioners GASM and GAMG, and grid sizes $30 \times 30 \times 5$ and $60 \times 60 \times 10$.



(a) Graph of total number of node cubes against average needed time for solving a matrix equation for grid size $30 \times 30 \times 5$.



(b) Graph of total number of node cubes against average needed time for solving a matrix equation for grid size $60 \times 60 \times 10$.

Figure 5.2: The influence of number of node cubes over which the rigid body modes are distributed on the needed time to solve Eq. 5.2 using Krylov methods IDR(4) and CG, preconditioners GASM and GAMG, and grid sizes $30 \times 30 \times 5$ and $60 \times 60 \times 10$.

These figures show that the differences between IDR(4) and CG are small, and the differences between the preconditioners GASM and GAMG are larger. As seen before using these grid sizes, GASM is considerably faster.

Another difference is that the optimum for the cube size lies at another value. If we look at grid size $30 \times 30 \times 5$ we see that for GAMG the optimum lies at cube size 5 (36 cubes) and for GASM it lies at cube size 6 (25 cubes). For $60 \times 60 \times 10$ the optimum lies around cube size 8 (128 cubes) for both preconditioners. For $60 \times 60 \times 10$ the location of the optimum is less accurate. It looks like the actual optimum (when interpolating the graphs) is higher for GAMG than for GASM, like for the grid size $30 \times 30 \times 5$.

It appears that when using the GAMG preconditioner, more deflation vectors can be constructed before the overhead becomes apparent in the total computation time. This is likely because GAMG takes more time than GASM.

Although Fig. 5.2b does not allow for an accurate determination of the optimum, it is clear that the number of cubes at optimum is larger for $60 \times 60 \times 10$ than for $30 \times 30 \times 5$. This shows that as the grid size increases, the number of allowed deflation vectors increases as well. In this case a two-fold increase of the dimensions (eight-fold in the number of degrees of freedom) results in around a four- to five-fold increase of the optimal number of cubes.

5.6 Computing using multiple processors

At this point we cannot proceed with single-processor computing. Running a larger example with one processor requires too much memory for a laptop. The code was adapted to work using multiple processors, and to run on a cluster. This allows us to scale up the grid size even more, while still solving only Eq. 5.2. There are two limitations brought by the adaptation:

1. The deflation subspace matrix $\boldsymbol{Z}$ was stored as a sparse matrix in PETSc. This is advantageous, because when using deflation with RBM divided in cubes, the information of one vector is spread over many, leaving many zero elements. However, when using parallel computing, any matrix-vector product has to be calculated in parts, by different processors. The processors only own a part of the vector and matrix. It is currently not possible in PETSc to correctly partition a sparse matrix in such a way that it can be multiplied with a partitioned vector. Therefore $\boldsymbol{Z}$ has to be stored as a group of vectors, which cannot be stored as sparse entities. This will likely reduce the benefits of using the deflation type RBM in cubes + eigenvectors.

2. Even when using a $\boldsymbol{Z}$ stored as a vector, there are problems with the creation of $\boldsymbol{Z}$ in PETSc in the case of RBM in cubes + eigenvectors. The vectors have to be set with three values at a time, because the values of every grid element now belong to different cubes, i.e. different deflation vectors. When using multiple processors, PETSc fails to put the values in the correct index, even when accounting for the fact that indices are only locally owned. The correct $\boldsymbol{Z}$ can therefore not be created for this deflation type. As opposed to this, the $\boldsymbol{Z}$ for the other deflation types just involve copying already computed vectors and can be created without problems. For this reason, the deflation type of RBM in cubes + eigenvectors will not be used when computing results for the larger grid sizes.

In Table 5.6 and Table 5.7 the iteration and timing results are shown for sizes $90\times90\times15$ and $120\times120\times20$, respectively. The residual norm graphs are Fig. B.6 and Fig. B.7. The simulations were run using 10 processors on a computing cluster.

Table 5.6: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $90 \times 90 \times 15$, $n = 397488$, using 10 processors, with varying Krylov methods, preconditioners and deflation types. The time is in between parentheses and expressed in seconds.

Method	Prec.	Deflation type			
		None	Eigenvector	RBM	Both
IDR(4)	GASM	1087 (13.55)	764 (9.90)	546 (7.28)	357 (4.97)
	GAMG	107 (7.77)	78 (6.10)	82 (6.48)	58 (4.97)
CG	GASM	Not	Not	450 (5.83)	306 (4.14)
	GAMG	Applicable	Applicable	71 (5.81)	51 (4.56)

Table 5.7: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $120 \times 120 \times 20$, $n = 922383$, using 10 processors, with varying Krylov methods, preconditioners and deflation types. The time is in between parentheses and expressed in seconds.

Method	Prec.	Deflation type			
		None	Eigenvector	RBM	Both
IDR(4)	GASM	1678 (48.94)	1149 (34.75)	754 (23.53)	451 (14.71)
	GAMG	111 (19.18)	81 (15.24)	87 (16.47)	60 (12.41)
CG	GASM	Not	Not	574 (17.80)	390(12.63)
	GAMG	Applicable	Applicable	77 (15.01)	54 (11.56)

We can conclude the following about these grid sizes:

- The shapes of residual norm graphs of the sizes look very much alike for the sizes $60 \times 60 \times 10, 90 \times 90 \times 15$ and $120 \times 120 \times 20$. The number of necessary iterations increases and the time increases, but with a greater factor than the iterations. The time can of course only be compared between the last two grid sizes, because it is affected by the usage of multiple processors.
- Up until grid size $60 \times 60 \times 10$ we consistly saw that GAMG took more time than GASM but took less iterations. If we consider $90 \times 90 \times 15$ this difference in time has in some cases become smaller and in other cases reversed. If we consider $120 \times 120 \times 20$ GAMG is faster than GASM in all cases. It is known that GAMG takes a lot of time to set up. In the smaller sizes and single-processor cases we have seen that this makes it slower than GASM. With the very small sizes it was even slower than Jacobi. Now that we consider large sizes and multiple processor cases it appears that the set-up time has become less relevant to the total time needed and it is now faster than both other preconditioners.
- In all cases, we see that when the RBM are readily available (as is the case), CG deflated with RBM is faster than IDR(4) deflated with RBM.

5.7 Influence of the number of processors

It is interesting to investigate the influence of the number of processors on the performance of the Krylov methods and preconditioners. In order to test this, the same matrix solve was performed as before, but with a changing number of processors, on the laptop. This was done for grid size $60 \times 60 \times 10$ and $90 \times 90 \times 15$. The latter could not run with one processor on the laptop, but it could when using multiple processors. The results can be found in Tables 5.8 and 5.9.

Table 5.8: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $60 \times 60 \times 10$, $n = 122793$ with varying number of processors, using the methods IDR(4) and CG and the preconditioners GASM and GAMG, deflated with RBM + eigenvectors. The time is in between parentheses and expressed in seconds.

Method	Prec.	Number of processors				
		1	2	4	6	8
IDR(4)	GASM	187 (14.54)	218 (9.71)	229 (8.04)	225 (8.03)	240 (8.75)
	GAMG	46 (26.33)	53 (16.30)	53 (11.91)	53 (12.19)	53 (12.06)
CG	GASM	166 (12.74)	192 (7.59)	199 (6.02)	198 (6.03)	205 (6.50)
	GAMG	41 (22.78)	47 (18.53)	46 (13.47)	46 (12.86)	47 (13.95)

Table 5.9: Table of average number of Krylov iterations and average needed time for solving Eq. 5.2 over 10 optimization iterations for the wafer stage with size $90 \times 90 \times 15$, $n = 397488$ with varying number of processors, using the methods IDR(4) and CG and the preconditioners GASM and GAMG, deflated with RBM + eigenvectors. The time is in between parentheses and expressed in seconds.

Method	Prec.	Number of processors			
		2	4	6	8
IDR(4)	GASM	330 (51.69)	334 (44.90)	334 (31.11)	334 (30.19)
	GAMG	58 (69.89)	56 (50.05)	57 (46.41)	58 (50.68)
CG	GASM	283 (36.86)	292 (26.82)	290 (26.49)	295 (25.95)
	GAMG	51 (53.28)	50 (34.77)	51 (34.77)	50 (32.85)

Ideally the number of processors is inversely proportional with the needed computing time, as observed. Some irregularities appear when using eight processors. This might be explained by the fact that this is the total number of processors on the laptop, so background processes might disturb the performance of the case with eight processors.

All combinations of Krylov methods and preconditioners benefit from the increased number of processors in similar ways. None of the combinations is doing significantly better than the others.

Now that we have solved Eq. 5.2 with many different grid sizes and learned a lot about the different combinations of Krylov methods, preconditioners and deflation types, it is time to apply deflation to the creation of the reduction bases $\boldsymbol{V}$ and $\boldsymbol{W}$. The current solver uses the combination of IDR(4) with GAMG, and no deflation. The results of this are already available. It is therefore best to compare the deflated version of IDR(4) with GAMG to these results. The best deflation method that functions in parallel computing is RBM + eigenvectors. We will use this method as deflation method in the next section.

5.8 Deflation applied to full eigenvalue optimization

The code was adapted to solve all matrix equations in lines 1 and 4 of Alg. 7 (and its counterpart that constructs $\boldsymbol{W}$) with deflation instead of only line 1. They were solved using IDR(4) with GAMG, and the results were compared for using deflation and not using it. As before the tolerance for the Krylov method is a relative tolerance of 10^{-6}. This was done for grid size $120 \times 120 \times 20$ ($n = 922383$) and $180 \times 180 \times 30$ ($n = 3046773$), on a computing cluster using 20 cores.

The time taken per optimization iterations is shown in Fig. 5.3. This includes solving the $2k$ matrix equations and all other operations needed to carry out the optimization iteration.

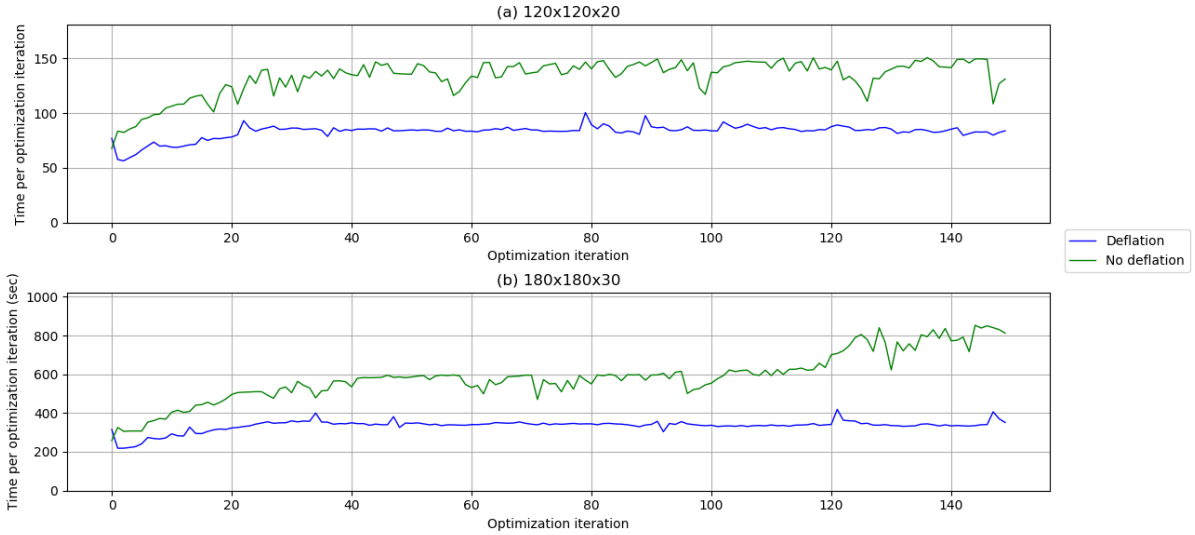


Figure 5.3: The time taken in seconds carrying out a full optimization iteration using IDR(4) and GAMG, on grid sizes $120 \times 120 \times 20$ (a) and $180 \times 180 \times 30$ (b), for 150 optimization iterations.

With grid size $120 \times 120 \times 20$, an optimization iteration took on average 133 seconds in the undeflated case and 83 seconds in the deflated case. This is a reduction of 38%, or by a factor of 1.60. With grid size $180 \times 180 \times 30$, an optimization iteration took on average 586 seconds in the undeflated case and 334 seconds in the deflated case. This is a reduction of 43%, or by a factor of 1.75. Fig. 5.3 also shows that in the deflated case the needed time is more constant over the optimization iterations.

We can also compare the convergence of the eigenvalues that are to be optimized, as well as the objective function $g(\boldsymbol{x})$, see 4.1, which is a function of these eigenvalues. Fig. 5.4 and 5.5 show this convergence.

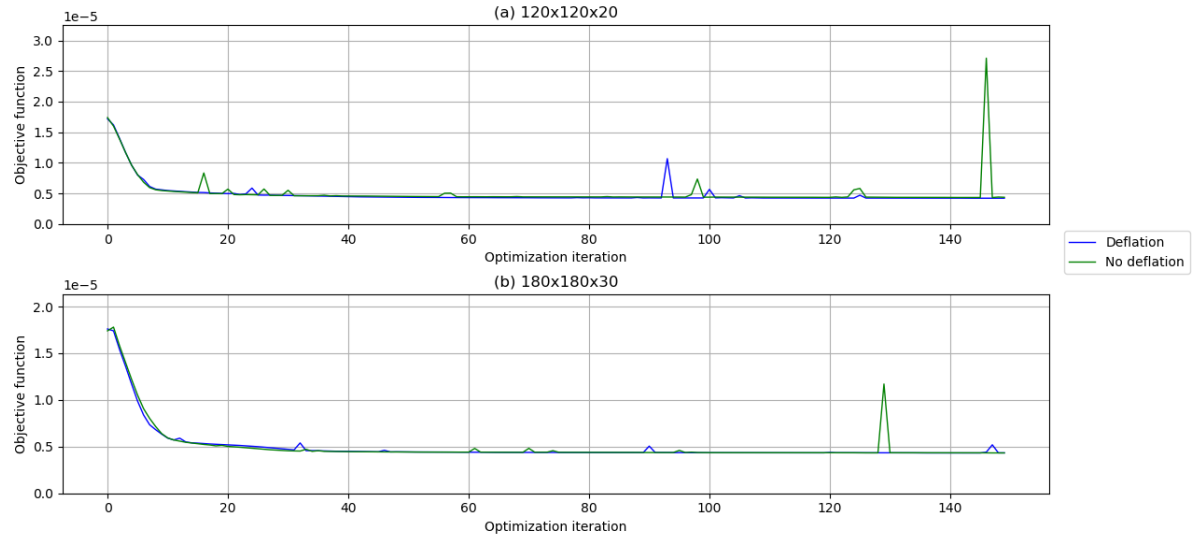


Figure 5.4: The convergence of the objective function for eigenvalue optimization iteration using IDR(4) and GAMG, on grid sizes $120 \times 120 \times 20$ (a) and $180 \times 180 \times 30$ (b), for 150 optimization iterations.

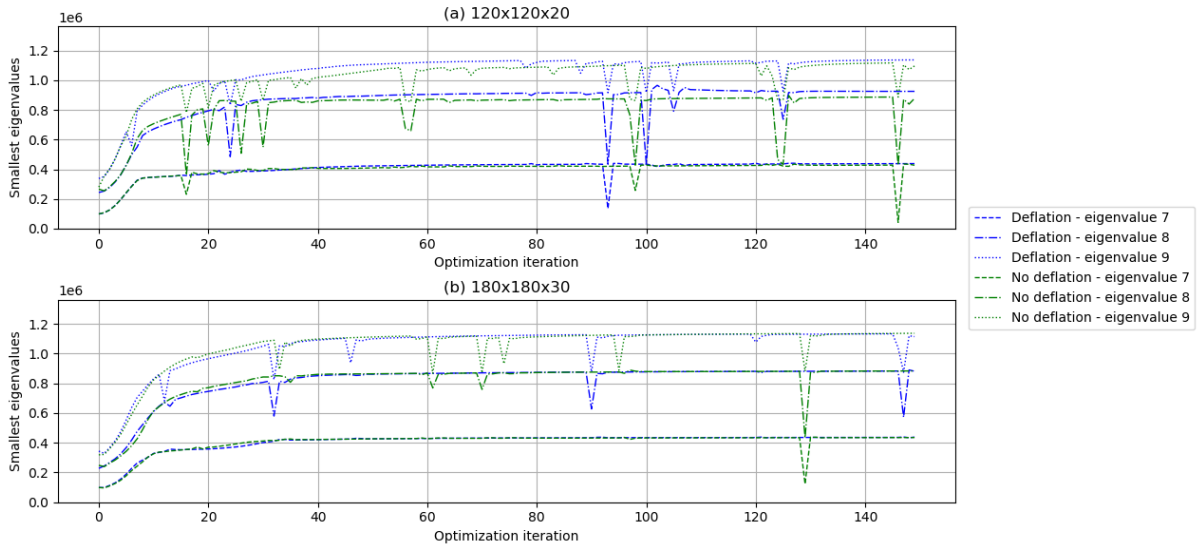
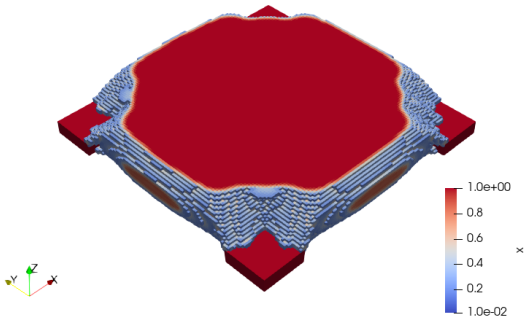


Figure 5.5: The convergence of the optimized eigenvalues, using IDR(4) and GAMG, on grid sizes $120 \times 120 \times 20$ (a) and $180 \times 180 \times 30$ (b), for 150 optimization iterations.

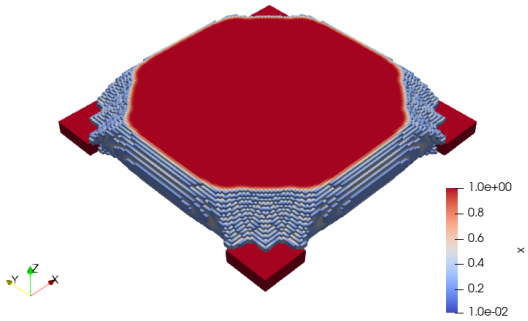
These figures show that for $180 \times 180 \times 30$ the convergence of the objective function and the eigenvalues is very similar for the eigenvalues when comparing deflation and no deflation. For $120 \times 120 \times 20$ eigenvalue 8 and 9 have a larger value for deflation than for no deflation. For this grid size the objective function has improved by using deflation.

Sometimes one or several of the eigenvalues dip down to a lower value (sometimes the value of the eigenvalue below), which causes a spike in the objective function. After the spike the objective function goes back into its convergence path. These spikes are called spurious modes. They seem to appear slightly less often when using deflation than when not using it.

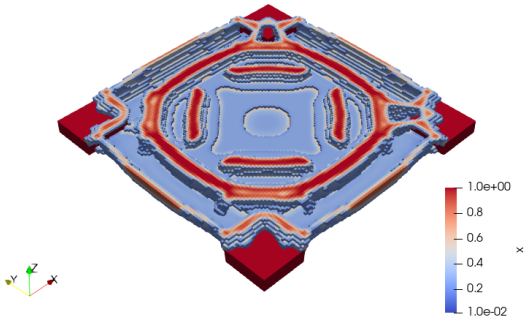
Because all matrix equations have been solved with and without deflation now, we can also look at the design (given by vector $\boldsymbol{x}$) in the two cases.



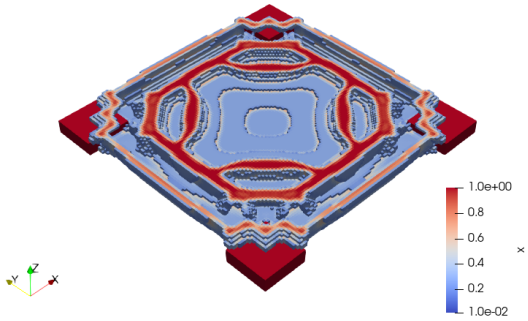
(a) Full design without using deflation.



(b) Full design using deflation.

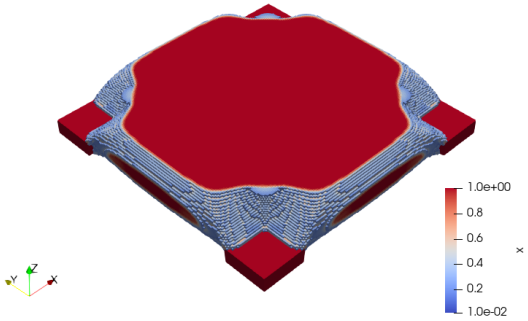


(c) Horizontal intersection without using deflation.

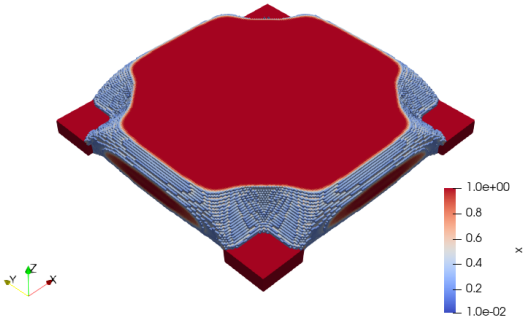


(d) Horizontal intersection using deflation.

Figure 5.6: The design $\boldsymbol{x}$ at the 150th optimization iteration, with and without deflation, using grid size $120 \times 120 \times 20$. Only elements with density greater than 0.2 are shown.

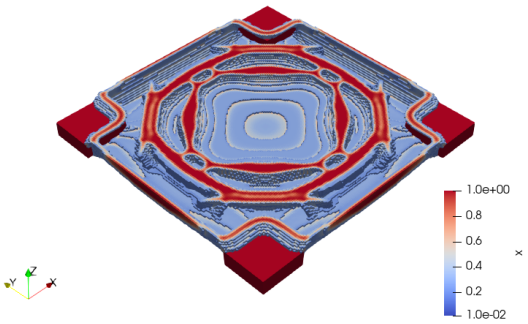


(a) Full design without using deflation.



(b) Full design using deflation.

(c) Horizontal intersection without using deflation.



(d) Horizontal intersection using deflation.

Figure 5.7: The design $\boldsymbol{x}$ at the 150th optimization iteration, with and without deflation, using grid size $180 \times 180 \times 30$. Only elements with density greater than 0.2 are shown.

Both without and with deflation the optimization results in a design that looks printable and similar to some extent. The plots representing deflation computations look slightly more smooth. In the plots of the full designs without using deflation there is a low-density chip of material in the corners that is gone when using deflation.

We have seen in the smaller grid sizes that deflation reduces the number of iterations needed to converge. Here we have performed the same number of iterations for both cases, so in theory the deflated case will have converged further. This might be the reason that the low-density chip is gone in the deflated case. On the other hand, the values of the objective function when not using deflation and using deflation are nearly equal. In general it cannot be concluded from sight whether one design is better than the other.

Chapter 6

Conclusions and recommendations

In this thesis we have investigated the effects of deflation on the number of iterations and the needed time to solve matrix equations used in Topology Optimization calculations. We have tested several Krylov methods, preconditioners and deflation types, on Static and Dynamic Topology Optimization.

We started off with a one-dimensional Poisson problem to illustrate the effect of applying deflation. To this example we applied the Conjugate Gradient method, deflated with region deflation based on the difference in coefficients in the governing equations. By using Jacobi preconditioning and region deflation the condition number was reduced by more than nine orders of magnitude and the number of iterations was brought below n. The example used three regions and because of the deflation the convergence worked in parallel between the regions.

After this, we have introduced the Static Topology Optimization problem using the SIMP method, applied to the MBB half-beam problem. This problem contained a stiffness matrix equation, that was solved using CG. We have applied several deflation types to this problem: eigenvector deflation, subdomain deflation, and various types of rigid body modes deflation. They improved the condition number and number of iterations. It appeared that basing the deflation regions off the near-zero-density regions (which act as rigid body modes) did not improve the number of iterations. However, using rigid body modes deflation with square regions, did improve the number of iterations greatly. It came close to the reference case of eigenvector deflation.

Hereafter, the Dynamic Topology Optimization problem was introduced. This problem contained an eigenvalue problem, shifted with a parameter σ^2, that was solved using model order reduction. Creating the bases for the model order reduction matrices involves solving several matrix equations, to which we applied eigenvector deflation (with eigenvalues from the previous optimization iteration), rigid body modes deflation on the entire grid, a combination of the two, and rigid body modes deflation in cubes combined with the eigenvectors. Because the matrix to be deflated was not always positive definite, other Krylov methods were introduced: GMRES, MINRES, BiCGSTAB and IDR(4). Apart from the Jacobi preconditioner, the GASM and GAMG preconditioner were introduced.

Firstly, combinations of these methods, preconditioners and deflation methods were applied to the solving of one of the matrix equations needed for the model order reduction. This was done next to the actual eigenvalue optimization. On small grid sizes the use of deflation already reduced the number of Krylov iterations and the needed time, and for larger grid sizes the effect increased.

The residual norm graphs showed that in all deflated cases the residuals converge better than in the undeflated cases. Especially the method/preconditioner combinations that were performing worse than others when undeflated, benefited a lot from the use of deflation. The best-performing Krylov methods were IDR(4) and, when applicable, CG. The best-performing preconditioner for smaller sizes is GASM. GAMG takes less iterations, but requires more prepare time. The larger the grid size, the less this effect becomes. For larger sizes GAMG is the best preconditioner.

Rigid body modes deflation and eigenvector deflation separately worked reasonably well, but using both worked better, and the best-performing type was eigenvectors with rigid body modes in cubes. This is a result that is similar to that of the two-dimensional static case. The difference between the last two increases with increasing grid size. The more cubes, the more iteration reduction due to deflation.

However, there is a balance, since more cubes results in a larger deflation subspace matrix $\boldsymbol{Z}$, introducing larger computation costs in computing the deflation matrix $\boldsymbol{P}$. The balance was calculated for combinations of IDR(4) and CG with GASM and GAMG, and for two grid sizes. In general, the optimal number of cubes increases when the computing time increases, because then the computation costs for $\boldsymbol{P}$ will start to dominate later. This is likely the reason why the optimum is slighty higher for GAMG than for GASM. No significant differences were found between IDR(4) and CG in this respect. The optimal number increases by a factor of four to five for an increase in grid size of two.

After this, the grid size was increased more, by performing the calculations in parallel on a computing cluster. Because of complications in the storage and creation of the deflation subspace matrix $\boldsymbol{Z}$ in parallel in PETSc, the deflation method of RBM in cubes + eigenvectors could not be used.

Overall, the number of iterations increased only slightly for increasing grid sizes. Instead the time needed per iteration increased. The larger the grid size, the less time computations with GAMG take compared to GASM. The undeflated computations with GAMG perform better with increasing grid size, so the factor of time gained using deflation is reduced. However, in all cases, there is still an improvement in both timing and number of iterations.

Using multiple processors is known to reduce the needed time and this was observed here as well. Using the available processors on the laptop it was found that increasing the number of processors reduced the time for all combinations of IDR(4) and CG with GASM and GAMG. None of the combinations performed significantly better than the others.

Lastly, deflation was applied to all matrix equations needed for the reduction bases. The largest grid sizes were used, and computations were done on a computing cluster. The method and preconditioner were IDR(4) and GAMG, as in the code used by the Topology Optimization project. As deflation type rigid body modes + eigenvectors was selected, which was consistently the best when using parallel computing. The time taken for the optimization was reduced by a factor of 1.60 for grid size $120 \times 120 \times 20$ and by a factor of 1.75 for grid size $180 \times 180 \times 30$. In the first case, deflation also caused the eigenvalues to converge to their optimal value faster. The design yielded by using deflation seemed slightly better converged after the same number of optimization iterations, seemingly implying that less optimization iterations are needed to reach the same results. However, since the objective functions of the two cases are very close to each other, the difference might be due to the fact that topology optimization always yields varying designs that are hard to value by sight.

Future research might include:

- In all cases, different types of deflation vectors can be tried. Since it is never certain whether a set of deflation vectors is the optimal choice, improvement is always possible. Other deflation vectors might be optimal for a different combination of iterative method and preconditioner.
- In the two-dimensional case, constructing deflation regions based on near-zero-density groups of elements did not have an effect on the iteration reduction. In the three-dimensional case, this idea could also be implemented, possibly with different results.
- Rigid body modes in cubes + eigenvector deflation was the best deflation type. However, it was not yet usable in parallel applications, due to PETSc limitations. If this obstacle could be overcome, the tests with grid sizes $90 \times 90 \times 15$ and larger could be run with this deflation type, likely yielding an even greater reduction in computation time. The optimal cube size could be investigated more thoroughly, using more grid sizes. Possibly a general relation could be derived relating the optimal cube size to the grid size.
- There are several other options to accelerate the eigenvalue optimization, some of which are currently being researched, such as skipping the computation of bases and using them for more than one optimization iteration. This is possible, because the further in the optimization process, the less the difference becomes between the optimization iterations and their bases. Conversely, on could argue that in the first few optimization iterations the bases do not have to be as accurate, since all variables are still changing fast. Possibly a less strict tolerance could be selected for the earlier iterations.

Bibliography

[1] Aage, N., Andreassen, E., and Lazarov, B. S. (2015). Topology Optimization using PETSc: An Easy-to-use, Fully Parallel, Open Source Topology Optimization Framework. *Structural and Multidisciplinary Optimization*, 51(3):565–572.

[2] Andreassen, E., Clausen, A., Lazarov, B. S., Schevenels, M., and Sigmund, O. (2010). Efficient Topology Optimization in MATLAB using 88 Lines of Code. *Structural and Multidisciplinary Optimization*, 43(1):1–16.

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

[4] Astudillo, R. and Van Gijzen, M. B. (2016). Induced Dimension Reduction Method for Solving Linear Matrix Equations. *Procedia Computer Science*, 80(2):222–232.

[5] Baggio, R., Franceschini, A., Spiezia, N., and Janna, C. (2017). Rigid Body Modes Deflation of the Preconditioned Conjugate Gradient in the Solution of Discretized Structural Problems. *Computers and Structures*.

[6] Besselink, B., Tabak, U., Lutowska, A., Van De Wouw, N., Nijmeijer, H., Rixen, D. J., Hochstenbach, M. E., and Schilders, W. H. (2013). A Comparison Of Model Reduction Techniques From Structural Dynamics, Numerical Mathematics and Systems and Control. *Journal of Sound and Vibration*, 332(19):4403–4422.

[7] Frank, J. and Vuik, C. (2003). On the Construction of Deflation-Based Preconditioners. *SIAM Journal on Scientific Computing*, 23(2):442–462.

[8] Grimme, E. (1997). *Krylov Projection Methods for Model Reduction*. PhD thesis, University of Illinois.

[9] Hestenes, M. and Stiefel, E. (1952). Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*.

[10] Huigsloot, M. (2018). Topology Optimization of Constrained Eigenfrequencies. MSc Thesis, Delft University of Technology.

[11] Jönsthövel, T. B. (2012). *The Deflated Preconditioned Conjugate Gradient Method Applied to Composite Materials*. PhD thesis, Delft University of Technology.

[12] Jönsthövel, T. B., Van Gijzen, M. B., Vuik, C., and Scarpas, A. (2013). On the Use of Rigid Body Modes in the Deflated Preconditioned Conjugate Gradient Method. *SIAM Journal on Scientific Computing*.

[13] Kaasschieter, E. F. (1988). Preconditioned Conjugate Gradients for Solving Singular Systems. *Journal of Computational and Applied Mathematics*, 24(1-2):265–275.

[14] Lingen, F. J., Bonnier, P. G., Brinkgreve, R. B., van Gijzen, M. B., and Vuik, C. (2014). A Parallel Linear Solver Exploiting the Physical Properties of the Underlying Mechanical Problem. *Computational Geosciences*, 18(6):913–926.

[15] Lu, T. T. and Shiou, S. H. (2002). Inverses of 2 $\times$ 2 Block Matrices. *Computers and Mathematics with Applications*, 43(1-2):119–129.

[16] Nicolaides, R. A. (1987). Deflation of Conjugate Gradients with Applications to Boundary Value Problems. *SIAM Journal on Numerical Analysis*.

[17] Paige, C. and Saunders, M. A. (1975). Solution Of Sparse Indefinite Systems Of Linear Equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629.

[18] Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems (Second Edition)*.

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

[20] Salimbahrami, B. and Lohmann, B. (2006). Order Reduction of Large Scale Second-Order Systems using Krylov Subspace Methods. *Linear Algebra and Its Applications*, 415(2-3):385–405.

[21] Schmit, L. (1960). Structual Design by Systematic Synthesis. In *2nd Conference on Electronic Computation*, pages 105–132.

[22] Sheikh, A. H. (2014). *Development Of The Helmholtz Solver Based On A Shifted Laplace Preconditioner And A Multigrid Deflation Technique*. PhD thesis, Delft University of Technology.

[23] Shewchuk, J. R. (1994). An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. *Carnegie Mellon University Report*.

[24] Sigmund, O. (2001). A 99 Line Topology Optimization Code Written in Matlab. *Structural and Multidisciplinary Optimization*.

[25] Sonneveld, P. (1989). CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52.

[26] Sonneveld, P. and Van Gijzen, M. B. (2008). IDR(s): A Family Of Simple And Fast Algorithms For Solving Large Nonsymmetric Systems Of Linear Equations. *Society*, 31(2):1035–1062.

[27] Tang, J. M. (2008). *Two-level Preconditioned Conjugate Gradient Methods with Applications to Bubbly Flow Problems*. PhD thesis, Delft University of Technology.

[28] Tang, J. M., Nabben, R., Vuik, C., and Erlangga, Y. A. (2007). Theoretical And Numerical Comparison Of Various Projection Methods Derived From Deflation, Domain Decomposition And Multigrid Methods. Technical report.

[29] Van der Veen, G., Langelaar, M., Van der Meulen, S., Laro, D., Aangenent, W., and Van Keulen, F. (2017). Integrating Topology Optimization in Precision Motion System Design for Optimal Closed-loop Control Performance. *Mechatronics*.

[30] Van der Vorst, H. A. (1992). 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.

[31] Vuik, C., Segal, A., and Meijerink, J. A. (1999). An Efficient Preconditioned CG Method for the Solution of a Class of Layered Problems with Extreme Contrasts in the Coefficients. *Journal of Computational Physics*.

[32] Yoon, G. H. (2010). Structural Topology Optimization for Frequency Response Problem using Model Reduction Schemes. *Computer Methods in Applied Mechanics and Engineering*, 199(25-28):1744–1763.

Appendix A: Notation, definitions and symbols

In this thesis, concepts, results and notation from linear algebra, iterative methods and general mechanics are used. A collection of those is presented here.

A.1 Notation

Vectors will always denoted in bold lowercases letters and matrices will be denoted in bold uppercase letters.

The vector $\boldsymbol{x} \in \mathbb{R}^n$ stands for:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n, \end{bmatrix}, \tag{A.1}$$

with $x_1, ..., x_n \in \mathbb{R},\ n \in \mathbb{N}$.

The all-zero and all-one vectors are denoted by $\mathbf{0}$ and $\mathbf{1}$, respectively.

The matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ stands for:

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}, \tag{A.2}$$

with $a_{11}, ..., a_{nm} \in \mathbb{R},\ n, m \in \mathbb{N}$.

A.2 Definitions

Definition A.1. *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. The inner product is defined as:*

$$(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i. \tag{A.3}$$

Definition A.2. *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{A} \in \mathbb{R}^{n \times m}$. The $\boldsymbol{A}$-inner product is defined as:*

$$(\boldsymbol{x}, \boldsymbol{y})_{\boldsymbol{A}} = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{y}. \tag{A.4}$$

Definition A.3. *Let $\boldsymbol{x} \in \mathbb{R}^n$. The 2-norm of $\boldsymbol{x}$ is defined as:*

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}. \tag{A.5}$$

In this thesis the 2-norm is used as the standard, so the 2 is omitted: $\|\boldsymbol{x}\| = \|\boldsymbol{x}\|_2$.

Definition A.4. *Let $\boldsymbol{x} \in \mathbb{R}^n$. The $\boldsymbol{A}$-norm of $\boldsymbol{x}$ is defined as:*

$$\|\boldsymbol{x}\|_{\boldsymbol{A}} = \sqrt{\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x}}. \tag{A.6}$$

Definition A.5. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Matrix $\boldsymbol{A}$ is called Symmetric Positive Definite (SPD) if:*

$$\boldsymbol{x}^T A\boldsymbol{x} > 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^{n\times 1} \setminus \{\boldsymbol{0}\}, \tag{A.7}$$

and Symmetric Positive Semi-Definite (SPSD) if:

$$\boldsymbol{x}^T A\boldsymbol{x} \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^{n\times 1}. \tag{A.8}$$

Definition A.6. *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. $\boldsymbol{x}$ and $\boldsymbol{y}$ are $\boldsymbol{A}$-orthogonal, or conjugate, if:*

$$\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{y} = 0. \tag{A.9}$$

Definition A.7. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. λ is an eigenvalue of $\boldsymbol{A}$ if there exists a vector $\boldsymbol{v} \neq \boldsymbol{0}$ such that:*

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}, \tag{A.10}$$

where $\boldsymbol{v}$ is defined as the eigenvector belonging to eigenvalue λ.

The set of n eigenvectors of $\boldsymbol{A}$ is called the spectrum, and denoted as:

$$\sigma(\boldsymbol{A}) = \{\lambda_1, ..., \lambda_n\}. \tag{A.11}$$

Definition A.8. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ with spectrum $\sigma(\boldsymbol{A}) = \{\lambda_1, ..., \lambda_n\}$. The condition number κ is defined as:*

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|. \tag{A.12}$$

If $\boldsymbol{A}$ is also SPD and $\lambda_i \in \mathbb{R}\ \forall i \leq n$, then

$$\kappa(\boldsymbol{A}) = \frac{\max_{1\leq i\leq n} \lambda_i}{\min_{1\leq i\leq n} \lambda_i} = \frac{\lambda_{max}(\boldsymbol{A})}{\lambda_{min}(\boldsymbol{A})}. \tag{A.13}$$

Definition A.9. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be SPSD with spectrum $\sigma(\boldsymbol{A}) = \{\lambda_1, ..., \lambda_n\}$ and $\lambda_i \in \mathbb{R}\ \forall i \leq n$. The effective condition number κ_{eff} is defined as*

$$\kappa_{\mathit{eff}}(\boldsymbol{A}) = \frac{\lambda_{max}(\boldsymbol{A})}{\lambda_{small}(\boldsymbol{A})}, \tag{A.14}$$

where λ_{small} is the smallest nonzero eigenvalue.

Definition A.10. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. If we have:*

$$a_{ij} = 0\ \forall j < i - k_1 \text{ and } \forall j > i + k_2 \tag{A.15}$$

for certain $k_1, k_2 \geq 0$, then k_1 and k_2 are called the lower and upper bandwidth, respectively.

Furthermore, the bandwidth is defined as $\max\{k_1, k_2\}$.

Definition A.11. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times m}$. The column space (or range) of $\boldsymbol{A}$ is the span of its columns. We denote this by $\mathcal{R}(\boldsymbol{A})$.*

The dimension of the column space is called the rank of $\boldsymbol{A}$. We denote this by rank($\boldsymbol{A}$).

The null space (or kernel) of $\boldsymbol{A}$ is the space spanned by all vectors $\boldsymbol{x}$ for which $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$. We denote this by $\mathcal{N}(\boldsymbol{A})$.

A.3 Symbols

In the different chapters concepts have been used from different research areas which have their own conventions regarding symbols. An effort was made to keep the symbols as clear and as non-contradictory as possible. Below a list of symbols used in the different chapters of this thesis can be found for extra clarity.

Chapter 1 & 2

$\boldsymbol{A}$	General matrix in matrix equation
$\boldsymbol{b}$	Right-hand-side of matrix equation
$\boldsymbol{E}$	Galerkin matrix
$\boldsymbol{M}$	Preconditioner matrix
n	Size of matrix $\boldsymbol{A}$
$\boldsymbol{P}$	Deflation matrix
$\boldsymbol{Q}$	Correction matrix
$\boldsymbol{p}$	Search direction
$\boldsymbol{r}$	Residual vector
$\hat{\boldsymbol{r}}$	Deflated residual
$\boldsymbol{x}$	General matrix equation solution
$\hat{\boldsymbol{x}}$	Deflated solution
$\boldsymbol{Z}$	Deflation subspace matrix

Chapter 3

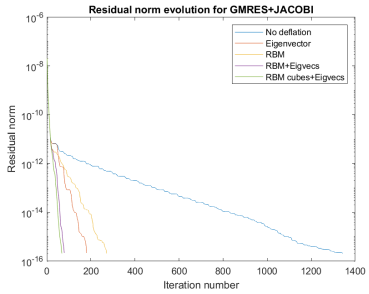
$\boldsymbol{A}$	General matrix in matrix equation
$\boldsymbol{b}$	Right-hand-side of matrix equation
c	Compliance
E	Elasticity modulus
E_0	Elasticity modulus of the material
E_{min}	Minimal elasticity modulus
f_V	Prescribed volume fraction
$\boldsymbol{f}$	Force vector
$\boldsymbol{k}_0$	Element stiffness matrix
$\boldsymbol{K}$	Stiffness matrix
$\boldsymbol{M}$	Mass matrix
n	Size of matrix $\boldsymbol{K}$ (number of degrees of freedom)
N	Number of elements in the design domain
p	Penalization power
$\boldsymbol{P}$	Deflation matrix
$\boldsymbol{u}$	Displacement vector
$\boldsymbol{u}_e$	Displacement vector of e-th element
$V(\boldsymbol{x})$	Material volume of the design
V_0	Constant design domain volume
$\boldsymbol{x}$	Vector of design variables (i.e. element density)
x_e	Design value (i.e. density) of element e

Chapter 4 & 5

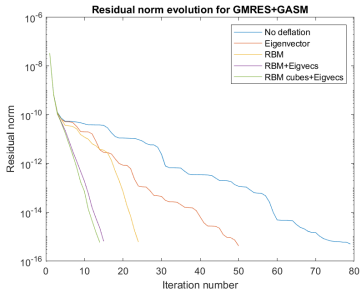
$\boldsymbol{A}$	Shorthand matrix in state-space formulation
$\bar{\boldsymbol{b}}$	Collection of three normalized actuation vectors
$\bar{\boldsymbol{b}}_r$	Reduced version of $\bar{\boldsymbol{b}}$
$\boldsymbol{b}$	Shorthand matrix in state-space formulation
$\bar{\boldsymbol{c}}^T$	Collection of three normalized sensing vectors
$\bar{\boldsymbol{c}}_r$	Reduced version of $\bar{\boldsymbol{c}}$
$\boldsymbol{c}$	Shorthand matrix in state-space formulation
$\boldsymbol{d}$	Vector of nodal displacements
$\boldsymbol{d}_r$	Vector of nodal displacements of reduced model
$\boldsymbol{E}$	Shorthand matrix in state-space formulation
$H(s)$	Transfer function in Laplace domain
$\hat{H}(s)$	Transfer function of reduced model
λ_i	i-th eigenvalue of the eigenvalue problem
k	Order of the reduced order model
$\boldsymbol{K}$	Stiffness matrix
$\boldsymbol{K}_r$	Stiffness matrix of reduced model
m	Number of maximized small eigenvalues
$\boldsymbol{M}$	Mass matrix
$\boldsymbol{M}_r$	Mass matrix of reduced model
$M_i(s_0)$	i-th moment around s_0
$\hat{M}_i(s_0)$	i-th moment around s_0 of reduced model
n	Size of matrices $\boldsymbol{K}$ and $\boldsymbol{M}$ (number of degrees of freedom)
s	Laplace variable
σ^2	Shifting parameter
$\boldsymbol{u}$	Vector of inputs (forces)
V	Prescribed volume fraction
$V(\boldsymbol{x})$	Material volume of the design
V_0	Constant design domain volume
$\boldsymbol{V}$	Reduction matrix, basis for input subspace
$\boldsymbol{W}$	Reduction matrix, basis for output subspace
$\boldsymbol{x}$	Vector of design variables (i.e. element density)
$\boldsymbol{y}$	Vector of outputs
$\boldsymbol{z}_i$	i-th eigenvector (eigenmode) of the eigenvalue problem

Appendix B: Residual Norm graphs

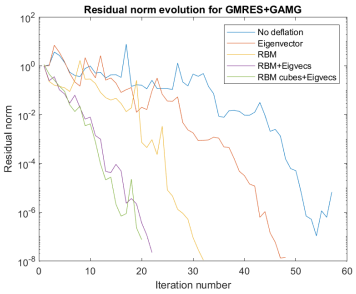
In this Appendix all residual norm graphs are shown that correspond to the results in Tables 5.1, 5.2, 5.3, 5.6 and 5.7. Note that for the BiCGSTAB method the number of iterations should be regarded as the double of what is shown in the graphs. Furthermore, PETSc only outputs the relative residual norm for IDR, as opposed to the absolute residual norm for the other methods.



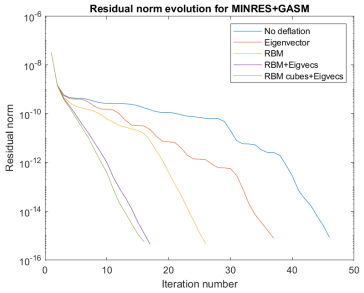
(a) The residual norm evolution using GMRES with Jacobi preconditioner.



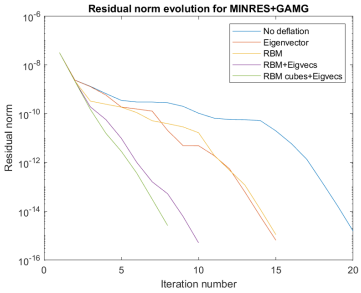
(b) The residual norm evolution using GMRES with GASM preconditioner.



(c) The residual norm evolution using GMRES with GAMG preconditioner.

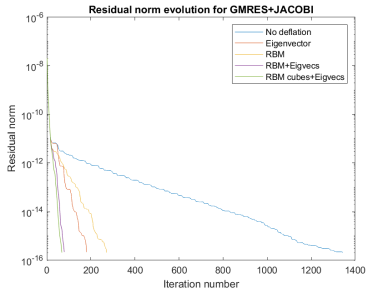


(d) The residual norm evolution using MINRES with GASM preconditioner.

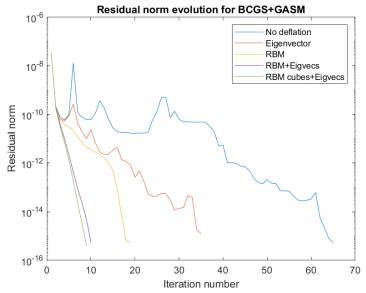


(e) The residual norm evolution using MINRES with GAMG preconditioner.

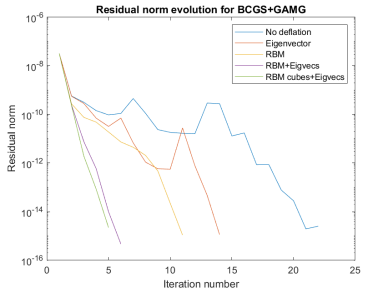
Figure B.1: The residual norm evolution for GMRES and MINRES combined with the different preconditioners and deflation types, using grid size $12 \times 12 \times 2$ and the tenth optimization iteration.



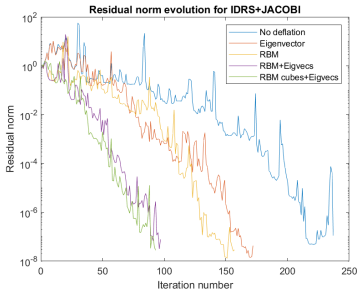
(a) The residual norm evolution using BiCGSTAB with Jacobi preconditioner.



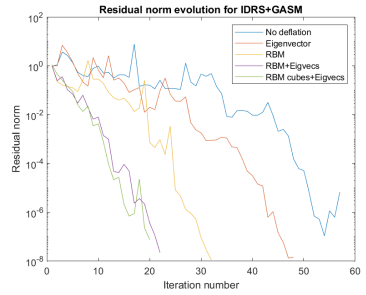
(b) The residual norm evolution using BiCGSTAB with GASM preconditioner.



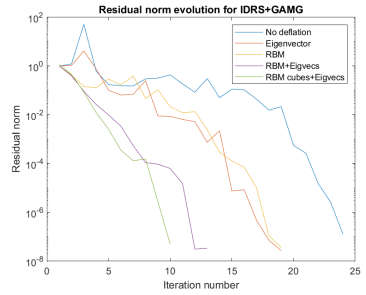
(c) The residual norm evolution using BiCGSTAB with GAMG preconditioner.



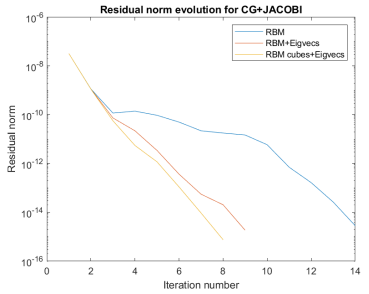
(d) The residual norm evolution using IDR(4) with Jacobi preconditioner.



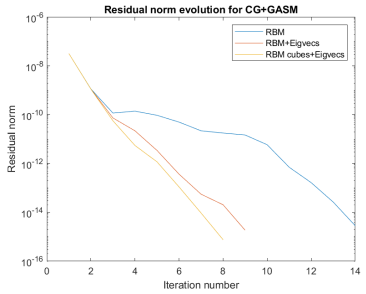
(e) The residual norm evolution using IDR(4) with GASM preconditioner.



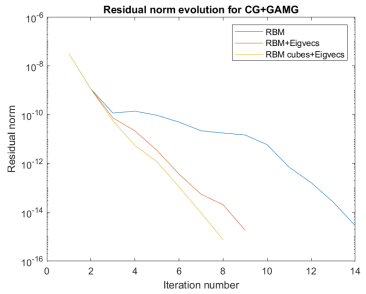
(f) The residual norm evolution using IDR(4) with GAMG preconditioner.



(g) The residual norm evolution using CG with Jacobi preconditioner.

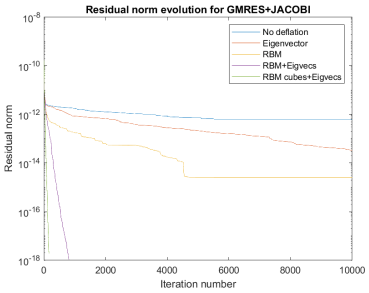


(h) The residual norm evolution using CG with GASM preconditioner.

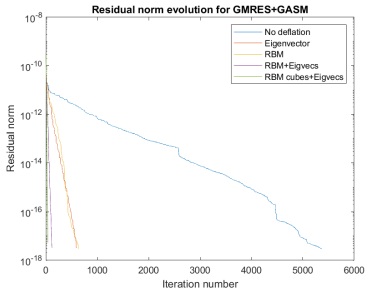


(i) The residual norm evolution using CG with GAMG preconditioner.

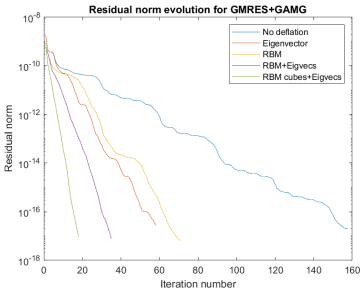
Figure B.2: The residual norm evolution for BiCGSTAB, IDR(4) and CG combined with the different preconditioners and deflation types, using grid size $12 \times 12 \times 2$ and the tenth optimization iteration.



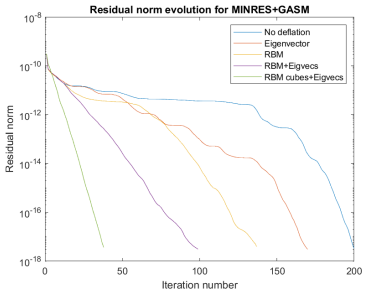
(a) The residual norm evolution using GMRES with Jacobi preconditioner.



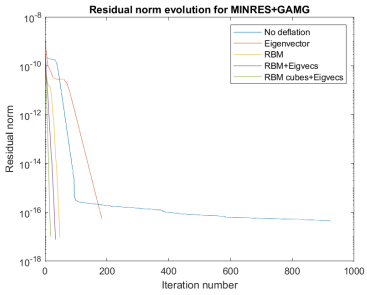
(b) The residual norm evolution using GMRES with GASM preconditioner.



(c) The residual norm evolution using GMRES with GAMG preconditioner.

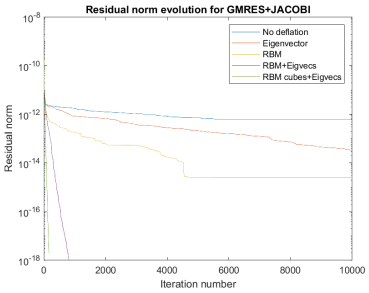


(d) The residual norm evolution using MINRES with GASM preconditioner.

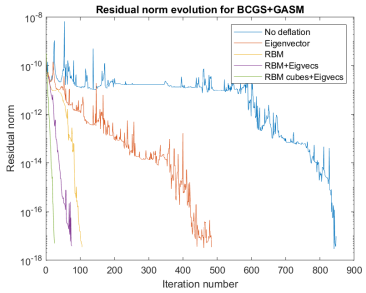


(e) The residual norm evolution using MINRES with GAMG preconditioner.

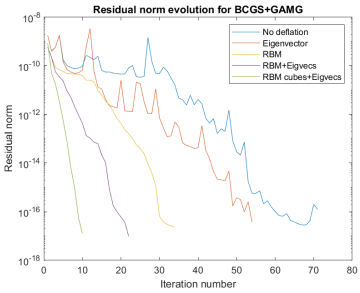
Figure B.3: The residual norm evolution for GMRES, MINRES and BiCGSTAB combined with the different preconditioners and deflation types, using grid size $30 \times 30 \times 5$ and the tenth optimization iteration.



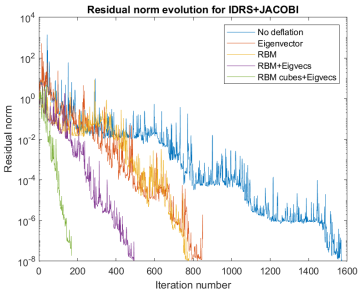
(a) The residual norm evolution using BiCGSTAB with Jacobi preconditioner.



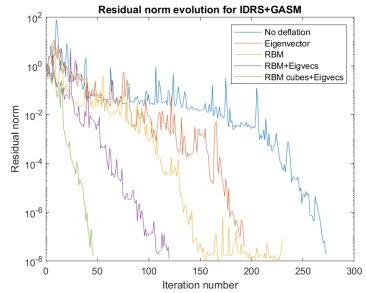
(b) The residual norm evolution using BiCGSTAB with GASM preconditioner.



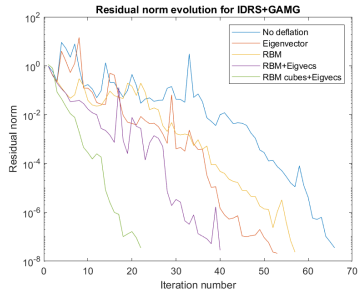
(c) The residual norm evolution using BiCGSTAB with GAMG preconditioner.



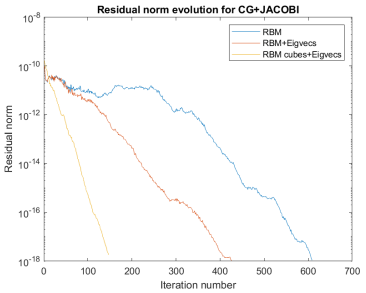
(d) The residual norm evolution using IDR(4) with Jacobi preconditioner.



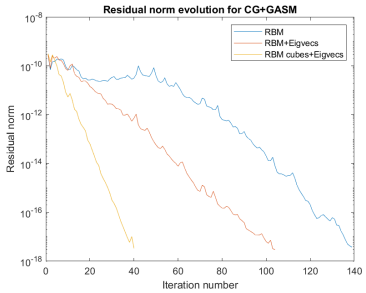
(e) The residual norm evolution using IDR(4) with GASM preconditioner.



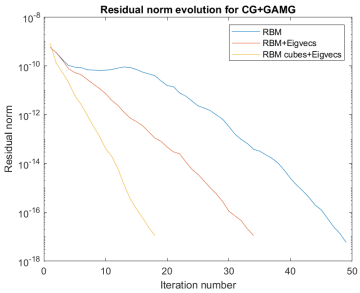
(f) The residual norm evolution using IDR(4) with GAMG preconditioner.



(g) The residual norm evolution using CG with Jacobi preconditioner.

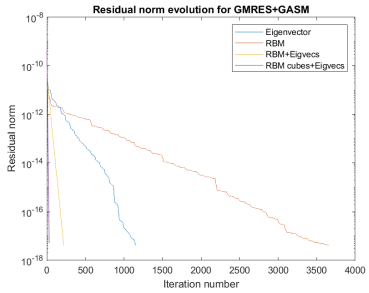


(h) The residual norm evolution using CG with GASM preconditioner.

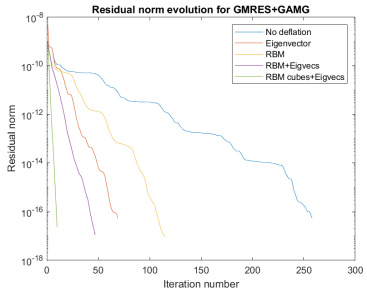


(i) The residual norm evolution using CG with GAMG preconditioner.

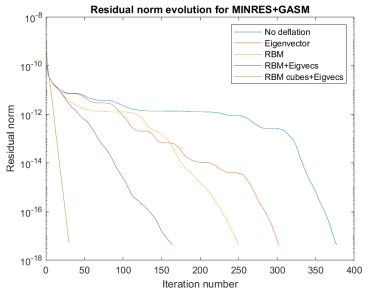
Figure B.4: The residual norm evolution for BiCGSTAB, IDR(4) and CG combined with the different preconditioners and deflation types, using grid size $30 \times 30 \times 5$ and the tenth optimization iteration.



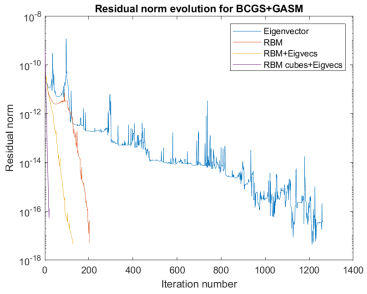
(a) The residual norm evolution using GMRES with GASM preconditioner.



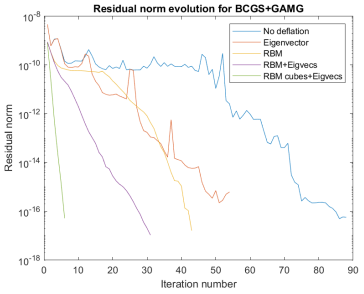
(b) The residual norm evolution using GMRES with GAMG preconditioner.



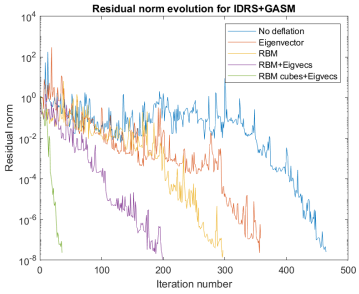
(c) The residual norm evolution using MINRES with GASM preconditioner.



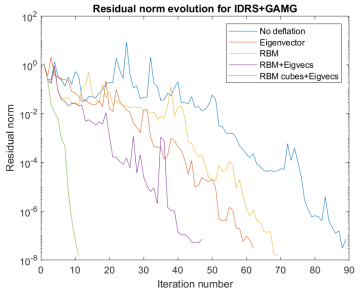
(d) The residual norm evolution using BiCGSTAB with GASM preconditioner.



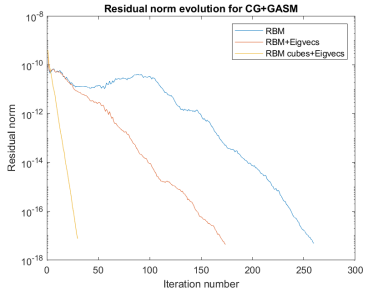
(e) The residual norm evolution using BiCGSTAB with GAMG preconditioner.



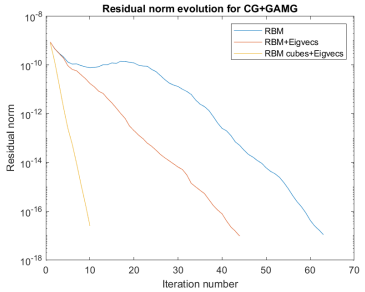
(f) The residual norm evolution using IDR(4) with GASM preconditioner.



(g) The residual norm evolution using IDR(4) with GAMG preconditioner.

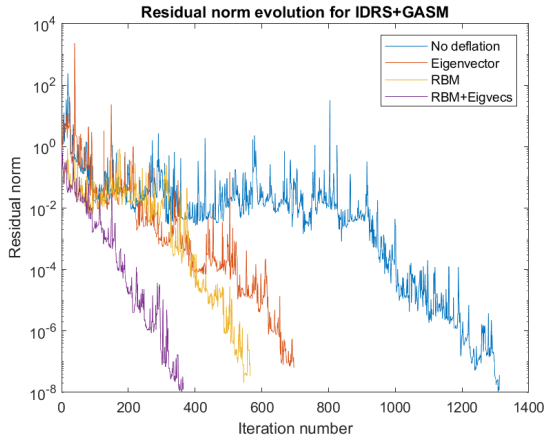


(h) The residual norm evolution using CG with GASM preconditioner.

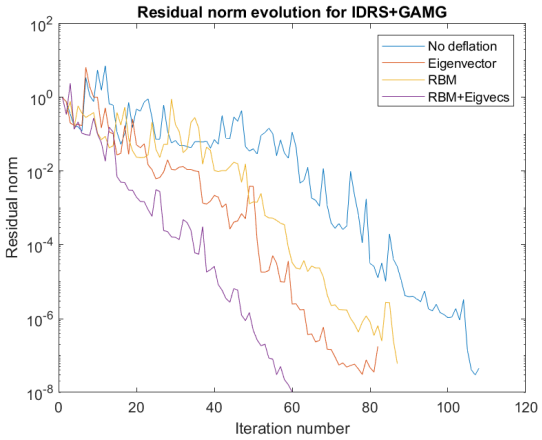


(i) The residual norm evolution using CG with GAMG preconditioner.

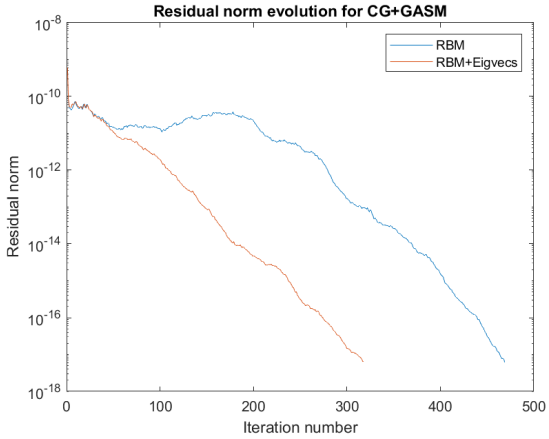
Figure B.5: The residual norm evolution for the different methods, combined with the GASM and GAMG preconditioners and different deflation types, using grid size $60 \times 60 \times 10$ and the tenth optimization iteration.



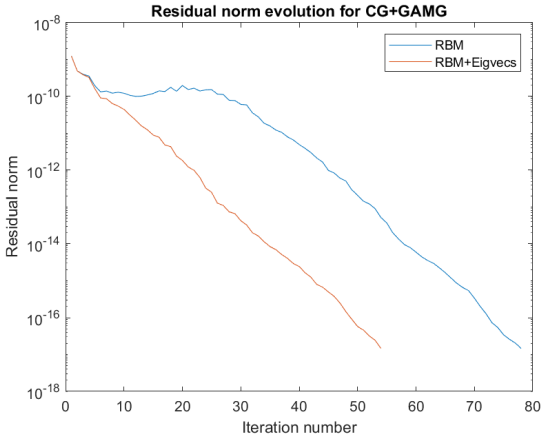
(a) The residual norm evolution using IDR(4) with GASM preconditioner.



(b) The residual norm evolution using IDR(4) with GAMG preconditioner.

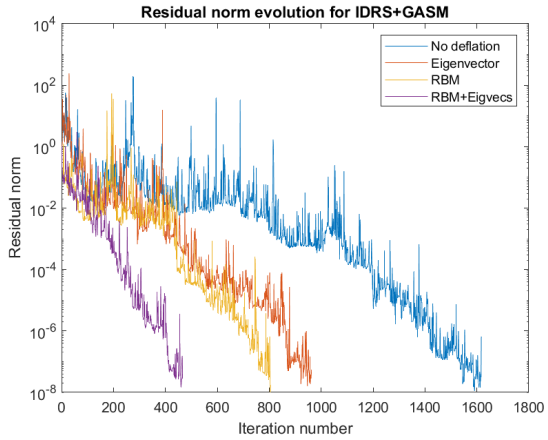


(c) The residual norm evolution using CG with GASM preconditioner.

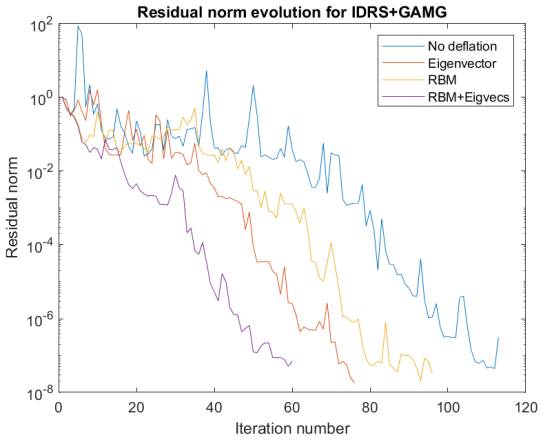


(d) The residual norm evolution using CG with GAMG preconditioner.

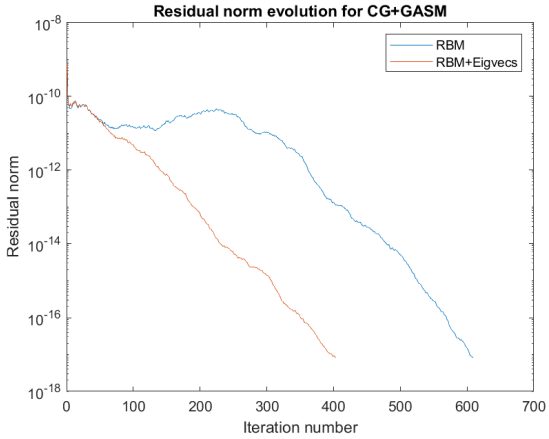
Figure B.6: The residual norm evolution for the different methods, combined with the GASM and GAMG preconditioners and different deflation types, using grid size $90 \times 90 \times 15$ and the tenth optimization iteration. The matrix equation was solved using 10 processors.



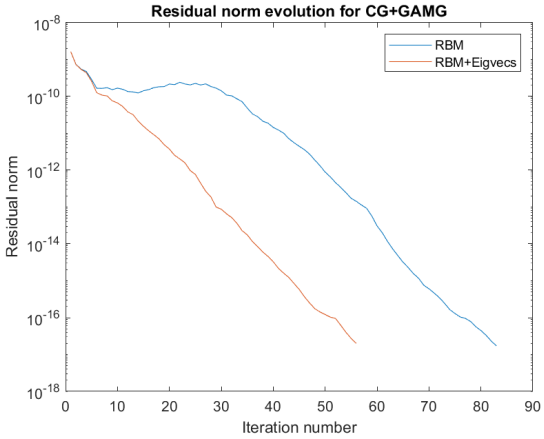
(a) The residual norm evolution using IDR(4) with GASM preconditioner.



(b) The residual norm evolution using IDR(4) with GAMG preconditioner.



(c) The residual norm evolution using CG with GASM preconditioner.



(d) The residual norm evolution using CG with GAMG preconditioner.

Figure B.7: The residual norm evolution for the different methods, combined with the GASM and GAMG preconditioners and different deflation types, using grid size $120 \times 120 \times 20$ and the tenth optimization iteration. The matrix equation was solved using 10 processors.