Numerical Methods for Large Thermo-Mechanical Systems

E. I. Maquelin

TU Delft

Numerical Methods for Large Thermo-Mechanical Systems

by

E. I. Maquelin

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday July 20, 2021 at 14:00.

Student number:	5144353	
Project duration:	December, 2020 – July, 2021	
Thesis committee:	Prof. dr. ir. C. Vuik,	TU Delft, supervisor
	Prof. dr. ir. H. X. Lin,	TU Delft
	Dr. ir. V. Dolk,	ASML

Master programme:	Applied Mathematics
Specialisation:	Computational Science and Engineering
Faculty:	EEMCS

TU Delft

Abstract

Numerical methods are investigated for solving large-scale sparse linear systems of equations, that can be applied to thermo-mechanical models and wafer-slip models. This thesis examines efficient numerical methods, in terms of memory, number of iterations required for convergence, and computation time. To be more specific, algebraic multigrid (AMG) methods and deflation methods are considered as preconditioners for the conjugate gradient method. We investigate if smoothed aggregation AMG or adaptive smoothing and prolongation based AMG improve upon the classical Ruge-Stüben AMG. It is shown that Ruge-Stüben AMG needs fewer iterations for the test problems. However, smoothed aggregation AMG has a smaller data-size, which is of interest for situations with limited memory or large systems of equations. Moreover, the mechanical problems considered have a coefficient matrix with a block structure, which can be exploited by preconditioners like block Jacobi or the incomplete block Cholesky decomposition; but also the smoothed aggregation AMG can take the block structure into account when creating coarser grids. Further, we examine if the results of the conjugate gradient method can be improved by adding a deflation preconditioner based on the proper orthogonal decomposition or rigid body modes. They are combined with a direct or stationary iterative preconditioner, resulting in two-level preconditioned conjugate gradient methods. The various implementations of such methods are discussed, and the deflation preconditioner is shown to generally reduce the number of iterations compared to the single preconditioner.

Contents

List of Abbreviations

A-DEF1	= Adapted Deflation variant 1
A-DEF2	= Adapted Deflation variant 2
AMG	= Algebraic Multigrid
aSP-AMG	= adaptive Smoothing and Prolongation based AMG
BiCG	= Biconjugate Gradient
BiCGSTAB	= BiCG Stabilised
(P)CG	= (Preconditioned) Conjugate Gradient
CGS	= Conjugate Gradient Squared
DEF1	= Deflation variant 1
dof	= degree of freedom
DPLS	= Dynamic Pattern Least Squared
EUV	= Extreme Ultra Violet
GMG	= Geometric Multigrid
GMRES	= Generalised Minimum Residual
IDR	= Induced Dimension Reduction
MG	= Multigrid
MIS	= Maximum Independent Set
MOR	= Model Order Reduction
MPC	= Model Predictive Control
PDE	= Partial Differential Equation
POD	= Proper Orthogonal Decomposition
RBM	= Rigid Body Modes
RS-AMG	= Ruge-Stüben Algebraic Multigrid
SA-AMG	= Smoothed Aggregation Algebraic Multigrid
SoC	= Strength of Connection
(S)SOR	= (Symmetric) Successive Over-Relaxation
SPD	= Symmetric Positive Definite
SPSD	= Symmetric Positive Semi-Definite
SRQCG	= Simultaneous Rayleigh Quotient minimisation by Conjugate Gradients
2L-PCG	= Two-Level Preconditioned Conjugate Gradient

List of Symbols

(unless stated differently)

A	= coefficient matrix
A_c	= coarse grid operator
Ap	= abstraction of A
b	= right-hand side
C	= coarse grid nodes
d	= deformation vector
D	= diagonal of A
e	= error
E	= Galerkin matrix
θ	= parameter related to dropping in AMG
F	= fine grid nodes
G	= graph
I	= interpolation operator
$\widetilde{I}$	= tentative interpolation operator
$\mathcal{I}$	= identity matrix
$\mathcal{K}_i(\cdot,\cdot)$	= ith order Krylov subspace
L	= lower triangular matrix
M	= preconditioner or smoother
μ	= time-steps threshold in POD-based deflation
N	= size of A
$\mathcal{N}(A)$	= nullspace/kernel of matrix A
N_t	= dimension of test space
P	= deflation matrix
P'	= fill-in nonzero pattern
Q	= correction matrix
r	= residual
r_0	= initial residual
R	= restriction operator
$\mathcal{R}(A)$	= range of matrix A
S_c	= SoC matrix
T	= thermal state vector
Δt	= time step
τ	= iterations threshold in POD-based deflation
U	= upper triangular matrix
W	= matrix containing interpolation weights
X	= basis test space
x	= unknown/solution
$\widetilde{x}$	= deflated solution
x_0	= initial guess
Z	= deflation-subspace matrix
ω	= damping parameter

Chapter 1

Introduction

This thesis investigates numerical methods for large-scale sparse linear systems of equations, that can be applied in the context of control and system design for thermo-mechanical systems. These types of problems are for example relevant to the development of high precision lithography machines and EUV systems in particular. EUV machines operate under vacuum conditions as the EUV light has such a small wavelength that it is absorbed by air. In the vacuum it is hard to get rid of the heat that arises due to exposure to EUV. The thermal disturbances can in turn cause imaging distortions. The imaging distortions have to be actively suppressed. This encounters challenges like handling the limited range of a thermal actuator and the many possibilities of spatially distributed thermal loads [1]. Therefore, high fidelity models are employed, yielding large dynamical systems which have to be solved quickly. For such situations, we search for efficient numerical methods, in terms of memory, number of iterations, and computation time.

Likewise, the wafer-slip model is investigated as it also entails solving large-scale sparse linear systems. The model represents the placing of a wafer on its support, which consists of a number of springs holding the wafer in place. The wafer-slip model is a combination of two thermo-mechanical models which are connected by spring elements that can slip. The slip and stick behaviour make this a challenging problem as the behaviour of the system evolves over time.

The aim is to find efficient methods for large-scale sparse linear systems. Recent studies [2–6] have come up with ways of combining different solvers, creating even more effective ones. Common techniques for solving large-scale systems are Krylov subspace methods. In this paper the main solver is the Krylov subspace method preconditioned conjugate gradient, since the problems considered are symmetric positive definite. As preconditioners/smoothers we consider multigrid, deflation, direct and stationary iterative methods. First, among the algebraic multigrid (AMG) methods not only the classical Ruge-Stüben AMG is used as preconditioner, but also smoothed aggregation AMG and adaptive smoothing and prolongation based AMG, using a direct method or a stationary iterative method as smoother. Second, we consider deflation methods based on the proper orthogonal decomposition or rigid body modes as preconditioners; resulting in a so-called two-level preconditioned conjugate gradient method, where the other preconditioner is a direct or stationary iterative method. Various implementation variants of such two-level solvers are examined.

Third, block preconditioners can be used when the coefficient matrix has a block structure – as is the case for the mechanical part of the thermo-mechanical model and the wafer-slip model – to exploit this structure and improve the results. The methods are applied to test problems representing the thermo-mechanical and wafer-slip models, but a smaller version of the original models to enable testing.

The paper is organised in the following way: the next chapter presents some well-known basic techniques for solving linear systems. In Chapter 3 the model problems are introduced, including current state-of-the-art methods reported in literature. Hereafter, the algebraic multigrid methods, deflation methods and block preconditioners are discussed in Chapters 4, 5, and 6 respectively. These are subsequently applied in Chapters 7 and 8 as preconditioners to the conjugate gradient method for test problems representing the thermo-mechanical and wafer-slip model respectively, to analyse their performance numerically. The final Chapter 9 gives a conclusion on the found results and recommendations for future research.

Chapter 2

Preliminaries

This chapter introduces some well-known methods for solving (large) linear systems of equations, denoted $Ax = b$. Throughout this paper it is assumed that A is real. For each algorithm, its classification with characteristics and abbreviations are given. For more details on the algorithms in this chapter the following sources can be consulted [7–9].

Methods for solving linear equations can broadly be divided into two categories: direct methods and iterative methods [7]. Direct methods find the solution in a finite number of operations in exact arithmetic [10]. Usually, the methods first factorise the coefficient matrix, and then the factored form is used to solve the linear system via forward and backward substitution. For instance, in the direct method the LU-decomposition, compute first $A = LU$ where L is unit lower and U is upper triangular. Then solve $Ly = b$ via forward substitution for y, followed by solving $Ux = y$ via backward substitution for x [8]. Another well-known example of direct methods is Cholesky decomposition. The Cholesky decomposition of A exists and is unique when A is symmetric positive definite (SPD), and is then given by $A = LL^T$, where L is a lower triangular matrix with real, positive diagonal entries. If instead a sparse approximation of the Cholesky decomposition is used, this is called the incomplete Cholesky decomposition [10]. Direct methods are accurate, robust and reliable. While they are the preferred methods for dense systems of small/medium size, they become expensive in terms of memory and computational demands for large or sparse systems. Namely, for sparse systems the phenomenon of fill-in occurs. This means that at the locations where zero entries are in the original matrix, nonzero entries appear in the decomposition hence sparsity is lost. In such cases, it is better to use iterative methods [8].

The accuracy in iterative methods is less than the theoretical machine precision of direct methods for well-conditioned problems. But often, this is justified by the fact that the models themselves also have a certain level of inaccuracy. The iterative methods use a given initial guess to generate a sequence of improving approximations to the solution. The two main classes of iterative methods are the stationary iterative methods and the Krylov subspace methods. Stationary iterative methods find the solution by solving a fixed point equation, via an operator that approximates the coefficient matrix; they are completely characterised by a single matrix. Examples of stationary iterative methods are (damped) Jacobi, Gauss-Seidel and (symmetric) successive over-relaxation ((S)SOR). For example, the damped Jacobi iteration is given by Equation 2.1, where D is the diagonal of A, and L and U are respectively the strict lower and the upper triangular part. The standard/non-damped Jacobi is obtained when $\omega = 1$.

$$x_{k+1} = \omega D^{-1}(b - (L + U)x_k) + (1 - \omega)x_k \tag{2.1}$$

The second subclass, the Krylov subspace methods, employ Krylov subspaces, which are defined by Equation 2.2 given some matrix B and some vector z.

$$\mathcal{K}_i(B, z) := span\{z, Bz, \dots, B^{i-1}z\} \tag{2.2}$$

The Krylov subspace methods search in the k^{th} iteration for an approximate solution in the k^{th} order Krylov subspace $x_0 + \mathcal{K}_k(A, r_0)$, where x_0 denotes the initial guess and $r_0 := b - Ax_0$ the initial residual. Examples of Krylov subspace methods are the induced dimension reduction method (IDR(s)), generalised minimum residual method (GMRES), (preconditioned) conjugate gradient ((P)CG), biconjugate gradient (BiCG), conjugate gradient squared (CGS) and BiCG stabilised (BiCGSTAB). CG is only applicable to SPD matrices, and then it has three nice properties: it generates approximations in $x_0 + \mathcal{K}_k(A, r_0)$, the error has minimal A-norm, and it has short recurrences (i.e. only results of the previous iteration are needed; work and memory do not increase when the number of iterations increases). It is impossible to construct a Krylov subspace method that has all these three nice properties and is applicable to general matrices. The other examples of Krylov subspace methods given above are not restricted to SPD matrices, at the cost of losing one of the three nice properties. GMRES has long recurrences; BiCG, CGS, BiCGSTAB, and IDR(s) do not have the optimality property regarding the error norm [8].

Both the direct methods and the stationary iterative methods are not efficient as standalone solvers for large-scale systems, but can be used as preconditioner or smoother. A preconditioner turns the problem into a better conditioned one. Namely, the convergence behaviour of Krylov subspace methods depends on the distribution of the eigenvalues of the coefficient matrix. The preconditioner is then a matrix M chosen such that the preconditioned system in Equation 2.3 has the same solution as original system $Ax = b$, but has a more favourable eigenvalue distribution [8].

$$M^{-1}Ax = M^{-1}b \tag{2.3}$$

Preconditioning is for instance used in PCG. The convergence of CG depends on the condition number and the amount and distribution of near-zero eigenvalues, since the corresponding eigenvectors do significantly contribute to the solution but may converge slowly. Preconditioning is then used to improve the convergence behaviour. The resulting PCG has cheap iterations, is easy to implement and does not require too much memory; though compared to the CG, the preconditioner does increase the work per iteration and the memory demand [11].

A smoother removes fast/(spatially) high frequency error-components. The damped Jacobi method can for instance be used as a smoother, then $M := D$ is the diagonal of A, and the smoothed solution is given in Equation 2.4.

$$x = x + M^{-1}(b - Ax) \tag{2.4}$$

Applying (multiple) Jacobi iterations reduces high frequency error-components while allowing the low frequency components to pass without much change, so smoothen the error [8].

Chapter 3

Problem Description

This thesis determines which solver configurations are suitable in the context of thermo-mechanical control problems, considering both the transient and steady-state temperature and deformation. A numerical analysis is performed on a small-size thermo-mechanical model of a mirror, which originally contains about 75,000 nodes. The dynamics of a finite element model of the thermal system can be described by a continuous-time model as in Equation 3.1, where T is the thermal state vector, u the actuation signal, w the disturbance signal, and A, B_u, B_w and E are sparse system matrices.

$$E\dot{T} = AT + B_u u + B_w w \tag{3.1}$$

Because the continuous model contains a time derivative of the thermal vector (denoted $\dot{T}$), the numerical integration scheme implicit Euler is used to construct the discrete-time system in Equation 3.2, where the coefficient matrix $E - \Delta t A$ is symmetric and Δt denotes the chosen fixed time-step.

$$(E - \Delta t A)T_{k+1} = ET_k + \Delta t(B_u u_k + B_w w_k) \tag{3.2}$$

The thermally-induced deformation can be computed for a given thermal state vector T. As the nodes can displace in three different directions, the mechanical model has about 225,000 mechanical degrees of freedom. The deformation d can be found by solving the mechanical model in Equation 3.3, where K_M is the sparse mechanical- and K_T the sparse thermal-stiffness matrix, and $\epsilon(T)$ the – possibly non-linear – temperature-strain relation.

$$K_M d = K_T \epsilon(T) \tag{3.3}$$

For the wafer-slip model, an efficient solver is sought as well. It models the process of placing a wafer on a wafer clamp, which is a device that supports the wafer and keeps it in place. The corresponding wafer displacements u are computed according to Equation 3.4, where F^E denotes the external forces. Including the slip in the model – done according to Coulomb friction diagrams – causes the stiffness matrix K to depend on solution u itself.

$$K(u)u = F^E \tag{3.4}$$

The solution of Equation 3.4 can be computed via an explicit or implicit scheme. The explicit scheme assumes that when solving the equation at time-step n for u_n, the stiffness matrix K is determined by the solution of the previous time step u_{n-1}. So the explicit scheme requires solving $K(u_{n-1})u_n = F_n^E$ at time step n. Explicit schemes have as advantage that they are easy to implement and are mathematically and physically less challenging than implicit schemes. The drawback is that at every time step an error is introduced by the assumption that K is determined by u_{n-1} instead of u_n [12]. In order to keep these errors small, the increase in time per step must be chosen small enough. This can cause large CPU times when many time steps are required, as well as an increasing error. That is why implicit schemes are preferred. Implicit schemes require solving $K(u_n)u_n = F_n^E$ at time step n. Since K depends on the unknown solution u_n, the Newton method is employed. Newton's method is an iterative root-finding method; for finding the root of a scalar function $f(x)$ it iteratively solves $x^{i+1} = x^i - \frac{f(x^i)}{f'(x^i)}$ until the desired accuracy is reached [13]. For the wafer-slip model, this translates to solving Equation 3.5, where K_t denotes the tangent stiffness matrix and F^I the in-balance force vector.

$$K_t(u_n^{i-1})\Delta u^i = F_n^E - F_n^{I,i-1} \tag{3.5}$$

An effective way of solving accurate thermo-mechanical models is using state-of-the-art (multi-level) iterative methods that can efficiently solve large-scale sparse linear systems of the form $Ax = b$. Recall from Chapter 2 that both direct and stationary iterative methods are not efficient as standalone solvers for such systems. Therefore, the basis solver that is used is a Krylov subspace method. Furthermore, since Krylov subspace methods are iterative methods, they are given an initial guess, which can accelerate the computation of transient temperature or deflation simulations by using the solution of the previous time step as initial guess for the current time step. Since the problems discussed in this thesis are all SPD, the Krylov subspace method that will be used is PCG. This method has nice properties – as discussed in Chapter 2 – and moreover, the conjugate gradient algorithm is "one of the best known iterative techniques for solving sparse symmetric positive definite linear systems" [7]. As preconditioner many different methods can be used, like the traditional direct or stationary iterative methods, but also more advanced methods can be considered.

In this thesis we will focus on certain multigrid and deflation preconditioners. More specifically, within multigrid we will look into AMG methods, since these are also applicable for unstructured grids (contrary to GMG). The classical Ruge-Stüben AMG is compared to the more widely applicable smoothed aggregation AMG – "one of the most promising AMG methods" [14] – and the adaptive smoothing and prolongation based AMG – designed for usability and efficiency for structural mechanics problems [4]. Within deflation we will consider methods based on proper orthogonal decomposition (POD) and rigid body modes (RBM). The POD-based deflation method was introduced in the article [2] by G. Diaz Cortes and suggested as an alternative to the standard options for deflation vectors which are expensive to compute or problem dependent. The POD-based deflation with incomplete Cholesky preconditioner yielded promising results for the test problem in the article. Moreover, the applicability of the method is not restricted to this specific test problem, but it can be used for any time-varying problem [2]. Besides this, deflation based on RBM, as well as its combination with POD-based deflation, will be treated. Namely, as written in [3], there exists a correlation between the amount of RBM of sub-bodies of materials and the amount of small eigenvalues of the stiffness matrix. Using RBM-based deflation removes those small eigenvalues and hence improves the convergence of PCG [3]. The parallel implementation of RBM-based deflated PCG has been shown to be quite competitive with smoothed aggregation AMG for a simulation of asphalt concrete [11]. For the mechanical test problems, the RBM are easily generated, making the RBM-based deflation a promising preconditioner. The deflation methods will be implemented in a two-level PCG setting according to a robust and reliable implementation variant.

The smoothers for multigrid and the second preconditioners for deflation methods are chosen to be direct or stationary iterative methods. This choice is a tradeoff between the amount of memory and the number of iterations needed, and the suitability for parallel computing. For instance, the stationary iterative method (damped) Jacobi performs well regarding the amount of memory needed and is well-parallelizable; however, this is at the cost of a higher number of iterations. The direct method incomplete Cholesky decomposition generally needs few iterations to converge, but has a higher memory cost and is not as easily parallelised [8]. In addition to their point-wise version, we will also use a block version for the test problems with block-structured coefficient matrices.

Moreover, not only the choice of smoother or second preconditioner is a trade-off, but this holds for the choice of preconditioner for PCG as well. Which solver is preferred depends on its performance for the specific problem it is applied to, whether it is suitable for parallel computing, the hardware available (the amount of memory and potential of parallelising) and available time. In this thesis we focus on sequential computing, comparing the solvers based on the amount of memory, number of iterations and computation time required for two test problems. Before the numerical analysis is performed, the upcoming chapters first provide a theoretical explanation of the methods that will be applied.

Chapter 4

Algebraic Multigrid

Multigrid (MG) methods improve upon the stationary iterative methods by using coarser grids. While the high frequency error components are solved quickly using stationary iterative methods, the smooth/low frequency error components can converge very slowly. This drawback is avoided in MG methods by using two complementary processes: smoothing and coarse-grid correction. Namely, it first applies a few smoothing steps in the form of applying some iterations of a stationary iterative method for example. The remaining smooth components can be transferred to a coarser grid without losing critical information. On this coarser grid the smooth components appear once more as high frequency components, hence can be solved quickly again. Then the coarse-grid solution is used to construct the solution on the fine grid. The steps of a two-grid method are given in Algorithm 1 [8].

Algorithm 1: Two-grid method for solving $Ax = b$, given initial guess x_0 [8].

```
• $\nu_1$ pre-smoothing steps          $x_0^s = \text{smoothed}(x_0, \nu_1)$
• Fine residual                        $r^f = b - Ax_0^s$
• Coarse residual                      $r^c = Rr^f$
• Coarse error                         $e^c = (A_c)^{-1} r^c$
• Fine error                           $e^f = Ie^c$
• Solution correction                  $x^f = x_0^s + e^f$
• $\nu_2$ post-smoothing steps         $x = \text{smoothed}\big(x^f, \nu_2\big)$
```

The various MG methods are defined by their choice of smoothing operators, coarsening strategy, interpolation operators and application strategy [4]. The application strategy entails the order in which the coarser grids are visited. The two-grid cycle described above simply goes from the fine grid to the next coarsest grid and immediately back to the fine grid; but more complex cycles are possible as well. By applying the two-grid cycle again at certain coarser grids, the well-known V-, W- and F-cycles can be constructed as illustrated in

Figure 4.1 [8]. If a V-cycle is used with ν_1 pre- and ν_2 post-smoothing steps, this is denoted V(ν_1, ν_2).

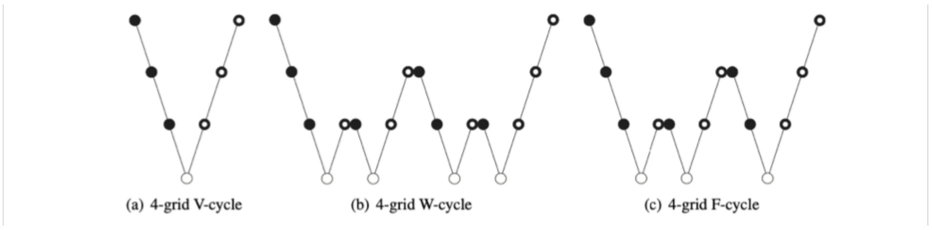


Figure 4.1: Three types of MG cycles for a 4-grid method. \ denotes restriction, / interpolation, • pre-smoothing, ∘ post-smoothing and ∘ the exact solution.
Figure from [8].

The MG class is divided into two subclasses: the geometric and the algebraic multigrid methods. In GMG methods, the coarser grids and the inter-grid operators between them are based on the physical properties of the grid [11]. These physical properties mostly determine the choice of coarse grid correction (i.e. the choice of inter- and coarse grid operators). There is more freedom in the choice of the smoothing operator, which should be chosen such that it removes the geometrically high frequency/oscillatory error components, as these are not removed by the coarse grid correction. Thus, the smoother has to adapt to the coarse grid correction [15].

On the other hand, in AMG methods the operators are constructed based on the matrix-entries, and are derived in a purely algebraic sense without explicit knowledge of the geometry [16]. This means that AMG methods can be used on unstructured grids, hence are the MG methods of choice in this paper. The coarser "grids" in AMG are not actual tangible grids but structures associated to the coarse grid operators. Contrary to GMG, in AMG the smoother is considered fixed and the coarse grid correction has to be adapted such that it exhibits complementary behaviour. The error components not removed by the smoother are the algebraically smooth components, which should then be removed by the coarse grid correction. Thus, in order to construct the coarse grid correction, the form of the algebraically smooth error components is needed. For a smoothing step with smoother M, define $S := \mathcal{I} - M^{-1}A$. Using the notation as in the pre-smoothing step of Algorithm 1, the smoothed vector is $x_0^s = Sx_0 + M^{-1}b$. The error after smoothing, e_0^s, relates to the error before smoothing, e_0, according to the following.

$$\begin{aligned} e_0^s &:= x_0^s - x_{true} && \text{where } x_{true} \text{ is the true solution of } Ax = b \\ &= Sx_0 + M^{-1}b - Sx_{true} - M^{-1}b && \text{since } Sx_{true} = x_{true} - M^{-1}b \\ &= Se_0 && \text{where } e_0 := x_0 - x_{sol} \end{aligned}$$

Components that are damped by smoothing are such that $Se \approx 0$, i.e. $e \approx M^{-1}Ae$. Algebraically smooth error components are those that are not damped by smoothing but stay approximately the same. Hence e is algebraically smooth if $Se \approx e$, i.e. if $M^{-1}Ae \approx 0$ [15]. In classical AMG, it is assumed that algebraically smooth error components have very small residuals. In that case, e is algebraically smooth when $r \approx 0$, and since $Ae = r$ this means that for smooth errors e we have

$Ae \approx 0$ [17]. This is why the set of algebraically smooth vectors is also called the near-kernel of A [18]. This knowledge about smooth vectors can be used in the construction of the coarse grid correction. The error operator corresponding to the coarse grid correction is $K_e := \mathcal{I} - I(RAI)^{-1}RA$ and is illustrated below based on Algorithm 1.

$$\begin{aligned}
x^f &= x_0^s + e^f \\
&= x_0^s + I(A_c)^{-1}R(b - Ax_0^s) && \text{where } A_c := RAI \\
&= K_e x_0^s + I(A_c)^{-1}Rb \\
e &:= x^f - x_{true} && \text{where } x_{true} \text{ is the true solution of } Ax = b \\
&= K_e x_0^s + I(A_c)^{-1}Rb - K_e x_{true} - I(A_c)^{-1}Rb && \text{since } x_{true} = K_e x_{true} + I(A_c)^{-1}Rb \\
&= K_e e_0^s && \text{where } e_0^s := x_0^s - x_{true}
\end{aligned}$$

The error of the solution computed by AMG (in this case the two-grid cycle without post-smoothing) is K_e multiplied with the error of the pre-smoothed initial guess; so K_e determines which error components are damped by the coarse grid correction. Since $(K_e)^2 = K_e$, this is a projection matrix hence has eigenvalues 0 and/or 1. The corresponding eigenspaces are

$$E_0^{K_e} := \{x \mid K_e x = 0\} = \{x \mid \exists y \text{ such that } x = Iy\}$$

and

$$E_1^{K_e} := \{x \mid K_e x = x\} = \{x \mid \exists y \text{ such that } x = (RA)^T y\}^\perp$$

where $^\perp$ denotes the orthogonal complement. So K_e removes the vectors in $E_0^{K_e}$, which is equal to the range of I. K_e should remove the algebraically smooth error components, hence the interpolation matrix I should be constructed such that these are contained in its range [15].

There are various AMG approaches for systems of partial differential equations (PDEs), like the classical AMG – also known as the Ruge-Stüben AMG (RS-AMG) –, smoothed aggregation AMG (SA-AMG) or adaptive smoothing and prolongation based AMG (aSP-AMG) [11]. The classical/RS-AMG method was developed for M-matrices[1]. The coarse grid nodes are selected based on the definition of strength of connection (SoC) which is the following: a node i is strongly connected with the nodes in the set $S_i := \{j \mid -A(i,j) \geq \theta \max_{k \neq i} -A(i,k)\}$, given a fixed parameter $0 < \theta < 1$. If A is an M-matrix, then $A(i,j) \leq 0$, hence the minus sign in S_i. The coarse grid C is then chosen to be a subset of the fine grid F such that each node in $F \backslash C$ is strongly connected to at least one coarse node, while also trying to avoid strong connections among coarse nodes themselves [15]. In the upcoming subsections, the SA-AMG and aSP-AMG are considered. For simplicity, the constructions of the coarse grid and inter-grid operators are given for the two-level case but can be extended to multiple levels in a recursive manner. Like mentioned in Chapter 3, the MG methods can be used as preconditioner to accelerate Krylov subspace methods. Each method is explained for solving $Ax = b$, but in the numerical experiments they will be applied as preconditioner. The AMG methods work then just as described below, but they are called at every iteration of PCG to perform the preconditioning step.

[1] B is an M-matrix if B is nonsingular, $B^{-1} \geq 0$, $B(i,i) > 0\ \forall i$, $B(i,j) \leq 0\ \forall i \neq j$ [7].

4.1 Smoothed Aggregation Algebraic Multigrid

Contrary to classical/RS-AMG method, where the coarse grid nodes are subsets of the fine grid nodes, this is not the case for aggregation AMG methods. In these, each coarse grid node originates from a number of fine grid nodes grouped together [19]. The smoothed aggregation AMG (SA-AMG) method combines aggregation with a smoother to construct the interpolation operator. Considering the problem $Ax = b$ where the $N \times N$ coefficient matrix is symmetric, SA-AMG works according to the following steps [14].

1. Constructing the graph
   SA-AMG constructs a graph G corresponding to matrix A. G consists of N vertices and has edges between all vertices i and j for which $A(i,j) \neq 0$ and $A(i,j)$ is not "too close to zero" as such to avoid using weak connections in the construction of the coarse grid [14]. This filtering of small entries is done by dropping entries that do not satisfy the condition in Equation 4.1 [20] – note that this is a different definition of strength than used in RS-AMG.

$$A(i,j)^2 > \theta^2 |A(i,i)||A(j,j)| \tag{4.1}$$

2. Constructing the coarse grid via aggregation
   In the coarsening step, the nodes of G are grouped together into aggregates, creating a coarser grid. The aggregates must not be too small as this leads to many aggregates hence a large coarse grid, resulting in high iteration costs. Nor must aggregates be too large as this leads to an interpolation operator closely resembling piecewise constant interpolation, resulting in poor convergence. Considering for example isotropic problems using a damped Jacobi smoother as interpolation smoother (see next item), the aggregates should preferably be uniformly shaped with a diameter[2] of length three. There exist multiple forms of aggregation; here the Basic Aggregation in Algorithm 2 will be considered. The more nodes ending up in the second phase of the algorithm, the less likely that the resulting aggregates will be uniform and have the desired specified diameter. Therefore, the roots in the first phase should be chosen such that as few nodes as possible end up in phase two. This means that the nodes in phase one should be "packed tightly", which can be done by choosing the next root to be close to the existing aggregates [14].

Algorithm 2: Basic Aggregation Procedure [14]

```
while there exists an unaggregated node not adjacent to an aggregate do
    Pick root node not adjacent to any existing aggregate
    Define new aggregate as root node plus all its neighbours
end
Sweep unaggregated nodes into existing aggregates or use them to form new aggregates
```

3. Constructing the inter-grid operators
   Having created a coarse grid, the interpolation I from the coarse to the fine grid has to be constructed. Then the restriction matrix is set to be the transpose of the interpolation, and

[2]The diameter of an aggregate is the maximum length of the shortest paths in the aggregate.

the coarse grid operator $A_c := I^T A I$ according to the Galerkin approach. First, a tentative interpolation matrix $\widetilde{I}$ is created according to Equation 4.2, where each row corresponds to a node of the fine grid and each column to an aggregate i.e. a node of the coarse grid.

$$\widetilde{I}(i,j) = \begin{cases} 1 & \text{if } i^{th} \text{ node is contained in } j^{th} \text{ aggregate,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

This tentative interpolator corresponds to piecewise constant interpolation, so each fine node value is based on a single coarse node value. To get a more robust method, the tentative interpolator is combined with a smoother to obtain the interpolation matrix via $I = \widetilde{S}\widetilde{I}$, where $\widetilde{S}$ denotes the interpolator smoother. Considering a damped Jacobi smoother yields the following interpolation operator [14].

$$I = \left(\mathcal{I} - \omega D^{-1} A\right) \widetilde{I} \tag{4.3}$$

For multidimensional problems, a modification could be made in the coarsening strategy. Namely, consider the graph corresponding to a block matrix modification of the coefficient matrix A instead of simply using A. Instead of treating each degree of freedom (dof) as a separate vertex, group the dofs per grid point – e.g. in the deformation domain, group the x-, y- and z-deformations of each node together – and in this way construct a block matrix. Then the coarser grid will contain all dofs per coarse node, instead of possibly losing a dof corresponding to a node while keeping another dof of the same node [14].

4.2 Adaptive Smoothing and Prolongation based Algebraic Multigrid

Assume we are solving $Ax = b$, where $A \in \mathbb{R}^{N \times N}$ is a sparse SPD matrix. The adaptive smoothing and prolongation based AMG (aSP-AMG) uses an affinity-based SoC definition. The "adaptive" term in the name refers to the adaptive and bootstrap AMG, since like in those methods no prior information about the near-kernel of A is mandatory, but an approximate near-kernel is constructed automatically. It is important for AMG methods to be able to accurately describe the near-kernel of A in order to construct the interpolation operator, as shown above. The steps of aSP-AMG are the following [21].

1. Construct the smoother.
2. Construct the test space X representing the algebraically smooth vectors.
3. Construct the affinity-based SoC matrix S_c based on X.
4. Construct the coarse grid C as a maximum independent set.
5. Construct the interpolation matrix I and use the Galerkin approach.

In the smoothing step $x = x + M^{-1}(b - Ax)$ the M matrix is chosen such that it can be factorised as $M^{-1} = \omega G^T G$. This is possible for e.g. damped Jacobi or the more advanced smoother aFSAI [21]. Then the smoothing step can be written as $x = S_\omega x + \omega G^T G b$, where $S_\omega := \mathcal{I} - \omega G^T G A$ and taking $\omega < \frac{2}{\lambda_{max}(G^T G A)}$ to ensure convergence. The following steps can exploit this factorisation of the smoother.

Next, aSP-AMG constructs a test space of limited size representing the algebraically smooth vectors of the coefficient matrix. In general, the smooth vectors can be estimated by solving the generalised eigenvalue problem $Av = \lambda M v$. Usually, a low accuracy solve already gives acceptable approximations [4]. The construction of the SoC matrix and the inter-grid operators all depend on the test space, so it is important that a good representation of the algebraically smooth vectors is obtained. For most preconditioners, the smooth vectors correspond to the small eigenvalues of $M^{-1}A$. Hence the eigenvectors of S_ω corresponding to the eigenvalues close to 1 are a consistent approximation. These eigenvectors can be approximated by for example the power method, Lanczos method or the Simultaneous Rayleigh Quotient minimisation by Conjugate Gradients (SRQCG) method [21]. However, the power method converges slowly in general. Moreover, if an approximation of (part of) the near-kernel of A is available a-priori, then this should be taken advantage of since computing X represents a large part of the set-up costs. For instance, for elasticity problems, the RBMs are a suitable initial guess. The Lanczos method allows for an initial guess, but only a single vector. The SRQCG method takes a matrix as initial guess. Also, its initial convergence is often faster than that of the Lanczos method. Therefore, SRQCG will be used to generate the basis of the test space (possibly using the power method to generate an initial guess if one is not available). SRQCG is based on minimising Rayleigh quotients[3] and does so over multiple linearly independent vectors simultaneously. New vectors are generated based on a linear combination of the current iterates and search directions, via a conjugate gradient technique. Moreover, it accelerates convergence by employing Ritz projections. Finally, the columns of the $N \times N_t$ matrix X are created, where N_t denotes the dimension of the test space, and these columns represent the basis of the test space. The SRQCG algorithm is given in Appendix A [4].

Instead of the SoC definition used in classical AMG, which assumes the matrix A to be an M-matrix, an affinity-based SoC definition is used, hence is wider applicable [4]. The affinity is based on the matrix X according to Equation 4.4, where x_i^T denotes the i^{th} row of X, and x_j the transposed j^{th} row of X. $S_c(i,j)$ represents the influence between the nodes i and j.

$$S_c = \frac{\left(x_i^T x_j\right)^2}{\left(x_i^T x_i\right)\left(x_j^T x_j\right)} \tag{4.4}$$

The S_c matrix is given the same sparsity pattern as A and hereafter is filtered [21]. Filtering is often done by dropping all entries below a given threshold. However, the affinity-based SoC definition causes the matrix entries to be close together in a small interval around one, making it hard to determine the threshold-value. Therefore, instead the average number of connections per node is required to be equal to a given integer parameter θ [4].

The construction of the coarse grid is then done like in classical AMG, only with a different

[3]Given a real symmetric square matrix B its Rayleigh quotient is $\rho(z) := \frac{z^T B z}{z^T z}$, where z is a nonzero vector [22].

SoC definition underlying S_c. This means that from the original set of fine nodes F, a subset C is selected which are the nodes of the coarse grid. The coarse nodes are the maximum independent set (MIS) of nodes on the adjacency graph associated to the filtered S_c [21].

Having constructed the coarse grid, the $N \times N_c$ interpolation matrix I has to be determined, where N_c is the number of coarse grid nodes. Assuming a C/F ordering (i.e. the nodes are ordered such that first all the coarse nodes are considered and then the nodes in $F \backslash C$) we get $I := \begin{bmatrix} \mathcal{I} \\ W \end{bmatrix}$, where the $N_f \times N_c$ matrix W contains the interpolation weights (N_f denoting the number of nodes in $F \backslash C$). W is constructed via the Dynamic Pattern Least Squared (DPLS) method (for its algorithm see Appendix A). DPLS is an iterative method which constructs the pattern of the interpolation operator dynamically during setup. The interpolation weights are computed such that the interpolation residual is minimised while preserving sparsity, which is done as follows. Take a fine node $i \in F \backslash C$ and set $\mathcal{C}_i := \{j \in C \mid \exists$ a path of strong connections between i and j shorter than $d_p\}$, where d_p is a given parameter, usually set to 1 or 2. $\mathcal{C}_i$ are the coarse nodes that are possibly used to interpolate the fine node i. From $\mathcal{C}_i$, k distinct nodes are iteratively selected and gathered in $\bar{\mathcal{C}}_i$, such that the linear combination of the corresponding rows in X gives the best approximation of x_i with respect to the Euclidean norm. The interpolation weights w_{ij} $j = 1, \ldots, k$ corresponding to node i are computed such that they minimize $||x_i - \sum_{j \in \bar{\mathcal{C}}_i} w_{ij} x_j||_2$. Using the stopping criterion relating to the condition number of $X_{c,i} := X(\bar{\mathcal{C}}_i, :)^T$, ill-conditioning is avoided which would cause too large interpolation weights. Namely, stop when $\text{cond}(X_{c,i}) > \kappa_p$, given a fixed parameter κ_p. To maintain sparsity – hence to avoid constraining the AMG efficiency – an additional parameter n_{max} is introduced, which is the maximum number of interpolation points per fine node. Repeat the above until each fine node in $F \backslash C$ has been treated and store the resulting weights in W to obtain the interpolation operator I [4, 21]. Finally, the Galerkin approach is used, so the restriction matrix is set to be the transpose of the interpolation and the coarse-grid matrix $A_c := I^T A I$ [21].

Chapter 5

Deflation

Deflation methods show similarities with the basis of MG methods, but use for example model order reduction (MOR) techniques instead of coarser grids. MOR techniques lessen the computational complexity. Deflation can be used to treat unwanted eigenvalues that deteriorate the convergence. Instead of solving $Ax = b$, solve the deflated system in Equation 5.1 for the deflated solution $\widehat{x}$. Then the solution x is determined via $x = Qb + P^T\widehat{x}$.

$$PA\widehat{x} = Pb \tag{5.1}$$

The matrices P and Q used, are defined in the Definition 1 for symmetric positive semi-definite (SPSD) coefficient matrices but can be generalised to non-SPSD matrices as well [6].

Definition 1. Let the coefficient matrix $A \in \mathbb{R}^{N\times N}$ be SPSD and have d zero eigenvalues, and let the deflation-subspace matrix $Z \in \mathbb{R}^{N\times k}$ with full rank and $k < N - d$ be given. Then the Galerkin matrix is defined as $E := Z^TAZ \in \mathbb{R}^{k\times k}$ where Z must be chosen such that E is invertible, the correction matrix is $Q := ZE^{-1}Z^T \in \mathbb{R}^{N\times N}$ and the deflation matrix is $P := \mathcal{I} - AQ \in \mathbb{R}^{N\times N}$ [6].

The columns of the deflation-subspace matrix Z are called the deflation/projection vectors and are such that E is nonsingular (which is the case if $\mathcal{N}(A) \not\subset \mathcal{R}(Z)$). For the deflation method to yield good results, the deflation subspace should contain most of the system's variability; this is usually problem-dependent. Eigenvectors can be used as deflation vectors and can be effective in reducing the effective condition number and are expected to accelerate the convergence. However, eigenvectors are often expensive to compute and dense. Meanwhile, the deflation-subspace matrix is wanted to be sparse yet give good approximations of the eigenvectors [2, 5, 6]. Therefore, we suggest below to use POD or RBM to construct the deflation-subspace matrix. But first some general information about the relation between deflation methods and MG is given, as well as the implementation of deflation methods as preconditioner for PCG.

To clarify the relation between deflation and MG methods, consider a two-grid cycle solving $Ax = b$. Take as interpolation operator I the deflation-subspace matrix Z, as restriction its transpose, and as coarse grid operator $A_c := I^TAI$. If also one preconditioning step is applied with as preconditioner the deflation matrix P, then the resulting MG is a deflation method.

So while the two might look quite different at first, deflation is actually a specific type of MG method.

Deflation methods can be used to accelerate iterative methods. Considering the iterative method PCG, its combination with deflation is a type of two-level PCG (2L-PCG) method. 2L-PCG methods are CG with a traditional first-level preconditioner – the Jacobi method for example – and a second-level preconditioner – like the deflation matrix P. There are various implementations of 2L-PCG methods concerning deflation, which can be expressed by defining $\mathcal{P}$ and $\mathcal{A}$ in Equation 5.2 solving the deflated system. The rest of this section is based on [5], an article in which multiple 2L-PCG implementations are investigated.

$$\mathcal{P}\mathcal{A}\widehat{x} = \mathscr{b} \tag{5.2}$$

The straightforward implementation of 2L-PCG is DEF1 (deflation variant 1), where $\mathcal{P} = M^{-1}$, $\mathcal{A} = PA$ and $\mathscr{b} = M^{-1}Pb$. Then Equation 5.2 gives the deflated solution, and $x = Qb + P^T\widehat{x}$. This results in a well-defined system which can be solved by PCG since PA is SPSD. While theoretically DEF1 seems like a good implementation, in practice it is sensitive to perturbations in the coarse grid solutions and the termination criterion being set too small. This is due to the following. DEF1 shifts unwanted eigenvalues (which are the extreme eigenvalues since removing those decreases the condition number) to zero. However, if roundoff errors occur or approximate coarse solves are used, the zero eigenvalues of DEF1 may become very small yet nonzero values, causing the condition number to grow significantly and deteriorating the convergence.

To avoid this sensitivity to perturbations in the coarse solves, DEF1 has been adapted to A-DEF1 (adapted deflation variant 1) in Equation 5.3, where DEF1 is additively combined with the correction matrix Q. Recalling the relation between MG methods and deflation, the A-DEF1 deflation implementation corresponds to a V$(0, 1)$-cycle.

$$\mathcal{P}_{A-DEF1} = M^{-1}P + Q \tag{5.3}$$

The additional Q term shifts unwanted eigenvalues to one instead of zero, as shown in Theorem 2 from [5]. Now small perturbations yield only small changes in eigenvalues and condition number, contrary to DEF1. The drawback is that $\mathcal{P}_{A-DEF1}$ is not symmetric (nor is it symmetric with respect to the inner product induced by A). For PCG to have guaranteed convergence, $\mathcal{P}$ in Equation 5.2 should be SPD. A-DEF1 does not have an SPD operator, nor can it be transformed or decomposed into one. Thus, convergence is not guaranteed.

Theorem 2. *Let the spectrum of DEF1 be given by*

$$\sigma(M^{-1}PA) = \{0, \ldots, 0, \lambda_{k+1}, \ldots, \lambda_n\}$$

such that $\lambda_{k+1} \leq \ldots \leq \lambda_n$. *Let the spectrum of A-DEF1 be given by*

$$\sigma(M^{-1}PA + QA) = \{1, \ldots, 1, \mu_{k+1}, \ldots, \mu_n\}$$

or the spectrum of A-DEF2 be given by

$$\sigma(P^TM^{-1}A + QA) = \{1, \ldots, 1, \mu_{k+1}, \ldots, \mu_n\}$$

such that $\mu_{k+1} \leq \ldots \leq \mu_n$. Then $\lambda_i = \mu_i$ for all $i = k+1, \ldots, n$.

This disadvantage of non-SPD of A-DEF1 gives reason to consider A-DEF2 (adapted deflation variant 2), which is defined in Equation 5.4. As a consequence of the theorem above which also considers A-DEF2, it is like A-DEF1 not sensitive to perturbations in coarse solves. In MG terms, A-DEF2 corresponds to a V$(1,0)$-cycle. While $\mathcal{P}_{A-DEF2}$ is not symmetric, A-DEF2 can be shown to be equivalent to a method based on a symmetric operator and that it is an appropriate preconditioner for CG. Moreover, A-DEF2 can deal with severe termination tolerances. Therefore, the robust A-DEF2 is a better implementation than DEF1 and A-DEF1 – based on theory, numerical analysis and computational cost[4] [5]. In this paper, 2L-PCG methods are implemented according to A-DEF2.

$$\mathcal{P}_{A-DEF2} = P^T M^{-1} + Q \tag{5.4}$$

The 2L-PCG method based on DEF1, A-DEF1 and A-DEF2 is given in Algorithm 3 with the matrices and vectors specified in Table 5.1.

Algorithm 3: General 2L-PCG algorithm for solving $Ax = b$.
Algorithm from [5]

```
Select arbitrary $x_{arbi}$ and set $x_{start}$, $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $x_{end}$ as in Table 5.1.
$x_0 := x_{start}$, $r_0 := b - Ax_0$
$z_0 := \mathcal{M}_1 r_0$, $p_0 := \mathcal{M}_2 z_0$
for $j$ = 0, 1, ..., until convergence do
    $w_j = \mathcal{M}_3 A p_j$
    $\alpha_j = \frac{r_j^T z_j}{p_j^T w_j}$
    $x_{j+1} = x_j + \alpha_j p_j$
    $r_{j+1} = r_j - \alpha_j w_j$
    $z_{j+1} = \mathcal{M}_1 r_{j+1}$
    $\beta_j = \frac{r_{j+1}^T z_{j+1}}{r_j^T z_j}$
    $p_{j+1} = \mathcal{M}_2 z_{j+1} + \beta_j p_j$
$x_{it} = x_{end}$
```

Method	$\boldsymbol{x_{start}}$	$\boldsymbol{\mathcal{M}_1}$	$\boldsymbol{\mathcal{M}_2}$	$\boldsymbol{\mathcal{M}_3}$	$\boldsymbol{x_{end}}$
PCG	x_{arbi}	M^{-1}	$\mathcal{I}$	$\mathcal{I}$	x_{j+1}
DEF1	x_{arbi}	M^{-1}	$\mathcal{I}$	P	$Qb + P^T x_{j+1}$
A-DEF1	x_{arbi}	$M^{-1}P + Q$	$\mathcal{I}$	$\mathcal{I}$	x_{j+1}
A-DEF2	$Qb + P^T x_{arbi}$	$P^T M^{-1} + Q$	$\mathcal{I}$	$\mathcal{I}$	x_{j+1}

Table 5.1: Choices for in Algorithm 3 for PCG, DEF1, A-DEF1 and A-DEF2 [5].

[4] [5] compared PREC, AD, DEF1, DEF2, A-DEF1, A-DEF2, BNN, R-BNN1 and R-BNN2 and found that A-DEF2 is the best and most robust method among them.

5.1 Proper Orthogonal Decomposition based Deflation

Standard options for deflation vectors are expensive to compute (like eigenvectors) or problem dependent (like subdomain vectors). Therefore, consider POD basis vectors as deflation vectors for time-dependent problems. POD is an MOR technique where a basis is constructed via a collection of snapshots. Snapshots are solutions of the system at certain points in time and should capture the dynamics of the system to be solved. They can be obtained via tactics like the recycling approach, moving window approach, and training phase approach. Here, the moving window approach is used [2].

POD constructs the orthonormal basis $\Psi = [\psi_1, \ldots, \psi_p] \in \mathbb{R}^{N \times p}$ based on the snapshots $\{y_i\}_{i=1}^m$ in the following way. The orthonormal basis vectors $\{\psi_j\}_{j=1}^p$ are the eigenvectors that correspond to the p largest eigenvalues of the data correlation matrix $R := \frac{1}{m} YY^T \in \mathbb{R}^{N \times N}$ [2]. In Matlab these can be computed efficiently via the $[U, \Sigma, V] = svd(Y, 0)$ command, $Y \in \mathbb{R}^{N \times m}$ with $m << N$, which results in an economy-size decomposition: only the first m columns of U are computed and Σ is $m \times m$. As $Y = U\Sigma V^T$ is the singular value decomposition, the U matrix (the left-singular vectors of Y) contains the eigenvectors of YY^T, hence of R [23]. In MOR methods, the model is then projected onto the space spanned by the columns of Ψ yielding a reduced order model. But if the POD method is used for deflation, the deflation-subspace matrix Z is set to consist of the POD basis vectors [2].

As mentioned, the snapshots are obtained in a moving window approach, where the snapshots are the m most recent solutions computed at the previous time steps. This means that for the first few time-step there is no deflation-subspace matrix yet hence the iterative method without deflation is applied [2]. Thereafter, the deflation-subspace could in theory be updated every subsequent time step. This can however be expensive. For this reason, also other updating strategies can be employed such as updating only when the number of iterations at the previous time-step exceeds a certain threshold τ. Another parameter μ can be added to ensure that the deflation-subspace is updated at least once every μ time steps.

5.2 Rigid Body Modes based Deflation

For mechanical problems, another option for deflation vectors is RBM. These are the translations and rotations of an unconstrained object such that there is no internal deformation [24]. In three dimensions, an object has in general six RBM, namely three translations and three rotations. The translation vectors for a three-dimensional object with m nodes ($\{x_1, y_1, z_1, x_2, y_2, z_2 \ldots, x_m, y_m, z_m\}^T$) and considering a single domain are uniform displacement in x-, y- and z-directions:

$$\begin{aligned}
\text{Translation in } x\text{-direction: } & \{1, 0, 0, 1, 0, 0, \ldots, 1, 0, 0\}^T \\
\text{Translation in } y\text{-direction: } & \{0, 1, 0, 0, 1, 0, \ldots, 0, 1, 0\}^T \\
\text{Translation in } z\text{-direction: } & \{0, 0, 1, 0, 0, 1, \ldots, 0, 0, 1\}^T
\end{aligned}$$

The three rotations are given below, where $r_j := \sqrt{x_j^2 + y_j^2 + z_j^2}$

Rotation in the x, y-plane:
$\theta_j = \tan^{-1}\left(\frac{y_j}{x_j}\right)$, $\phi_j = \cos^{-1}\left(\frac{z_j}{r_j}\right)$,

$$\begin{Bmatrix} -r_1 \sin(\theta_1)\sin(\phi_1) \\ r_1 \cos(\theta_1)\sin(\phi_1) \\ 0 \\ \vdots \\ -r_m \sin(\theta_m)\sin(\phi_m) \\ r_m \cos(\theta_m)\sin(\phi_m) \\ 0 \end{Bmatrix}$$

Rotation in the x, z-plane:
$\theta_j = \tan^{-1}\left(\frac{z_j}{x_j}\right)$, $\phi_j = \cos^{-1}\left(\frac{y_j}{r_j}\right)$,

$$\begin{Bmatrix} -r_1 \sin(\theta_1)\sin(\phi_1) \\ 0 \\ r_1 \cos(\theta_1)\sin(\phi_1) \\ \vdots \\ -r_m \sin(\theta_m)\sin(\phi_m) \\ 0 \\ r_m \cos(\theta_m)\sin(\phi_m) \end{Bmatrix}$$

Rotation in the y, z-plane:
$\theta_j = \tan^{-1}\left(\frac{y_j}{z_j}\right)$, $\phi_j = \cos^{-1}\left(\frac{x_j}{r_j}\right)$,

$$\begin{Bmatrix} 0 \\ r_1 \cos(\theta_1)\sin(\phi_1) \\ -r_1 \sin(\theta_1)\sin(\phi_1) \\ \vdots \\ 0 \\ r_m \cos(\theta_m)\sin(\phi_m) \\ -r_m \sin(\theta_m)\sin(\phi_m) \end{Bmatrix}$$

These vectors make up the columns of the deflation-subspace matrix Z. Instead of using a single domain, multiple domains can be used as well. In the case of using k subdomains, then each domain has six RBM, constructed for each domain by only assigning the appropriate values for translation/rotation to the nodes corresponding to that specific domain. The size of the resulting Z is then $N \times 6k$ [3].

Chapter 6

Block Preconditioners

If the coefficient matrix A has a block structure, block preconditioners can be useful. This is the case for the mechanical part of the thermo-mechanical model and for the wafer-slip model. For these cases, each node has a dof associated to the x-, y- and z-direction, corresponding to 3×3 blocks. A simple block preconditioner is the block Jacobi method. Instead of using the diagonal of A, use the block diagonal consisting of all the blocks on the diagonal of A.

Another option is the incomplete block Cholesky decomposition, which exploits the block structure of A when constructing the decomposition. The block algorithm used in this paper is based on the algorithms from [25, 26]. First, the matrix A is reduced to a smaller matrix Ap where the $n_b \times n_b$ blocks regarding the dofs of a single node are reduced to a single point – for the models considered here $n_b = 3$. The fill-in nonzero pattern P' is determined via Algorithm 4 for Ap (see Figure 6.1 where $n_b = 4$)[5]. Then, incomplete block Cholesky is performed for A with fill-in nonzero pattern P', according to Algorithm 5 (where $X(i,j)$ denotes the $(i,j)^{\text{th}}$ entry of matrix X, and $X_{i,j}$ the $(i,j)^{\text{th}}$ block of X).

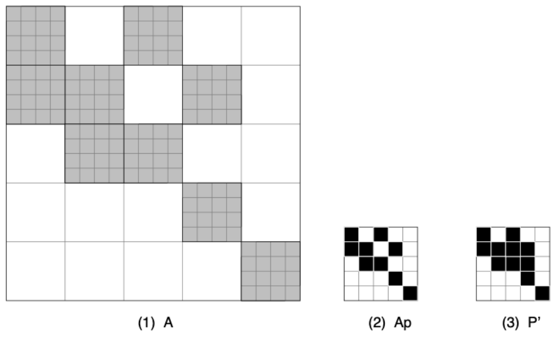


Figure 6.1: Figure from [25] illustrating the block matrix A, its abstraction Ap and its fill-in nonzero pattern P'.

[5]Note that this fill-in is different from Matlab's *ichol* function, where a drop-tolerance is used.

Algorithm 4: Computation of fill-in nonzero pattern [25].
Input: matrix Ap, n = length of Ap, k = level of fill-in.
Output: nonzero pattern P' with fill-in level k.

```
$P$ = nonzero pattern of $Ap$
$L(i,j) = \begin{cases} \infty \text{ if } P(i,j) = 0 \\ 0 \text{ otherwise} \end{cases}$
$P' = ones(n,n)$
for $i = 2 : n$ do
    for $p = 1 : i-1$ do
        if $L(i,p) \leq k$ then
            for $j = p+1 : n$ do
                $L(i,j) = min\{L(i,j), L(i,p) + L(p,j) + 1\}$
    for $j = 1 : n$ do
        if $L(i,j) > k$ then
            $P'(i,j) = 0$
```

Algorithm 5: Incomplete block Cholesky [26].
Input: matrix A, fill-in nonzero pattern P', block size n_b.
Output: upper triangular incomplete block Cholesky factor R of A.

```
$n = \frac{length(A)}{n_b}$
for $k = 1 : n-1$ do
    for $j = k : n$ do
        if $P'(k,j) \neq 0$ then
            $T = A_{k,j} - \sum_{s=1}^{k-1} R_{s,k}^T R_{s,j}$
            if $k = j$ then
                $R_{k,k}^T = ichol(T)$
            else
                $R_{k,j} = R_{k,k}^{-T} T$
$R_{n,n} = ichol(A_{n,n} - \sum_{s=1}^{n-1} R_{s,n}^T R_{s,n})$
```

The determination of the blocks in the construction of block preconditioners has to be done carefully. In both the thermo-mechanical and wafer-slip model, a number of dofs have been constrained. In the system equations this is translated by removing the rows and columns corresponding to those dofs from the coefficient matrix A. Then simply taking 3×3 blocks is not guaranteed to group the dofs of a single node together, since for some nodes one or multiple dofs may have been removed. In order to avoid this, at the location of the removed rows and columns the corresponding rows and columns of the identity matrix are placed. For the corresponding entries of the right-hand side vector, zeros can be added. Then using 3×3 blocks does group the dofs per node. Moreover, if the matrix is to be reordered to reduce the amount of fill-in, this reordering is done based on its abstraction Ap to ensure that the dofs corresponding to the same node stay together.

Chapter 7

Analysis of Iterative Methods for the Thermo-Mechanical Problem

In the upcoming chapters, the methods discussed are applied to the test problems, where sequential computing is practiced. The LDL decomposition (a variant of the Cholesky decomposition where $A = LDL^T$, with L lower unit triangular and D diagonal) is used to perform the coarse grid solves for AMG methods and to compute the inverse of the Galerkin matrix E for deflation methods. The PCG method is considered converged when the relative residual norm reaches a predetermined accuracy. In this chapter a small-size test problem representing the thermo-mechanical model is considered, first its thermal and then its mechanical part. The AMG and RBM-based deflation methods are compared based on the steady-state computation. Since the POD-based deflation method is suggested for time-dependent problems, this preconditioner is applied for the transient computation. In the next chapter, the waver-slip model is treated. The number of iterations required for convergence, the time to solve the problem and the data-size of the methods are given. The data-size is the amount of memory needed to store all the required pre-computed data in Matlab, which contains the inter-grid operators, coarse grid operators, deflation subspaces, smoothers and preconditioners, but excluding the original coefficient matrix A (since this matrix has to be stored for the problem definition, and is used by every method and has the same amount of memory for each method). For efficiently storing the matrices, the incomplete Cholesky decomposition matrix L is stored by taking the transpose of L^T since Matlab then requires fewer bytes to store the matrix.

7.1 Thermal Part

The test model represents a thermo-mechanical box. A surface heat is applied to the top surface of the box, and a convection boundary condition is set at the bottom surface. The box is made of iron and has density = 7874 kg/m^3, Youngs modulus = $211 \cdot 10^9$ Pa, Poisson ratio = 0.29, heat capacity = 450 J/kg/K, thermal conductivity = 80.4 W/m/K and thermal strain $\epsilon(T) = (T - 22) \cdot 10^{-6}$. The box consists of 3077 nodes. The thermal part of the model is described by Equation 7.1, where $T \in \mathbb{R}^{3077}$ is the temperature, E_T, A_T, B_T are sparse system matrices in $\mathbb{R}^{3077 \times 3077}$, and w the input signal in $\mathbb{R}^{3077}$.

$$E_T \dot{T} = A_T T + B_T w \tag{7.1}$$

From this model the steady-state temperature can be found by solving

$$-A_T T_\infty = B_T w \tag{7.2}$$

In order to solve the continuous-time model in Equation 7.1 for times other than the steady-state, it is discretised. This is done via the implicit Euler method, resulting in the discrete-time system in Equation 7.3, where Δt is the specified size of the time steps.

$$(E_T - \Delta t A_T) T_{k+1} = E_T T_k + \Delta t B_T w_k \tag{7.3}$$

The AMG methods are used to solve the steady-state temperature. The RS-, SA- and aSP-AMG methods are compared based on the required number of iterations, time, and data-size. After this, the POD-based deflation method is applied – via the A-DEF2 implementation – to determine the transient temperature.

7.1.1 Algebraic Multigrid

The three AMG methods considered in Chapter 4 are applied for the steady-state temperature of the thermo-mechanical test model, denoted $Ax = b$ where $A := -A_T$, x denotes the unknown steady-state temperature T_∞ and $b := B_T w$ according to Equation 7.2. The parameter settings for RS-AM are $\theta = 0.8$, for SA-AMG $\theta = 10^{-8}$ and $\omega = 2/3$ for the interpolation smoother, and for aSP-AMG $\theta = 100$, $N_{it} = 5$, $N_t = 100$, $d_p = 2$, $\kappa_p = 10^8$, $n_{max} = 10$. All three methods use one pre- and one post-smoothing step. Using multiple smoothing steps would in general reduce the number of iterations but increase the computation time for this thermo-mechanical problem. In Table 7.1 the methods are applied as preconditioners for PCG with as smoothers damped Jacobi, SSOR and incomplete Cholesky. iChol1 denotes incomplete Cholesky with zero-fill, i.e. incomplete Cholesky with the same nonzero pattern as A. iChol2 denotes incomplete Cholesky with threshold dropping using a threshold of 10^{-4} and with diagonal shift 0.001. The threshold dropping entails that non-diagonal elements with magnitude smaller than the local drop tolerance (at step k of the factorisation this is $10^{-4}||A(k : end, k)||_1$) are dropped from the factorisation. The diagonal shift means that instead of factorising A, the incomplete Cholesky is computed for $A + 0.001D$ (where D denotes the diagonal of A) to avoid encountering nonpositive pivots [27]. iChol3 is the same as iChol2 except that reordering is performed according to Matlab's *dissect*, which reorders the matrix to reduce the amount of fill-in [28]. The methods are applied for 2 and 4 levels (also counting the original fine grid), visited according to a V-cycle.

In the table we see that RS-AMG requires the least iterations, but SA-AMG has significantly smaller coarse grid sizes and and smaller data-sizes. The time needed to construct the operators is generally smallest for SA-AMG. The time required for preparing the solver is smallest for SA-AMG as well, though the difference with RS-AMG is very small. The solve time of SA-AMG is mostly larger than for RS-AMG, but again the difference is small. The total time is for most cases smaller for SA-AMG than for RS-AMG. aSP-AMG is considered with a test space constructed via SRQCG and via eigenvectors. In both cases it requires more iterations, takes longer and has a larger data-size than the other two methods.

Comparing the smoothers used in Table 7.1, damped Jacobi has the smallest data-size, but needs the most iterations. On the other hand, incomplete Cholesky preconditioners need fewer iterations but have larger data-sizes. iChol3 needs the least iterations, and its data-size is smaller than that of iChol2 thanks to the reordering. The SSOR smoother has the third largest data-size and second most number of iterations. Thus, when memory is very limited, the damped Jacobi smoother can be used; otherwise iChol3 is preferred.

Precon-ditioner	**Smoother**	**Levels**	**Size coarse**	**# it.s**	t_{op}	t_{prep}	t_{solve}	t_{tot}	**Data-size (MB)**
RS-AMG	damped Jacobi	2	867	30	1.57	0.03	0.10	1.69	3.96
RS-AMG	damped Jacobi	4	867, 803, 663	33	3.70	0.02	0.11	3.83	4.96
RS-AMG	iChol1	2	867	5	1.58	0.01	0.06	1.65	6.07
RS-AMG	iChol1	4	867, 803, 663	5	3.76	0.01	0.08	3.85	7.59
RS-AMG	iChol2	2	867	3	1.76	0.08	0.04	1.88	12.1
RS-AMG	iChol2	4	867, 803, 663	3	4.91	0.12	0.04	5.08	18
RS-AMG	iChol3	2	867	3	1.42	0.09	0.02	1.54	10.8
RS-AMG	iChol3	4	867, 803, 663	3	3.28	0.10	0.02	3.40	13.7
RS-AMG	SSOR	2	867	23	1.57	0.04	0.11	1.71	6.19
RS-AMG	SSOR	4	867, 803, 663	25	3.74	0.05	0.11	3.90	7.77
SA-AMG	damped Jacobi	2	50	79	1.38	0.01	0.11	1.50	2.84
SA-AMG	damped Jacobi	4	50, 4, 1	113	1.52	0.01	0.17	1.70	2.84
SA-AMG	iChol1	2	50	13	1.46	0.01	0.05	1.51	4.95
SA-AMG	iChol1	4	50, 4, 1	13	1.66	0.02	0.06	1.74	4.96
SA-AMG	iChol2	2	50	5	1.71	0.04	0.04	1.79	10.9
SA-AMG	iChol2	4	50, 4, 1	5	1.53	0.04	0.04	1.61	11
SA-AMG	iChol3	2	50	3	1.34	0.06	0.02	1.42	9.64
SA-AMG	iChol3	4	50, 4, 1	3	1.39	0.06	0.02	1.46	9.66
SA-AMG	SSOR	2	50	63	1.81	0.04	0.16	2.02	5.06
SA-AMG	SSOR	4	50, 4, 1	85	1.54	0.02	0.22	1.78	5.08
aSP-AMG (SRQCG)	damped Jacobi	2	515	129	30.01	–[6]	0.22	30.23	4.19
aSP-AMG (eigenvectors)	damped Jacobi	2	592	111	12.84	–[6]	0.31	13.15	4.86

Table 7.1: Computation of steady-state temperature with accuracy 10^{-8}, via RS-AMG, SA-AMG and aSP-AMG as preconditioner for PCG. The coefficient matrix has size 3077×3077. For damped Jacobi and SSOR smoothers $\omega = 0.1$. # it.s denotes the number of iterations, t_{op} the time required to construct the operators, t_{prep} the time to prepare the solver, t_{solve} the solve time, and t_{tot} the total time. The times are all given in seconds.

[6]For aSP-AMG, time t_{prep} is contained in t_{op}.

In Table 7.2, we take a closer look at aSP-AMG. For each choice of test space, the ideal interpolation via $W_{ideal} := -A_{FF}^{-1}A_{FC}$ gives the lowest number of iterations, as expected. Unexpectedly, a random W needs fewer or equal iterations as when W is computed by DPLS. Moreover, if using W_{ideal}, the random test space yields a smaller coarse grid and needs fewer iterations than the test space constructed via SRQCG. This seems to indicate that there might be a problem in the construction of the test space X via SRQCG. Therefore, aSP-AMG based on eigenvectors was also included in the table above.

X (test space)	**W (interpolation)**	**# iterations**	**Size coarse**
SRQCG (X_0 via power method)	DPLS	129	515
SRQCG (X_0 via power method)	W_{ideal}	55	515
SRQCG (X_0 via power method)	random	114	515
Eigenvectors of smallest eigenvalues	DPLS	111	592
Eigenvectors of smallest eigenvalues	W_{ideal}	47	592
Eigenvectors of smallest eigenvalues	random	111	592
Random	DPLS	130	482
Random	W_{ideal}	50	482
Random	random	114	482

Table 7.2: Computation of steady-state temperature with accuracy 10^{-8} via aSP-AMG preconditioner with damped Jacobi smoother with $\omega = 0.1$. The coefficient matrix has size 3077×3077.

7.1.2 Deflation

The POD-based deflation method from Chapter 5 is applied via 2L-PCG (implemented as A-DEF2) for the transient temperature, with the incomplete Cholesky with zero-fill (iChol1) as preconditioner. The transient temperature is computed for time $t_{start} = 0$ until time $t_{end} = 1800$ seconds, with steps of 2 seconds. The size of the deflation-subspace matrix Z is set to 14, and is computed in a moving horizon fashion. Z is updated when the number of iterations exceeds $\tau := 9$, and is updated at least once every $\mu := 10$ time-steps. The result is given in Figure 7.1a. The figure shows that the number of iterations for POD-based deflation needs, after a few initial time steps, only a single iteration to solve the system. To compare with PCG with incomplete Cholesky preconditioner but without any deflation, see Figure 7.1b. As was done with POD-based deflation, the result of the previous time step is used as initial guess for the following time step. Now the number of iterations is at least 6, so using POD-based deflation lowers the number of iterations compared to PCG. Moreover, the total simulation time decreases by approximately 43% when adding POD-based deflation.

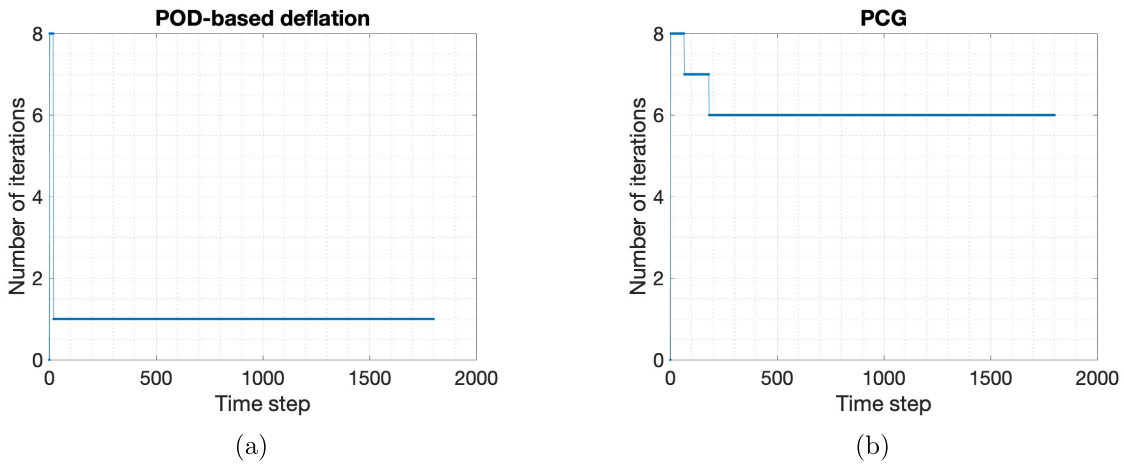


Figure 7.1: Number of iterations required for convergence with accuracy 10^{-10} per time step, where the transient temperature is computed by POD-based deflation in 2L-PCG (7.1a) or by PCG (7.1b), both with incomplete Cholesky preconditioner.

7.2 Mechanical Part

The mechanical part of the test model is described by Equation 7.4, where the thermal forces vector $f_T := L_0 + L_1 T$ links the thermal part to the mechanical part of the model. Each of the 3077 nodes can be deformed in the x-, y- and z-direction, resulting in a system of size $3 \cdot 3077 = 9231$. However, the box is mechanically constrained in its center at six dofs, hence the unconstrained deformation $d_f \in \mathbb{R}^{9225}$, and the corresponding stiffness matrix $K_{ff} \in \mathbb{R}^{9225\times 9225}$ and right-hand side $f_{T,f} \in \mathbb{R}^{9225}$.

$$K_{ff} d_f = f_{T,f} \tag{7.4}$$

7.2.1 Algebraic Multigrid

The AMG methods are used to compute the steady-state deformation, $Ax = b$, where $A := K_{ff}$, x denotes the unknown deformation d_f and $b := f_{T_\infty,f}$ according to Equation 7.4. The parameter settings are the same as for the steady-state temperature, except that for aSP-AMG now $N_t = 10$. In Table 7.3 the results are shown. As was the case for the thermal model, the RS-AMG method requires the least iterations, while SA-AMG yields the smallest coarse grids. The total computation time for aSP-AMG is much larger than for the other two methods. Moreover, SA-AMG and aSP-AMG are close in number of iterations for the damped Jacobi smoother and using 2 levels, but the coarse grid of aSP-AMG is much larger than for SA-AMG. So SA-AMG is preferred over aSP-AMG. The time constructing the operators and the solve time are larger for SA-AMG than for RS-AMG, while the time preparing the solver is smaller for SA-AMG. The total time is larger for SA-AMG, but its data-size is smaller than for RS-AMG. If wanting to solve the problem for multiple right-hand sides, then the operators have to be computed only once, hence then the most important time factor is the solve time, which is generally smallest for RS-AMG. So in these cases RS-AMG is preferred over SA-AMG, but only if enough memory is available. If the amount of memory is limited, then SA-AMG should be used instead.

Comparing the smoothers used in the table, damped Jacobi has again the smallest data-size, but does not yield the largest number of iterations. Instead, iChol1 requires the most iterations and also has a larger data-size. SSOR requires similar amounts of data, but fewer iterations. iChol2 significantly reduces the number of iterations compared to iChol1, but at the cost of approximately doubling the data-size. The reordering in iChol3 reduces the number of iterations even further and decreases the data-size compared to iChol2.

Precon-ditioner	**Smoother**	**Levels**	**Size coarse**	**# it.s**	t_{op}	t_{prep}	t_{solve}	t_{tot}	**Data-size (MB)**
RS-AMG	damped Jacobi	2	3916	83	2.54	0.32	0.67	3.53	49.5
RS-AMG	damped Jacobi	4	3916, 1705, 790	218	5.77	0.29	2.18	8.24	43.3
RS-AMG	iChol1	2	3916	627	2.66	0.37	7.85	10.88	64.4
RS-AMG	iChol1	4	3916, 1705, 790	733	5.67	0.17	12.03	17.87	68.6
RS-AMG	iChol2	2	3916	18	2.48	0.78	0.45	3.72	112
RS-AMG	iChol2	4	3916, 1705, 790	19	5.20	1.18	0.68	7.06	155
RS-AMG	iChol3	2	3916	14	2.57	0.97	0.32	3.86	101
RS-AMG	iChol3	4	3916, 1705, 790	14	5.32	1.08	0.42	6.82	119
RS-AMG	SSOR	2	3916	66	2.7	0.42	1.10	4.24	64.8
RS-AMG	SSOR	4	3916, 1705, 790	169	5.14	0.22	3.06	8.42	69.3
SA-AMG	damped Jacobi	2	50	833	19.32	0.08	4.80	24.20	16.5
SA-AMG	damped Jacobi	4	50, 4, 1	900	18.38	0.05	6.43	24.86	16.5
SA-AMG	iChol1	2	50	2454	25.55	0.11	25.55	51.21	31.5
SA-AMG	iChol1	4	50, 4, 1	2459	21.96	0.12	30.51	52.59	31.5
SA-AMG	iChol2	2	50	24	23.62	0.72	0.67	25.01	79.2
SA-AMG	iChol2	4	50, 4, 1	24	25.55	0.67	0.69	26.91	79.2
SA-AMG	iChol3	2	50	17	19.74	0.78	0.34	20.86	67.7
SA-AMG	iChol3	4	50, 4, 1	17	21.71	0.79	0.39	22.89	67.7
SA-AMG	SSOR	2	50	624	25.00	0.11	7.42	32.53	31.8
SA-AMG	SSOR	4	50, 4, 1	666	21.95	0.15	8.11	30.21	31.8
aSP-AMG (SRQCG)	damped Jacobi	2	855	883	72.11	– [7]	5.67	77.79	20.5
aSP-AMG (eigenvectors)	damped Jacobi	2	759	880	65.16	– [7]	5.29	70.45	19.6

Table 7.3: Computation of steady-state deformation with accuracy 10^{-8}, via RS-AMG, SA-AMG and aSP-AMG as preconditioner for PCG. The coefficient matrix has size 9225×9225. For damped Jacobi and SSOR $\omega = 0.1$. # it.s denotes the number of iterations, t_{op} the time required to construct the operators, t_{prep} the time to prepare the solver, t_{solve} the solve time, and t_{tot} the total time. The times are all given in seconds.

[7]For aSP-AMG, time t_{prep} is contained in t_{op}.

Due to the rapid coarsening of SA-AMG, it has a larger total computation time. A modification of SA-AMG for multidimensional systems was suggested in Section 4.1, by grouping the dofs of each node together. So the deformation in the x-, y- and z-directions are grouped together for the construction of the aggregates, to avoid losing a deformation in a certain direction of a coarse node. The results are shown in Table 7.4. The sizes of the coarser grids are the same for the grouped and non-grouped SA-AMG. For the grouped version, the time to generate the operators, to prepare the solver and the total time are in most cases smaller than for the non-grouped version. But the solve time is mostly smaller for the non-grouped SA-AMG. Moreover, the grouped SA-AMG requires a lower or equal number iterations for all smoothers and levels considered – except the iChol1 smoother on two levels – and has a smaller data-size.

Grouped	**Smoother**	**Levels**	**Size coarse**	**# it.s**	t_{op}	t_{prep}	t_{solve}	t_{tot}	**Data-size (MB)**
yes	damped Jacobi	2	50	827	17.17	0.06	5.77	23.00	16.5
yes	damped Jacobi	3	50, 4	888	18.31	0.09	6.91	25.30	16.5
yes	iChol1	2	50	2557	18.06	0.11	25.48	43.66	31.4
yes	iChol1	3	50, 4	2375	17.57	0.10	25.55	43.22	31.5
yes	iChol2	2	50	24	18.04	0.65	0.91	19.61	79.1
yes	iChol2	3	50, 4	24	17.40	0.63	0.91	18.93	79.1
no	damped Jacobi	2	50	833	19.32	0.08	4.80	24.20	16.5
no	damped Jacobi	3	50, 4	899	18.41	0.05	5.23	23.69	16.5
no	iChol1	2	50	2454	25.33	0.13	25.56	51.02	31.5
no	iChol1	3	50, 4	2794	22.29	0.14	31.72	54.16	31.5
no	iChol2	2	50	24	22.65	0.67	0.70	24.01	79.2
no	iChol2	3	50, 4	24	23.72	0.68	0.70	25.10	79.2

Table 7.4: Computation of steady-state deformation with accuracy 10^{-8} via SA-AMG as preconditioner for PCG. The coefficient matrix has size 9225×9225. For damped Jacobi $\omega = 0.1$. # it.s denotes the number of iterations, t_{op} the time required to construct the operators, t_{prep} the time to prepare the solver, t_{solve} the solve time, and t_{tot} the total time. The times are all given in seconds.

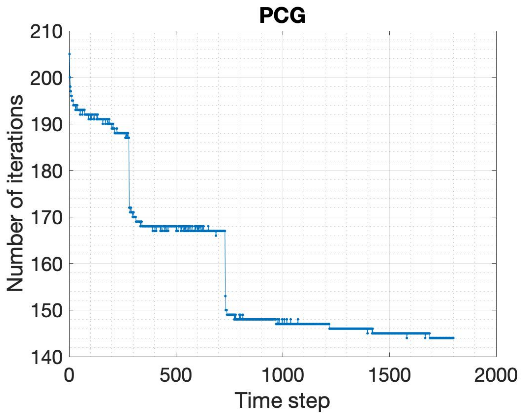
7.2.2 Deflation

Deflation methods based on RBM are used to compute the transient deformation as well as the steady state deformation. First, the transient deformation is treated, by considering both POD- and RBM-based deflation (where the RBM are based on a single domain). Then a nonlinear version of the transient deformation problem is solved. Thereafter, the steady-state deformation is computed via RBM-based deflation with various preconditioners and multiple domains.

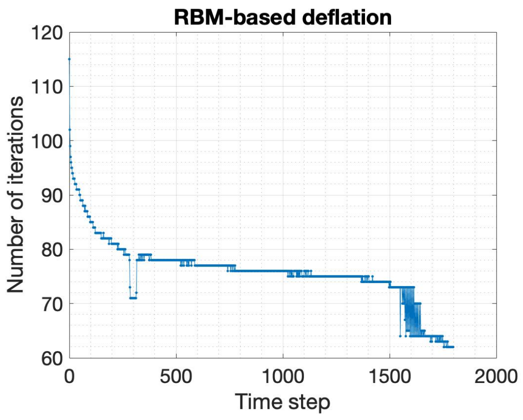
Transient Deformation

To compute the transient deformation, take the same time-interval as for the computation of the transient temperature T above and use this T for the right-hand side in Equation 7.4. The POD- and RBM-based deflation using a single domain, as well as their combination are considered, using incomplete Cholesky with zero-fill (iChol1) as preconditioner in 2L-PCG.

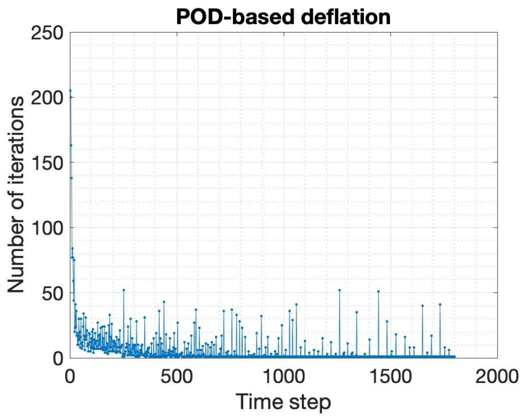
In Figure 7.2a, PCG with incomplete Cholesky preconditioner is applied to compute the transient deformation. The number of iterations decreases as time increases, but stays above 140 iterations. If using RBM-based deflation, Figure 7.2b shows that the number of iterations compared to PCG decreases, and over time becomes even less than 80. Moreover, the total simulation time is decreases by approximately 36%. For POD-based deflation, again the size of the deflation-subspace matrix Z is set to 14, $\tau := 9$, and $\mu := 10$. The POD-based deflation decreases the number of iterations per time step significantly, compared to both PCG and RBM-based deflation, as seen in Figure 7.2c. Compared to PCG, the total simulation time decreases by approximately 91% when adding POD-based deflation.



(a)



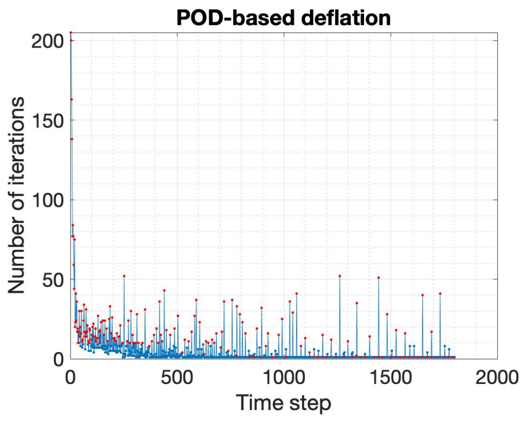
(b)



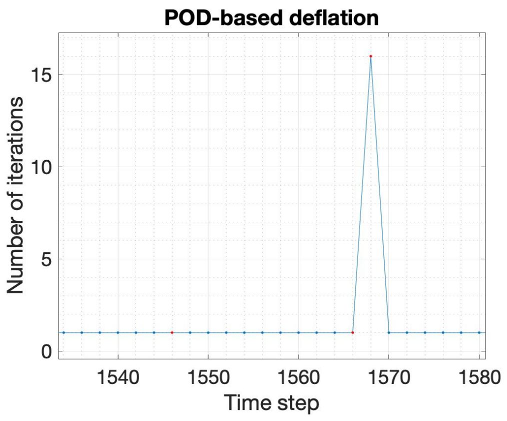
(c)

Figure 7.2: Number of iterations required for convergence with accuracy 10^{-8} per time step where the transient deformation is computed by PCG with incomplete Cholesky preconditioner (7.2a), and by 2L-PCG using incomplete Cholesky preconditioner and RBM-based deflation (7.2b) or POD-based deflation (7.2c).

Taking a closer look at the POD-based deflation, Figure 7.3a shows also when the deflation subspace is updated. Sometimes, e.g. at time 1566 seconds (Figure 7.3b), it happens that the subspace is updated due to the number of time steps exceeding μ and the number of iterations increases after this update. Then the next number of iterations exceeds threshold τ, the subspace is updated once more and the number of iterations decreases again. So it can happen that μ causes an increase in the number of iterations. However, not having this parameter would allow the number of iterations to increase as long as it stays below τ. If changing the value for μ such that it does not cause any updates in our simulation, while keeping the other parameters the same, the total number of iterations increases from 5127 to 5373; showing that μ is indeed useful to lower the number of iterations.



(a) Full time interval



(b) Zoomed in at time interval $[1540, 1580]$.

Figure 7.3: Number of iterations required for convergence with accuracy 10^{-8} per time step where the transient deformation is computed by 2L-PCG using incomplete Cholesky preconditioner and POD-based deflation. The red dots indicate that the deflation subspace is updated.

Another option is combining POD- and RBM-based deflation, by computing the POD deflation-subspace matrix as usual and adding the six RBM to it. Then the number of iterations decreases even further. Where the total number of iterations over all time steps for POD-based deflation was 5127, the combination with RBM needs only 2439 iterations. Thus, by only adding the six RBM the total number of iterations is more than halved. Moreover, the total simulation time decreases by approximately 94% compared to PCG, which is about 3% more than when using only POD.

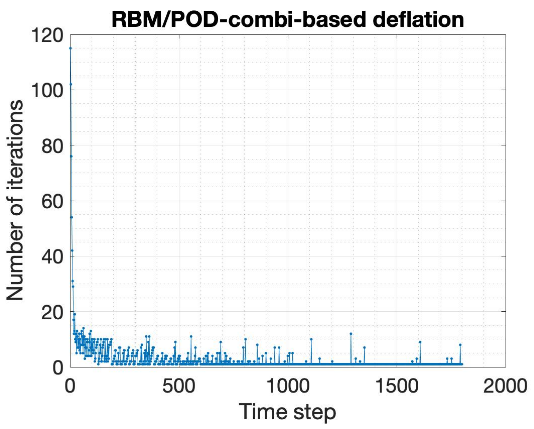


Figure 7.4: Number of iterations required for convergence with accuracy 10^{-8} per time step where the transient deformation is computed by 2L-PCG using incomplete Cholesky preconditioner and the combination of POD- and RBM-based deflation.

Nonlinear Problem

The temperature-strain $f_{T,f}$ can also be taken nonlinear, resulting in the nonlinear mechanical test problem. In order to make sure that the nonlinearity is clearly expressed, the right-hand side $B_T w$ of the thermal model is also changed. $B_T w := 2B_T w$ such that the difference in temperature is larger in the simulations (the difference was only 0.5, now it is almost 20). The nonlinear temperature-strain is set to $f_{T,f} := L_0 + L_1 T + \alpha_1 L_1 T^2 + \alpha_2 L_1 T^4$ and its coefficients α_1 and α_2 are chosen such that its derivative with respect to T is zero at some given temperature T^*.

$$\begin{aligned}
f_{T,f} &:= L_0 + L_1 T + \alpha_1 L_1 T^2 + \alpha_2 L_1 T^4 \\
&= L_1 \left(v + T + \alpha_1 T^2 + \alpha_2 T^4\right) && \text{Since } L_0 = L_1 v \text{ where } v := -22 \cdot ones(3077, 1) \\
f'_{T,f} &= 0 \text{ at } T = T^* \quad \Leftrightarrow && 0 = L_1 \left(I + 2\alpha_1 T^* + 4\alpha_2 {T^*}^3\right) \\
& && 0 = 1 + 2\alpha_1 T^* + 4\alpha_2 {T^*}^3
\end{aligned}$$

$$\text{Set } \alpha_2 = 1 \quad \Rightarrow \quad \alpha_1 = \frac{1 + 4{T^*}^3}{-2T^*}$$

The thermal model has $T_{min} = 21.8$ and $T_{max} = 40.4$. Taking $T^* = 30$, the scalar form of the nonlinearity in $f_{T,f}$ excluding L_1 is shown in Figure 7.5.

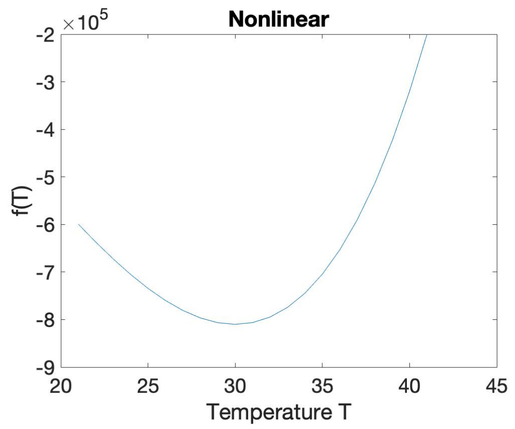
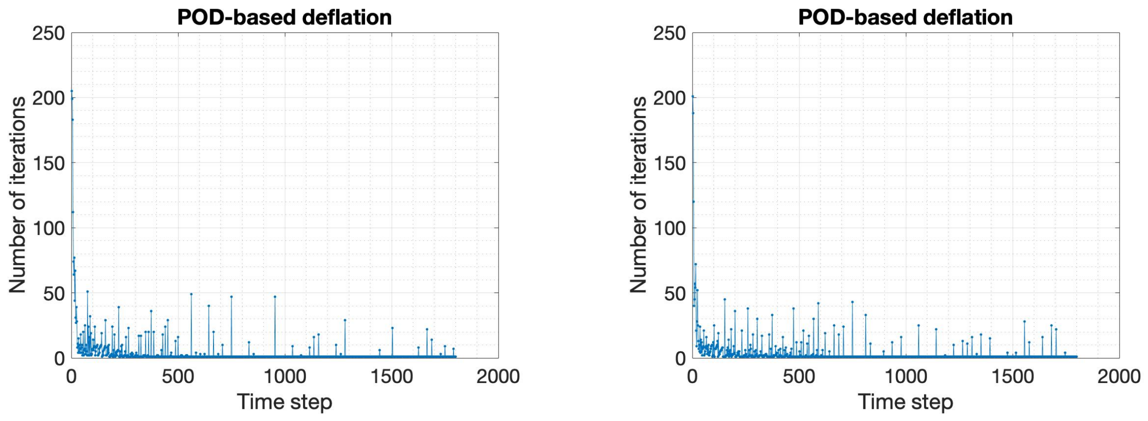


Figure 7.5: $f(T) := -22 + T + \alpha_1 T^2 + \alpha_2 T^4$, where $\alpha_2 = 1$, $\alpha_1 = -\frac{1+4\alpha_2(T^*)^2}{2T^*}$, and $T^* = 30$.

This test problem is constructed to see if POD-based deflation can deal with nonlinear problems as well. The transient deformation is again computed for time 0 to 1800 with time steps of 2 seconds, and the same parameter settings for POD are used as above. Figure 7.6a shows the number of iterations required by POD-based deflation with incomplete Cholesky preconditioner for the linear problem with renewed $B_T w$; it requires in total 3445 iterations. Figure 7.6b shows this for the nonlinear problem, which requires a total of 3301 iterations. So POD-based deflation can handle such nonlinear problems as well as the linear problems.



(a) Linear problem (b) Nonlinear problem

Figure 7.6: Number of iterations required for convergence with accuracy 10^{-8} per time step where the transient deformation is computed by 2L-PCG using incomplete Cholesky preconditioner and POD-based deflation.

Steady-State Deformation

The steady-state deformation $Ax = b$, where $A := K_{ff}$, x is the unknown deformation d_f and $b := f_{T_\infty f}$ as was solved before by AMG methods, is now solved by RBM-based deflation as preconditioner in 2L-PCG. We use either the incomplete Cholesky (iChol1), Jacobi or the less conventional damped Jacobi with $\omega = 0.1$ as preconditioner. Various numbers of domains are used for the construction of the RBM. The resulting numbers of iterations are shown in Table 7.5. For comparison, PCG with (damped) Jacobi preconditioner requires 280 iterations to reach the desired accuracy, and PCG with iChol1 preconditioner 201 iterations.

If considering a single domain, using the damped Jacobi preconditioner exceeds 280 iterations, while deflation should reduce the number of iterations. This is due to the additional Q term used in A-DEF2. Namely, the eigenvalues of A lie in some interval $[a, b]$. The preconditioner M^{-1} should centre the eigenvalues around 1 to decrease the condition number compared to the condition number of A, so that $M^{-1}A$ has eigenvalues in interval $[1 - \epsilon_1, 1 + \epsilon_2]$ for some $\epsilon_1, \epsilon_2 \geq 0$. Then $P^T M^{-1}A$ shifts some extreme eigenvalues of $M^{-1}A$ to 0, yielding eigenvalues in $\{0\} \cup [1 - \epsilon_3, 1 + \epsilon_4]$ with $\epsilon_3 \leq \epsilon_1, \epsilon_4 \leq \epsilon_2$. Finally, $M^{-1}PA + QA$ shifts those eigenvalues to 1 instead of 0, so the final eigenvalues lie in the interval $[1 - \epsilon_3, 1 + \epsilon_4]$; this shift to 1 prevents sensitivity to perturbations as mentioned in Chapter 5 [5]. If using a Cholesky or Jacobi preconditioner, this works as described. However, for the damped Jacobi the preconditioned eigenvalues are not necessarily centered around 1 (though they do decrease the condition number), hence shifting extreme eigenvalues to 1 can worsen the condition number again, which explains the poor convergence behaviour. To solve this problem, a scalar γ can be placed in front of Q (namely $P^T M^{-1}A + \gamma QA$) such that the extreme eigenvalues are not shifted to 1 but to γ which should be set to a value within the interval of eigenvalues. To test the strategy of using a shift γ for the damped Jacobi preconditioner set the parameter $\gamma = 0.01$. Using RBM (still considering a single domain) then requires only 233 iterations, which is now as expected below the 280 iterations required by PCG with the same damped Jacobi preconditioner. Instead of shifting the eigenvalues via a scaler in front of Q, we could use a shift in the preconditioner via e.g. $M^{-1} = \omega D^{-1} + 10^{-11}$. Then 301 iterations are needed, which is lower than when no correction is used, but still exceeds 280. So the initial approach of shifting Q is better. Note that the incomplete Cholesky preconditioner requires significantly fewer iterations than (shifted damped) Jacobi preconditioners; this is due to the fact that the Cholesky preconditioner clusters the eigenvalues closer to 1.

Preconditioner \ **# domains**	**1**	**2**	**3**	**4**	**5**	...	**10**	**30**
iChol1	105	104	97	91	91		85	75
Jacobi	233	228	479	411	406		402	389
damped Jacobi, $\omega = 0.1$	384	382	601	553	577		557	473
damped Jacobi, $\omega = 0.1$, $\gamma = 0.01$	233	228	479	411	408		401	389

Table 7.5: Number of iterations required for convergence with accuracy 10^{-8} of the steady-state deformation computed by 2L-PCG using RBM-based deflation with multiple preconditioners and various numbers of domains.

Now look at the table entries for multiple domains. In theory, more domains means a larger deflation subspace, hence is expected to decrease the required number of iterations. This is the case for the incomplete Cholesky preconditioner, but not for (damped) Jacobi. Note that for the damped Jacobi with shift, if considering multiple domains, the number of iterations is less than without the shift, but still does not decrease for an increasing number of domains. We take a closer look at the Jacobi preconditioner with RBM on one and on three domains in Table 7.6. It shows that the deflation improves the distribution of the eigenvalues for both cases. On a single domain, the condition number of $P^TM^{-1}A+QA$ is larger than that of $M^{-1}A$, but still the number of iterations decreases compared to PCG. For three domains, the condition number of $P^TM^{-1}A+QA$ is smaller, but unexpectedly this does not result in fewer iterations than for the single domain.

Matrix	**Min. eigenvalue**	**Max. eigenvalue**	**Condition number**
A	$5.528 \cdot 10^5$	$3.2165 \cdot 10^{11}$	$5.8190 \cdot 10^5$
$M^{-1}A$	$1.839 \cdot 10^{-5}$	5.4628	$3.5704 \cdot 10^5$
$P^TM^{-1}A+QA$, 1 domain	$4.3430 \cdot 10^{-4}$	5.4595	$8.2033 \cdot 10^5$
$P^TM^{-1}A+QA$, 3 domains	0.0010	5.4453	$5.0407 \cdot 10^5$

Table 7.6: Steady-state deformation. 2L-PCG using RBM-based deflation and Jacobi preconditioner for one and three domains.

To ensure that no mistake is made in the construction of the RBM, each RBM is considered separately and the number of iterations for Jacobi is given in Table 7.7. This table shows that all RBM perform well on a single domain, as none of them exceeds the 280 iterations required by PCG (deflation should always be at least as good as the non-deflated method). For multiple domains, every RBM requires more iterations; it is not the case that a single RBM is to blame for the overall increase of iterations. Moreover, the fact that RBM-based deflation with incomplete Cholesky does perform well supports the assumption that the deflation-subspace construction is correct.

# domains / ***Z***	**1**	**2**	**3**	**4**	**5**	…	**10**	**30**
x-translation	280	280	908	280	1017		919	901
y-translation	280	280	970	785	939		937	944
z-translation	233	229	938	938	927		902	898
rotation in x, y-plane	280	280	786	281	896		895	898
rotation in x, z-plane	280	280	1010	280	991		975	973
rotation in y, z-plane	280	281	989	731	993		988	983

Table 7.7: Number of iterations required for convergence with accuracy 10^{-8} of steady-state deformation computed by 2L-PCG using RBM-based deflation and Jacobi preconditioner, for various numbers of domains.

Next, the deflation-subspace matrix Z is modified to a matrix $\widetilde{Z}$ which is conjugate to A, i.e. $\widetilde{Z} = [\widetilde{Z}_1, \ldots, \widetilde{Z}_k]$ is such that $\widetilde{Z}_i^T A \widetilde{Z}_j = 0$ for $i \neq j$. This is done via the eigendecomposition of $A_c := Z^TAZ$ in the following way. Construct the deflation subspace Z consisting of the RBM as usual, then factorise $A_c = U\Lambda U^T$ (which is possible since A_c is a diagonalisable and real symmetric matrix), set $\widetilde{Z} := ZU$ and $A_c := \widetilde{Z}^T A \widetilde{Z}$. Then the new coarse grid operator A_c is the diagonal matrix containing the eigenvalues of the original coarse matrix $Z^T AZ$, which we place in descending order. The deflation step can now be viewed as a series of deflations, each having a rank one. Then the individual contribution of each column of $\widetilde{Z}$ can be studied as is done in Table 7.8 for Jacobi and incomplete Cholesky preconditioners for deflation using RBM based on a single domain. Note that the first column of $\widetilde{Z}$ corresponds to the largest eigenvalue, the second column to the second largest eigenvalue, etc. When considering a single column of $\widetilde{Z}$ and using the incomplete Cholesky preconditioner, the number of iterations decreases as the column number increases, and so the eigenvalue to which the column corresponds decreases. Thus, if a smaller size deflation-subspace matrix $\widetilde{Z}$ is to be used, then the columns corresponding to the smallest eigenvalues should be used as they decrease the number of iterations most, as also seen when comparing the number of iterations if using columns 1-3 compared to columns 4-6 in the last two rows of the table. For the Jacobi preconditioner, using a single column of $\widetilde{Z}$ does not have this nice (and expected) behaviour. Every column yields 280 iterations, except the third column which yields fewer iterations. This shows that also on a single domain, when given the choice between incomplete Cholesky and Jacobi preconditioners, the incomplete Cholesky is preferred, as its behaviour is more predictable.

Columns of $\widetilde{Z}$ used for deflation	**iChol1 preconditioner**	**Jacobi preconditioner**
All 6	105	233
1	192	280
2	190	280
3	186	233
4	186	280
5	186	280
6	183	280
1-3	163	233
4-6	151	280

Table 7.8: Number of iterations required for solving the steady-state deformation by 2L-PCG using RBM-based deflation with incomplete Cholesky or Jacobi preconditioner, on a single domain, for convergence with accuracy 10^{-8} when using certain columns of $\widetilde{Z}$ for deflation.

Another option of modifying the deflation-subspace matrix Z is by scaling the RBM such that each column of Z has 2-norm one. This does not result in great changes in the number of iterations, as seen in Table 7.9. This approach also does not solve the issue of the number of iterations increasing for the Jacobi preconditioner when more RBM domains are used.

# domains / **Preconditioner**	**1**	**2**	**3**	**4**	**5**	...	**10**	**30**
iChol1	105	104	97	91	91		85	75
iChol1, scaled RBM	105	105	97	91	91		85	75
Jacobi	233	228	479	411	406		402	389
Jacobi, scaled RBM	233	228	479	411	408		401	389

Table 7.9: Number of iterations required for convergence with accuracy 10^{-8} of steady-state deformation solved by 2L-PCG with RBM-based deflation and Jacobi or incomplete Cholesky preconditioners for various numbers of domains.

Finally, the tolerance is increased from 10^{-8} to 10^{-3}, to see if this causes the increasing number of iterations for 2L-PCG with Jacobi preconditioner and RBM-based deflation for an increasing number of domains. For this new tolerance on a single domain 118 iterations are required, on two domains 119, and on three domains 173; so the tolerance also is not the cause.

To conclude, incomplete Cholesky is preferred as preconditioner for 2L-PCG using RBM-based deflation. This is because (damped) Jacobi preconditioners require more iterations for RBM based on multiple domains, and do not have the nice behaviour observed for the diagonalised deflation-subspace matrix on a single domain. The (damped) Jacobi preconditioners do not behave as expected for RBM based on multiple domains, since the number of iterations increases; investigating it showed the following. First a shift in Q for the damped Jacobi preconditioner did improve the convergence behaviour but only for a single domain. For multiple domains, the condition number, separate RBM, scaled RBM, and a lower tolerance were investigated. No clear reason has been found as to why Jacobi does not improve if multiple domains are used for deflation.

If using 2L-PCG with RBM-based deflation and incomplete Cholesky preconditioner, Table 7.10 shows for various numbers of domains the required number of iterations, time and data-size. For comparison, PCG with incomplete Cholesky preconditioner requires 201 iterations, has data-size 30.1 MB and computation time $t_{tot} = 1.12$ s. Thus, when adding deflation based on RBM from a single domain the number of iterations almost halves, at the cost of a comparatively small data increase. If more domains are used, the number of iterations decreases further, but the time to compute the operators increases, as does the memory.

# domains	**# it.s**	t_{op}	t_{prep}	t_{solve}	t_{tot}	**Data-size (MB)**
1	105	0.02	0.27	0.89	1.18	31.4
2	104	0.03	0.27	0.90	1.12	32.2
3	97	0.04	0.29	0.83	1.16	33.9
4	91	0.04	0.26	0.78	1.08	33.9
5	91	0.06	0.25	0.79	1.10	35.6
10	85	0.10	0.26	0.76	1.11	40
30	75	0.36	0.29	0.68	1.33	58

Table 7.10: Number of iterations required for convergence with accuracy 10^{-8} of steady-state deformation computed by 2L-PCG using RBM-based deflation with incomplete Cholesky preconditioner for various numbers of domains. # it.s denotes the number of iterations, t_{op} the time required to construct the operators, t_{prep} the time to prepare the solver, t_{solve} the solve time, and t_{tot} the total time. The times are all given in seconds.

7.2.3 Block Preconditioners

The coefficient matrix of this test problem has a block structure of 3×3 blocks. To exploit this structure, block preconditioners can be used. Again, we consider the steady-state deformation, and compare the standard point-wise with the block version of Jacobi and incomplete Cholesky preconditioners for PCG. The point-wise Jacobi preconditioner needs 280 iterations and costs 15.1 MB. The block Jacobi does slightly better, namely only 278 iterations and 15.1 MB. The point-wise incomplete Cholesky with zero fill-in and no diagonal shift takes 201 iterations and 15.3 MB. The incomplete block Cholesky preconditioner takes only 132 iterations but 33.9 MB. Due to the fill-in being computed based on the blocks instead of single entries, the incomplete block Cholesky factor has more nonzeros than the point-wise version. So the block versions reduce the number of iterations, but at the cost of a higher data demand.

Conclusions on Numerical Results

For the steady-state temperature and deformation computation of the thermo-mechanical test problem, RS-AMG is preferred among the three AMG preconditioners if enough memory is available as it requires fewer iterations and generally has a smaller solve-time. Otherwise SA-AMG should be used as it has smaller coarse grid and data-sizes. For the steady-state deformation, also an SA-AMG version based on grouping the dofs per node together has been used, which in most cases reduces the number of iterations needed, the total computation time and the data-size compared to the non-grouped version. aSP-AMG performs worst out of the three multigrid preconditioners.

2L-PCG using POD-based deflation greatly reduces the number of iterations and total simulation time required to compute the transient temperature, compared to PCG without deflation. For the transient deformation, the combination of POD- and RBM-based deflation gives the greatest reduction in iterations and time. Moreover, POD-based deflation can also deal with nonlinear test problems.

For the steady-state deformation, 2L-PCG using RBM-based deflation worked well – in the sense that using more domains reduces the number of iterations – if incomplete Cholesky is used as preconditioner, but not when using Jacobi. Block preconditioners reduce the number of iterations for the steady-state deformation compared to their point-wise versions, but at the cost of larger data-sizes.

Comparing AMG with deflation for the steady-state deformation, both have been considered with as smoother respectively second preconditioner the incomplete Cholesky decomposition with zero fill-in. In this case, the RBM-based deflation preconditioner has a smaller data-size, computation times, and number of iterations than the AMG preconditioners, hence is the preferred method.

Chapter 8

Analysis of Iterative Methods for the Wafer-Slip Problem

The next test problem represents a small-size wafer-slip model as described in Chapter 3. The displacements u have to be computed from $K(u)u = F^E$, where the stiffness matrix K depends on the solution u itself. The solution is computed via an implicit scheme which is solved using the Newton method, requiring solving at each iteration i of time step n the system $Ax = b$, where $A := K_t(u_n^{i-1})$ (so A changes as it depends on the solution of the previous Newton step $i-1$ of the current time step n), $x := \Delta u^i$, and $b := F_n^E - F_n^{I,i-1}$, with K_t denoting the tangent stiffness matrix and F^I the in-balance force vector. The test model consists of the multiple phases. The springs supporting the wafer are slowly removed and replaced by (clamp) forces; then backfill gas-pressure is added. Each phase requires solving the displacements u for certain time-steps via the Newton method.

The aim is to improve the computation of a single Newton step. The specifications for the test model are the following. There are 3304 nodes, each with 3 dofs. This gives a total of 9912 dofs, but 16 dofs have been constrained and the corresponding rows and columns have been removed from the matrix. Thus the coefficient matrix A has size 9896×9896. A is SPD (up to machine precision) hence the PCG method can be used. The condition number of A is $\kappa_2(A) = \frac{\lambda_{max}}{\lambda_{min}} = \frac{3.4934 \cdot 10^{10}}{1.5212} \approx 2.2964 \cdot 10^{10}$. The coefficient matrix is badly conditioned hence it is important that an effective preconditioner is found for PCG. When looking at the sparsity pattern of A in Figure 8.1, we see that the matrix has a block structure with blocks of size 3, and that each quarter of the matrix has approximately the same sparsity pattern.

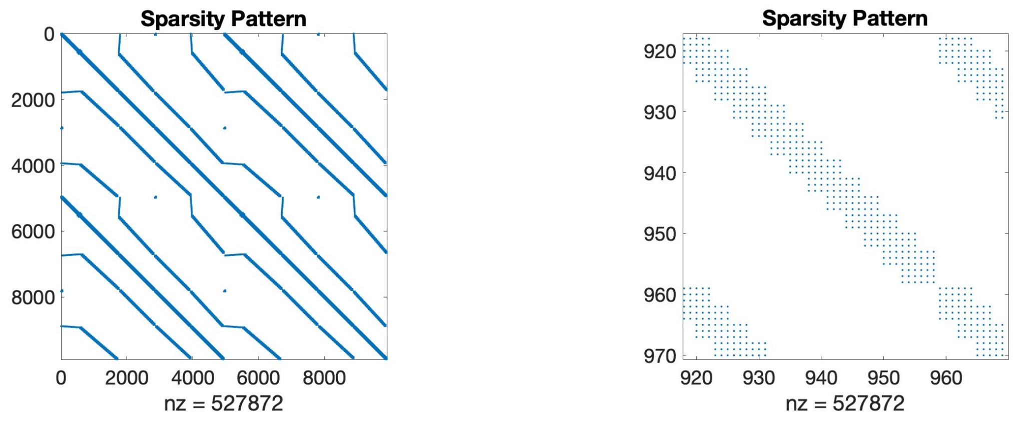


Figure 8.1: Sparsity pattern of the coefficient matrix (full and zoomed in) of a Newton step of the wafer-slip test model.

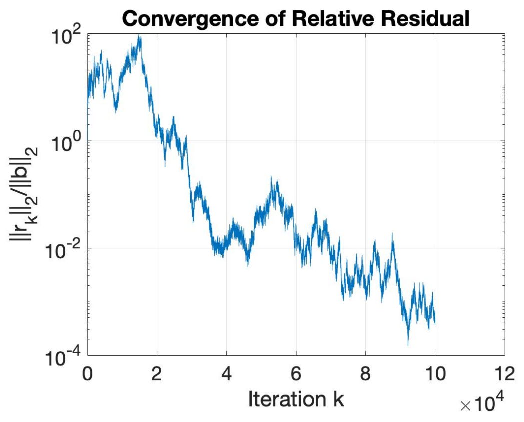
Below, various preconditioners for PCG are discussed to solve the first computation of displacements u in the model. We focus on solving only the first Newton step, since the total number of Newton steps required depends on the solver used. First, the convergence plots of each preconditioner are given. At the end they are all compared based on the number of iterations and data-size in Table 8.1. The methods are not compared based on computation time since preconditioners/smoothers that are implemented as scripts are compared to build-in Matlab functions which are optimised with respect to time.

Note that the coefficient matrix changes per Newton step, so the preconditioner might have to be recomputed or modified when computing further Newton steps. That is not the focus of this chapter, but some suggestions for updating the preconditioners are given in Appendix B.

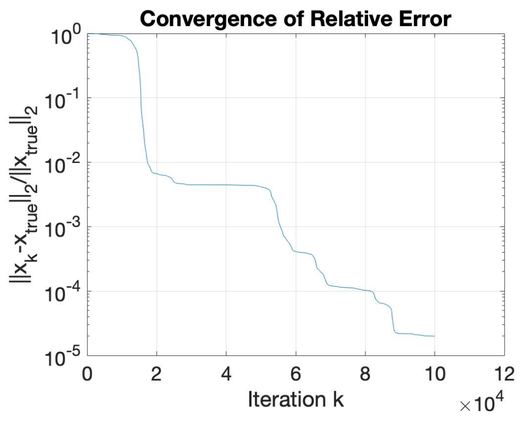
8.1 Deflation

We start with the Jacobi method. In Figure 8.2 the standard Jacobi preconditioner is used for PCG and the convergence behaviour of the relative residual and error norm are shown for the first 10^5 iterations. Even with this large number of iterations, the desired tolerance – the relative residual norm below 10^{-11} – is not yet reached.

The results of the block Jacobi preconditioner with 3×3 diagonal blocks are very similar to those of the standard Jacobi; there is no visible difference between the plots of the standard and the block Jacobi convergence. The final relative residual after 10^5 iterations is $4.4170 \cdot 10^{-4}$ for Jacobi and $6.4273 \cdot 10^{-4}$ for block Jacobi, so slightly worse. This shows that for the Jacobi preconditioner, exploiting the block structure of A in this case does not visibly improve the convergence.



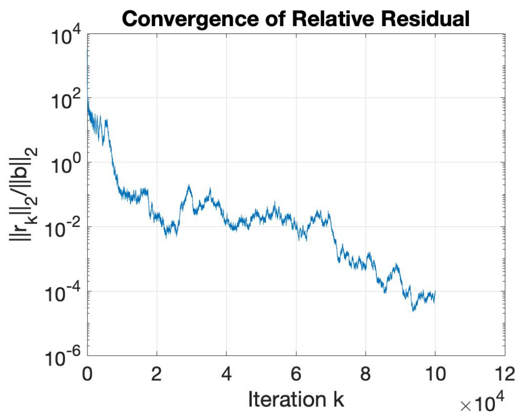
(a) Norm of the relative residual.



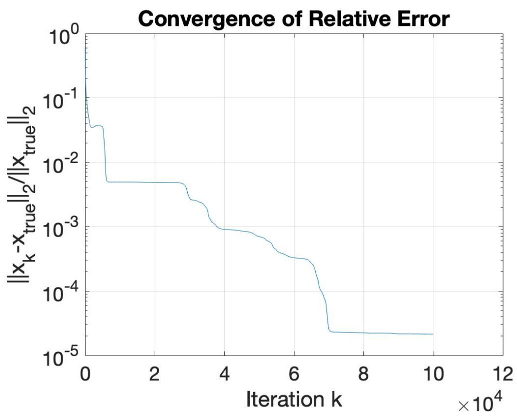
(b) Norm of the relative error.

Figure 8.2: Convergence behaviour of PCG with Jacobi preconditioner for the first Newton step of the wafer-slip test model.

The Jacobi preconditioner is also combined with RBM-based deflation in 2L-PCG, shown in Figure 8.3. Due to the physical properties of the wafer-slip model and the fact that certain nodes have been constrained, there are three (instead of six) RBM per ring in the wafer: the radial and vertical translation, and rotation around the circumference. Here a single ring is used. The convergence behaviour improves compared to not using deflation and the final relative residual decreases to $7.7257 \cdot 10^{-5}$, but still does not reach the desired tolerance. In each of the Jacobi error plots so far, the relative error norm stagnates for a number of iterations, and then suddenly drops. For the standard and block Jacobi these drops are at the same locations, but when using RBM-based deflation they change. This is because some eigenvalues are removed by the deflation and this improves the convergence by shortening/removing some stagnation intervals. However, there are still many such intervals left, possibly due to the distribution of the eigenvalues as plotted in Figure 8.4, which shows that there are multiple smaller eigenvalues. Namely, the RBM-based deflation has removed two eigenvalues within the ten smallest eigenvalues of $M^{-1}A$, and one eigenvalue within the ten largest. But they did not remove the smallest or largest eigenvalue; the condition number does not decrease, but it even increases from $2.5427 \cdot 10^{10}$ for $M^{-1}A$ to $3.383 \cdot 10^{14}$. However, the observed convergence behaviour is better when using RBM-based deflation than without.



(a) Norm of the relative residual.



(b) Norm of the relative error.

Figure 8.3: Convergence behaviour of 2L-PCG with Jacobi preconditioner combined with RBM-based deflation for the first Newton step of the wafer-slip test model.

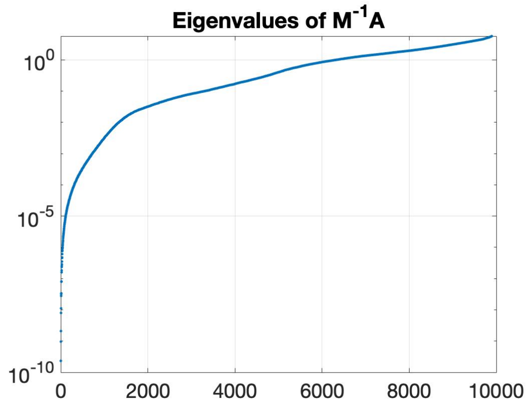
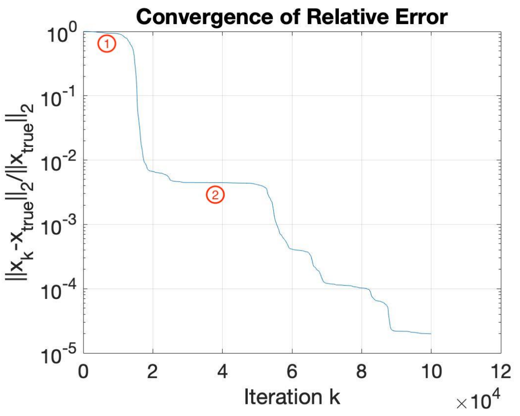
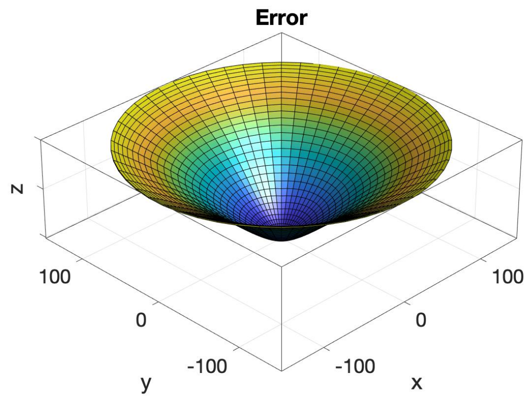


Figure 8.4: Eigenvalues of $M^{-1}A$, where M is the Jacobi preconditioner, for the first Newton step of the wafer-slip test model.

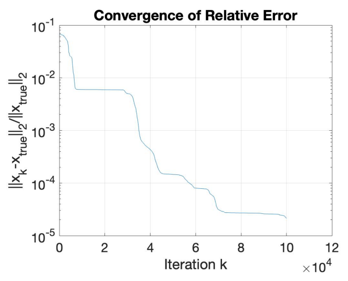
The RBM-based deflation does not remove the smallest eigenvalues for Jacobi preconditioners. Therefore, we consider 2L-PCG using deflation via eigenvectors instead of RBM to investigate the influence of the eigenvalues. In Figure 8.5b the absolute error of Jacobi preconditioned PCG is shown for the first stagnation interval as indicated in Figure 8.5a. This error resembles the absolute eigenvector corresponding to the third smallest eigenvalue of $M^{-1}A$. Using the corresponding eigenvector for deflation shortens the first stagnation interval, as shown in Figure 8.5c. In the second stagnation interval, the absolute error resembles the absolute value of a linear combination of the eigenvectors corresponding first and second smallest eigenvalues. Considering the eigenvectors of the three smallest eigenvalues as deflation vectors then also reduces the second stagnation interval, as shown in Figure 8.5d. Using say the 20 smallest eigenvectors reduces the required number of iterations to below 10^5 and removes some more stagnation intervals, but still the large stagnation interval at the end remains, see Figure 8.5e. Going even further and using the 100 smallest eigenvectors, this still shows the stagnation interval at the end, but it is shorter and moreover the number of iterations is significantly decreased (Figure 8.5f). If looking again at the plot of the eigenvalues of $M^{-1}A$ in Figure 8.4, it shows large gaps between the smaller eigenvalues, causing a stagnation interval when it is not yet converged for such an eigenvalue and then the sudden drop when it is. The larger eigenvalues are closer together. The relatively large stagnation interval at the end of the convergence plots may be caused by rounding errors and the small PCG-tolerance used, and is not caused by a single eigenvalue that is slow to converge. Based on the relative error norm one could say that nothing much happens in the final stagnation interval and that the method could just be stopped upon entering this interval. However, usually the true solution is not known, so that interval cannot be determined from the error. Instead, the relative residual is used to determine when the method has converged. Contrary to the error, the residual still significantly decreases over this interval, as seen in Figure 8.5g.



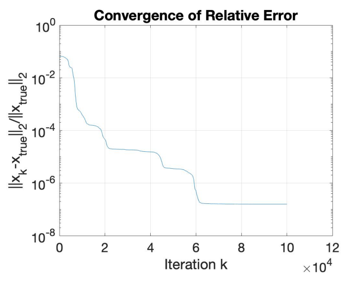
(a) Norm of the relative error of PCG with Jacobi preconditioner.



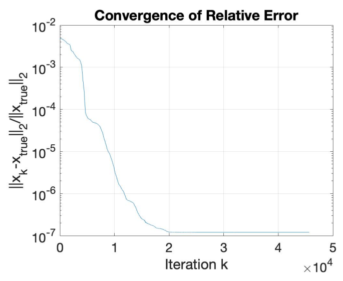
(b) Absolute error of PCG with Jacobi preconditioner at stagnation interval 1 as indicated in Figure 8.5a.



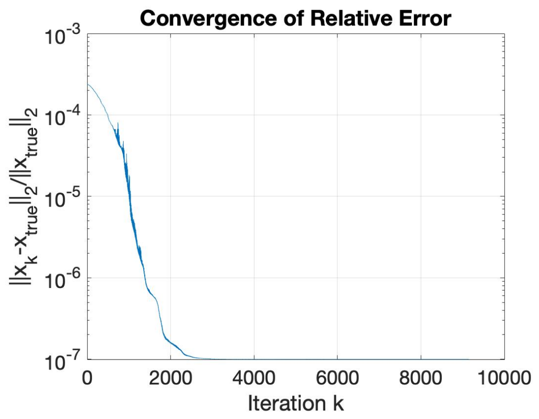
(c) Norm of the relative error of 2L-PCG with Jacobi preconditioner combined with deflation based on the third smallest eigenvector.



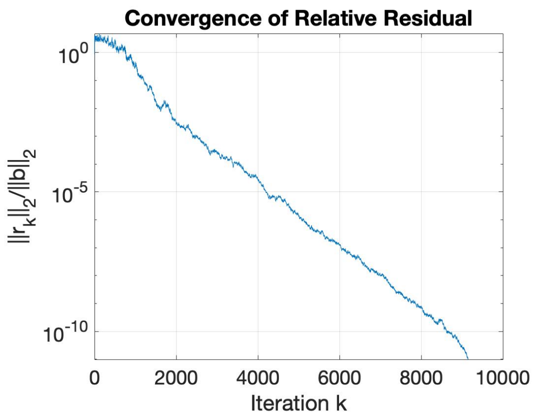
(d) Norm of the relative error of 2L-PCG with Jacobi preconditioner combined with deflation based on the three smallest eigenvectors.



(e) Norm of the relative error of 2L-PCG with Jacobi preconditioner combined with deflation based on the 20 smallest eigenvectors.



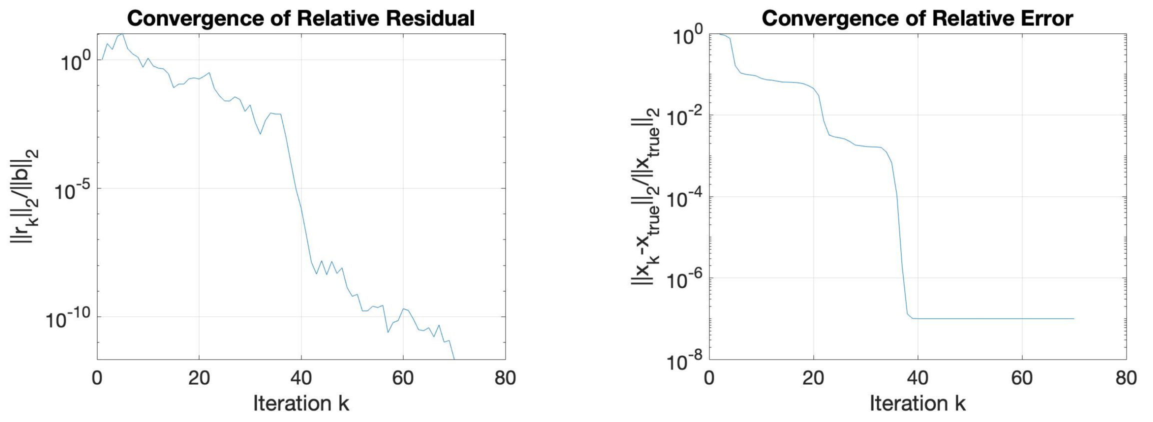
(f) Norm of the relative error of 2L-PCG with Jacobi preconditioner combined with deflation based on the 100 smallest eigenvectors.



(g) Norm of the relative residual of 2L-PCG with Jacobi preconditioner combined with deflation based on the 100 smallest eigenvectors.

Figure 8.5: Convergence behaviour of (2L-)PCG with Jacobi preconditioner for the first Newton step of the wafer-slip test model.

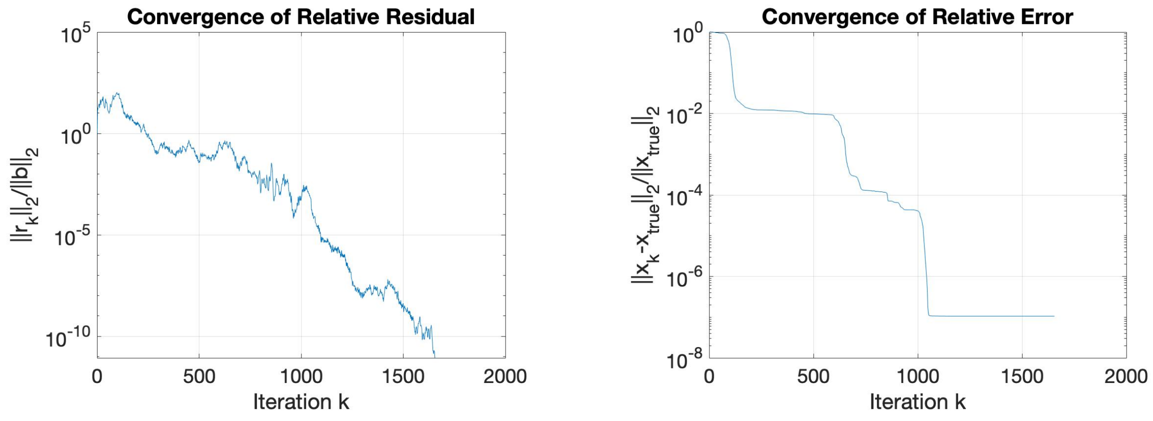
Next, consider the incomplete Cholesky decomposition as preconditioner. Because of the matrix properties, a small drop tolerance of 10^{-7} and diagonal shift of 10^{-7} are used to ensure that the method does not encounter a nonpositive pivot and that the number of iterations required is below the default maximum number of iterations 100. Also, the matrix is reordered according to Matlab's *dissect*. This results in 1,557,858 nonzeros in the incomplete Cholesky factor. For comparison, the Cholesky factor has 1,578,624 nonzeros, so the decrease in the number of nonzeros is small. As shown in Figure 8.6, the method converges in 69 iterations. If using 2L-PCG with incomplete Cholesky and RBM-based deflation using a single ring, only 57 iterations are required. The drop tolerance in the incomplete Cholesky decomposition can be increased if the diagonal shift is chosen such that no nonpositive pivots are encountered. This can be done by increasing the shift. Then the number of nonzeros in the incomplete Cholesky factor decreases but the number of iterations increases.



(a) Norm of the relative residual. (b) Norm of the relative error.

Figure 8.6: Convergence behaviour of PCG with incomplete Cholesky preconditioner for the first Newton step of the wafer-slip test model.

For this preconditioner a block version is employed as well. The incomplete block Cholesky decomposition with 3×3 blocks is used with a diagonal shift of 10^{-4}, and a fill-in level of 4. The resulting number of nonzeros is 936,916 and the number of iterations is 1654. The convergence is shown in Figure 8.7. Note that the implementation of the incomplete block Cholesky is not optimised with respect to time like the *ichol* function in Matlab, so it requires significantly more time to compute the block form at the moment. Again, adding RBM-based deflation based on a single ring improves the convergence; it reduces the number of iterations. Note that here – contrary to Jacobi – the condition number of $P^T M^{-1}A + QA$ is $1.8551 \cdot 10^8$ and does decrease compared to $M^{-1}A$ which has a condition number of $1.0284 \cdot 10^9$.



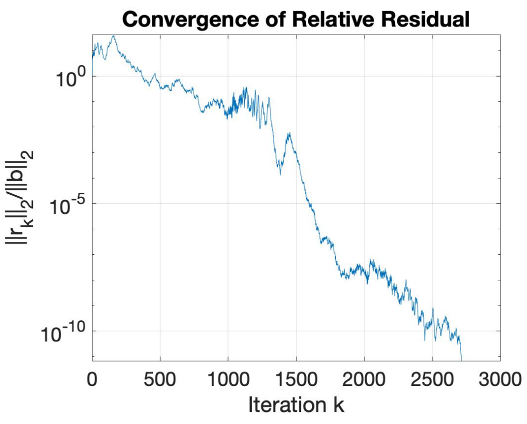
(a) Norm of the relative residual. (b) Norm of the relative error.

Figure 8.7: Convergence behaviour of PCG with incomplete block Cholesky preconditioner for the first Newton step of the wafer-slip test model.

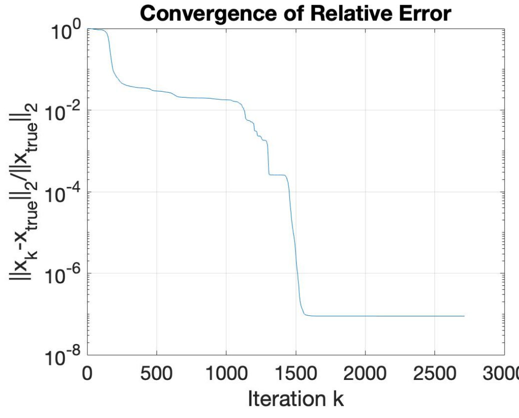
8.2 Algebraic Multigrid

Grouped SA-AMG can also be used as a preconditioner. This way we can exploit the matrix structure by grouping the dofs per node together, i.e. grouping a 3×3 block together and considering it as a single dof when constructing the coarse grid. Note that for the thermo-mechanical model we have already seen that the grouped SA-AMG generally uses less iterations, data and total time than the non-grouped version, so only the grouped SA-AMG will be applied. The figure below shows the convergence of PCG using as preconditioner grouped SA-AMG with $\theta = 0.01$, two levels and incomplete Cholesky smoother with *dissect* reordering, drop tolerance 10^{-4} and diagonal shift 10^{-3}. The coarse grid operator is only 126×126, so much smaller than the original fine grid. The number of iterations required to reach the desired tolerance is 2713. Note that similarly to the Jacobi preconditioner, again a relatively large stagnation interval occurs at the end for the error plot; this also happened for the incomplete Cholesky preconditioners.

When comparing to RS-AMG with the same preconditioner and $\theta = 0.8$, it requires the same number of iterations and the convergence plots are very similar to grouped SA-AMG. However, the coarse grid of RS-AMG has size 5012×5012 and the data-size increases from 35.1 to 51.7 MB. aSP-AMG has for the thermo-mechanical model already been shown to perform worst among the three AMG methods, hence is not applied for the wafer-slip model. So the best AMG method to use as preconditioner is the grouped SA-AMG method.



(a) Norm of the relative residual.



(b) Norm of the relative error.

Figure 8.8: Convergence behaviour of PCG with as preconditioner grouped SA-AMG with incomplete Cholesky smoother for the first Newton step of the wafer-slip test model.

8.3 Comparison

In Table 8.1 the number of iterations and data-size of the various preconditioners are given. The Jacobi method uses the least data, but also takes so many iterations that it is not usable in this case. It might seem counter-intuitive that block Jacobi has a smaller data-size than point-wise Jacobi, but this is caused by the following. For block Jacobi, storing D^{-1}, the inverse of the block diagonal matrix of A, requires more bytes than for point-wise Jacobi (554,488 compared to 79,344), but storing $A - D$ requires fewer bytes (8,050,680 compared to 8,683,640); hence the block Jacobi has a smaller data-size.

The incomplete Cholesky preconditioner with drop tolerance 10^{-7}, diagonal shift 10^{-7} and *dissect* reordering (denoted iChol$(10^{-7}, 10^{-7})$ in the table) needs few iterations to converge, but uses almost as much data as the Cholesky decomposition which needs only 2 iterations to converge. This shows that the incomplete Cholesky with such high levels of fill in are no improvement upon simply using the Cholesky decomposition, which is generally considered expensive. The incomplete Cholesky preconditioner combined with RBM-based deflation shows that using a single ring reduces the number of iterations. But using more rings does not necessarily decrease this number further and can even increase it compared to a single ring. Moreover, it increases the data-size even further.

The incomplete block Cholesky preconditioner is used with fill-in level 4, diagonal shift 10^{-4} and *dissect* reordering (denoted block iChol$(4, 10^{-4})$) and corresponds to a lower triangular matrix with 936,916 nonzeros. For comparison, iChol$(2.5 \cdot 10^{-5}, 10^{-4})$ is also considered and results in 1,127,730 nonzeros. Note that the block version has fewer nonzeros, (hence) a lower data-size and also fewer iterations. Thus, exploiting the block structure of the coefficient matrix is beneficial. While the number of iterations is much higher than iChol$(10^{-7}, 10^{-7})$, it uses significantly less data than the Cholesky decomposition. Again, using RBM-based deflation on a single ring reduces the number of iterations at the cost of a slightly larger data-size.

Using as preconditioner grouped SA-AMG with incomplete Cholesky smoother, we see that compared to solely using iChol as preconditioner, it reduces the number of iterations and increases the data-size. Compared to preconditioner block iChol$(4, 10^{-4})$, the data-sizes are in the same neighbourhood when grouped SA-AMG with iChol$(10^{-4}, 10^{-3})$ or iChol$(5 \cdot 10^{-3}, 3 \cdot 10^{-4})$ smoother is used, but in both cases more iterations are needed by SA-AMG. If grouped SA-AMG uses iChol$(10^{-7}, 10^{-7})$ as smoother, then the number of iterations is lower than using iChol$(10^{-7}, 10^{-7})$ as preconditioner combined with RBM-based deflation. While the data-size of grouped SA-AMG is larger than iChol$(10^{-7}, 10^{-7})$ with RBM-based deflation using a single ring, it is lower if more rings are used and moreover the number of iterations remains lower for grouped SA-AMG. If grouped SA-AMG uses block iChol as smoother, the number of iterations is almost equal to when using block iChol preconditioner combined with RBM-based deflation, but the data-size is larger.

Preconditioner	**Settings**	**# iterations**	**Data-size (MB)**
Jacobi		$> 10^5$	8.51
Block Jacobi		$> 10^5$	8.36
Jacobi & RBM	#rings = 1	$> 10^5$	8.97
Cholesky decomposition		2	56.7
iChol$(10^{-7}, 10^{-7})$		69	56.1
iChol$(10^{-7}, 10^{-7})$ & RBM	#rings = 1	57	56.5
iChol$(10^{-7}, 10^{-7})$ & RBM	#rings = 2	53	57
iChol$(10^{-7}, 10^{-7})$ & RBM	#rings = 3	59	57.4
iChol$(10^{-7}, 10^{-7})$ & RBM	#rings = 10	59	60.6
Block iChol$(4, 10^{-4})$		1654	37.1
Block iChol$(4, 10^{-4})$ & RBM	#rings = 1	1363	37.6
iChol$(2.5 \cdot 10^{-5}, 10^{-4})$		1940	42.9
SA-AMG with iChol$(2.5 \cdot 10^{-5}, 10^{-4})$	#levels = 2, $\theta = 0.01$	1920	44.1
SA-AMG with iChol$(10^{-4}, 10^{-3})$	#levels = 2, $\theta = 0.01$	2713	35.1
SA-AMG with iChol$(5 \cdot 10^{-5}, 3 \cdot 10^{-4})$	#levels = 2, $\theta = 0.01$	1684	39.9
SA-AMG with iChol$(10^{-7}, 10^{-7})$	#levels = 2, $\theta = 0.01$	48	57.2
SA-AMG with block iChol$(4, 10^{-4})$	#levels = 2, $\theta = 0.01$	1362	38.3

Table 8.1: Results of (2L-)PCG with various preconditioners for the first Newton step of the wafer-slip test model.

Conclusions on Numerical Results

The above analysis shows that for the wafer-slip test model, the Jacobi preconditioner requires too many iterations to be of use. The incomplete Cholesky preconditioner can be combined with RBM-based deflation using a single ring to reduce the number of iterations. Also, an AMG preconditioner with incomplete Cholesky as smoother could be used to lower the number of iterations; where the best to use is grouped SA-AMG. Of course, adding RBM-based deflation or using grouped SA-AMG increases the data-size. The incomplete Cholesky should be given a drop tolerance (and corresponding diagonal shift) such that it is cheaper to store than the Cholesky decomposition, otherwise that could be used as well and converges faster – but is usually too expensive in practice. The incomplete block Cholesky preconditioner resulted in fewer iterations and a smaller data-size than its point-wise version, hence is better to use. However, this is with the side-note that the computation of the block version is at the moment much slower than Matlab's built-in *ichol* function. To reduce the number of iterations further, incomplete block Cholesky should be combined with RBM-based deflation based on a single ring.

Chapter 9

Conclusion

The goal of this thesis was to find an efficient way of solving large-scale sparse linear systems of equations arising in large thermo-mechanical models where both the thermal and mechanical part of the model have to be solved, as well as in wafer-slip models. For this reason, we have examined the efficiency of certain iterative methods, in terms of the number of iterations, the computation time and the amount of memory needed. The main solver used is the state-of-the-art Krylov subspace method PCG, which generates iterates in the Krylov subspace $x_0 + \mathcal{K}_k(A, r_0)$ with minimal A-norm and has short recurrences. We have considered AMG methods, deflation methods and block preconditioners as preconditioner for PCG; combined with direct and stationary iterative methods as smoother for AMG or as second preconditioner for deflated PCG. The resulting solvers have been applied to test problems representing small-size thermo-mechanical and wafer-slip models.

The AMG methods considered are the classical/RS-AMG, SA-AMG and aSP-AMG. The numerical results have shown that the SA-AMG preconditioner requires the least amount of memory and quickly reduces the size of the coarser grids. Moreover, if the coefficient matrix has a block structure, the grouped SA-AMG exploits this structure resulting in less memory and in most cases smaller total computation times and fewer iterations as well. However, for the thermo-mechanical model RS-AMG needs the least iterations, and has smaller solve-times. aSP-AMG performs worst among the AMG methods. Thus, the preferred AMG preconditioner is either (grouped) SA-AMG or RS-AMG for the thermo-mechanical model, depending on the amount of memory available. The preferred AMG preconditioner for the wafer-slip model is grouped SA-AMG.

Thereafter, the deflation methods based on RBM and/or POD have been discussed. They are used as second-level preconditioner, besides a direct or stationary iterative method as first-level preconditioner; the resulting solver is a 2L-PCG method. The implementation of 2L-PCG methods is in this paper done according to A-DEF2. This avoids sensitivity to perturbations in coarse solves, is able to deal with severe termination tolerances and guarantees convergence. The POD-based deflation has been used for the transient thermo-mechanical test problem and greatly reduces the number of iterations and total simulation time. The RBM-based deflation has been applied for the mechanical part of the thermo-mechanical model and for the wafer-slip model. Moreover, for the transient mechanical part of the thermo-mechanical test problem, the combination of RBM-

and POD-based deflation has been used and this gives the largest reduction in iterations and time. For its steady-state case, RBM-based deflation considered not only RBM computed based on a single domain, but on multiple domains as well. In that case, if the incomplete Cholesky is the single-level preconditioner, then it reduces the number of iterations when the number of domains for the RBM increases, as expected. But on the other hand, with a Jacobi preconditioner the number of iterations increases, which is undesirable. This shows the importance of choosing the single-level preconditioner of the 2L-PCG method with care. Finally, for the wafer-slip model, using RBM based on a single ring reduces the number of iterations required to converge, at the cost of a small increase in the required amount of memory.

If the coefficient matrix has a block structure, it can also be taken advantage of by the direct and stationary iterative preconditioners. For both the Jacobi and incomplete Cholesky decomposition, block versions have been constructed. The mechanical part of the thermo-mechanical model and the wafer-slip model exhibit a 3×3 block structure, since each node has degrees of freedom regarding the x-, y- and z-direction. The block preconditioners then reduce the number of iterations compared to their point-wise versions (possibly increasing the required amount of memory).

To summarise, (grouped) SA-AMG is preferred over RS-AMG if memory is limited. For transient problems, POD-based deflation greatly reduces the number of iterations and simulation time, even more so when combined with RBM. For the mechanical problems, RBM-based deflation reduces the required number of iterations at a low memory cost – lower than for AMG – if using a single domain/ring. For block-structured matrices, the block direct or stationary iterative preconditioners decrease the number of iterations compared to their point-wise versions.

So from the solver combinations investigated, the best is using 2L-PCG methods when possible, where deflation is used as the second preconditioner, implemented according to A-DEF2. Only the steady-state temperature does neither have RBM nor is transient, hence none of the deflation methods were applied. In this case RS-AMG is preferred, unless the available memory is limited, then SA-AMG is best to use.

Future Research

There are several possibilities for future research, which due to the time span fell outside the scope of this thesis and have not yet been addressed. One could look further into the increasing number of iterations when employing 2L-PCG with RBM-based deflation preconditioner using multiple domains to construct the RBM, combined with the Jacobi preconditioner. In Section 7.2.2 this phenomenon has been investigated from several angles, but the cause was not found. In the category of block preconditioners, the incomplete block Cholesky implementation of Algorithm 5 could be optimised with respect to time. Similarly, the SA-AMG algorithm could be made more time-efficient by parallelising it, as discussed in [14]. This thesis focused on the sequential implementation of the solvers. The parallel setting could be investigated as well. A class of preconditioners that exploits parallel computing is the domain decomposition methods. They use a divide-and-conquer strategy, where the problem is solved by splitting the domain into multiple subdomains. The paper [29] investigates efficient parallel preconditioners based on domain decomposition techniques for problems on unstructured grids. It considers one- and two-level Schwarz, and (approximate) Schur complement domain decomposition methods.

For multidimensional systems like the mechanical part of the thermo-mechanical model and the wafer-slip model, tensors could be investigated as a way of preserving the multidimensional structure of the model. Article [30] introduces a MOR technique for systems with multiple independent variables, by combining the techniques for tensor decomposition and POD. The difficulty lies in generalising tensors to unstructured grids.

For the wafer-slip model, the preconditioning of the first Newton step has been investigated in Chapter 8. In the following Newton steps, the coefficient matrix changes due to its dependence on the solution itself. However, only the computation of the initial preconditioner has been discussed, not yet a way to update the preconditioner for changing coefficient matrices. Namely, recomputing the entire preconditioner is wasteful since the coefficient matrix does not change “too much” per step (which translates in low-rank updates). In Appendix B, a start is made with investigating possible ways to update the preconditioner.

Bibliography

[1] V. Bakshi. *EUV Sources for Lithography.* SPIE – The International Society for Optical Engineering, 2006.

[2] G. Diaz Cortes. POD-Based Deflation Method For Reservoir Simulation. 2019. Dissertation at Delft University of Technology.

[3] T. B. Jönsthövel, M. B. Van Gijzen, C. Vuik, and A. Scarpas. On the use of rigid body modes in the deflated preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 35, 2013.

[4] A. Franceschini, V. A. Paludetto Magri, G. Mazzucco, N. Spiezia, and C. Janna. A robust adaptive algebraic multigrid linear solver for structural mechanics. *Comp. Methods in Appl. Mech. and Eng.*, 2 2019.

[5] J. M. Tang, R. Nabben, C. Vuik, and Y. A. Erlangga. Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods. *Journal of Scientific Computing*, 39:340–370, 6 2009.

[6] J. M. Tang. Two-Level Preconditioned Conjugate Gradient Methods with Applications to Bubbly Flow Problems. 2008. Dissertation at Delft University of Technology.

[7] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, 2 edition, 2003.

[8] C. Vuik and D. J. P. Lahaye. Scientific computing (wi4201), 2019. Delft University of Technology Faculty of Electrical Engineering, Mathematics and Computer Science.

[9] P. Sonneveld and M. B. Van Gijzen. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM Journal on Scientific Computing*, 31:1035–1062, 2008.

[10] G. Allaire and S. M. Kaber. Direct Methods for Linear Systems. In J. E. Marsden, L. Sirovich, and S. S. Antman, editors, *Numerical Linear Algebra*, volume 55 of *Texts in Applied Mathematics*, chapter 6, pages 97–124. Springer, 2008.

[11] T. B. Jönsthövel, M. B. Van Gijzen, S. MacLachlan, C. Vuik, and A. Scarpas. Comparison of the deflated preconditioned conjugate gradient method and algebraic multigrid for composite materials. *Computational Mechanics*, 50:321–333, 2011.

[12] A. S. Ahmadian. Chapter 6 - Numerical Methods and Procedures. In *Numerical Models for Submerged Breakwaters : Coastal Hydrodynamics and Morphodynamics*. Elsevier Science & Technology, 2016.

[13] A. Quarteroni, F. Saleri, and P. Gervasio. *Scientific Computing with MATLAB and Octave*, volume 2 of *Texts in Computational Science and Engineering*. Springer, 4 edition, 2014.

[14] R. S. Tuminaro and C. Tong. Parallel Smoothed Aggregation Multigrid: Aggregation Strategies on Massively Parallel Machines. *Proceedings of the IEEE/ACM SC2000 Conference*, 2000.

[15] M.E. Verbeek. *Iterative solvers and preconditioning for electromagnetic boundary integral equations*. 2001. Proefschrift Universiteit Utrecht.

[16] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Algebraic Multigrid Based On Element Interpolation (AMGe). *SIAM Journal on Scientific Computing*, 22, 01 2002.

[17] U. M. Yang. Parallel Algebraic Multigrid Methods — High Performance Preconditioners. In Are Magnus Bruaset and Aslak Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*. Springer Berlin Heidelberg, 2006.

[18] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. Adaptive Smoothed Aggregation (αSA) Multigrid. *SIAM Review*, 47:317–346, 6 2005.

[19] R. Gandham, K. Esler, and Y. Zhang. A GPU accelerated aggregation algebraic multigrid method. *Computers and Mathematics with Applications*, 68:1151–1160, 11 2014.

[20] A. Fujii, A. Nishida, and Y. Oyanagi. Evaluation of parallel aggregate creation orders: Smoothed aggregation algebraic multigrid method. volume 172, pages 99–122. Springer New York LLC, 2005.

[21] V. A. Paludetto Magri, M. Ferronato, A. Franceschini, and C. Janna. A novel AMG approach based on adaptive smoothing and prolongation for reservoir simulations. European Association of Geoscientists and Engineers, EAGE, 2018.

[22] J. T. Katsikadelis. Chapter 12 - Multi-degree-of-freedom systems: Free vibrations. In J. T. Katsikadelis, editor, *Dynamic Analysis of Structures*, pages 523–600. Academic Press, 2020.

[23] MathWorks. Singular Values. https://nl.mathworks.com/help/matlab/math/singular-values.html, R2021a.

[24] M. Zoutendijk. Applying Deflation Methods in a Topology Optimization Procedure, 2019. Thesis at Delft University of Technology.

[25] B. Yang, H. Liu, H. Zhong, and Z. Chen. Decoupled Block-Wise ILU(k) Preconditioner on GPU. 3 2017. University of Calgary.

[26] A. Napov. Conditioning analysis of incomplete Cholesky factorizations with orthogonal dropping. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1148–1173, 2013.

[27] MathWorks. ichol: incomplete Cholesky factorization. https://nl.mathworks.com/help/matlab/ref/ichol.html, R2021a.

[28] MathWorks. Sparse Matrix Reordering. https://www.mathworks.com/help/matlab/math/sparse-matrix-reordering.html, R2021a.

[29] M. Sala. Domain decomposition preconditioners: theoretical properties, application to the compressible Euler equations, parallel aspects. 2003. Dissertation at École Polytechnique Fédérale de Lausanne.

[30] F. Van Belzen and S. Weiland. A tensor decomposition approach to data compression and approximation of ND systems. *Multidimensional Systems and Signal Processing*, 23:209–236, 2010.

[31] V. A. Paludetto Magri. Development of Scalable Linear Solvers for Engineering Applications. 2018. Doctor of Philosophy, Università Degli Studi Di Padova, Department of Civil, Construction and Environmental Engineering (ICEA).

[32] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* Philadelphia: Society for Industrial and Applied Mathematics, second edition, 2002.

[33] C. Calgaro, J. P. Chehab, and Y. Saad. Incremental incomplete LU factorizations with applications. *Numerical Linear Algebra with Applications*, 17:811–837, 10 2010.

Appendix A

aSP-AMG

Algorithm 6: Interpolation weights generation via DPLS [4, 31].
Input: $d_p, \kappa_p, n_{max}, N_t, X, S_c, C, F$
Output: W

```
for $node\ i \in F \backslash C$ do
    $\bar{X} = X$, $x_i = X(i,:)^T$, $r = x_i$
    $k = 0$, $\bar{\mathcal{C}}_i = \emptyset$, $R = \emptyset$
    $\mathcal{C}_i = \{j \in C \mid \exists$ path from $i$ to $j$ in the $S_c$ graph shorter than $d_p\}$
    while $cond(R) \leq \kappa_p$ & $k < n_{max}$ do
        $k = k + 1$
        Select $\bar{j} \in \mathcal{C}_i \backslash \bar{\mathcal{C}}_i$ for which $\bar{x}_j$ has maximal affinity with $r$
        $\bar{\mathcal{C}}_i = \bar{\mathcal{C}}_i \cup \{\bar{j}\}$
        $\beta = min_\beta ||x_i - \sum_{j \in \bar{\mathcal{C}}_i} \beta_j \bar{x}_j||$
        $r = x_i - \sum_{j \in \bar{\mathcal{C}}_i} \beta_j \bar{x}_j$
        $R(:, end + 1) = \bar{x}_{\bar{j}}$
        Compute Householder reflection $Q$ nullifying last $N_t - k$ rows of last column of $R$
        $R = QR$, $r = Qr$
        for $all\ j \in \mathcal{C}_i \backslash \bar{\mathcal{C}}_i$ do
            $\bar{x}_j = Q\bar{x}_j$
    $w_i = R^{-1} r$
```

Algorithm 7: Test space generation via SRQCG [4, 31].
Input: $N_{it}, N_t, A, N, G, X_0$
Output: X

```
$Z_0 = (\mathcal{I} - GAG^T)X_0$
Orthonormalize $Z_0$
$SZ_0 = GAG^T Z_0$
$\beta_1 = 0; P_1 = zeros(N, N_t)$
for $k = 2 : N_{it} + 1$ do
    for $i = 1 : N_t$ do
        $q_{k-1}(i) = (SZ_{k-1}(:, i))^T Z_{k-1}(:, i)$          Compute Rayleigh quotient
        $R_{k-1}(:, i) = SZ_{k-1}(:, i) - q_{k-1}(i) Z_{k-1}(:, i)$     Compute residual
        if $k > 2$ then
            $\beta_{k-1}(i) = 2(SP_{k-1}(:, i))^T R_{k-1}(:, i)/n_{k-1}(i)$
        $P_k(:, i) = 2R_{k-1}(:, i) - \beta_{k-1}(i) P_{k-1}(:, i)$
        $SP_k(:, i) = GAG^T P_k(:, i)$
        $m_k(i) = (SZ_{k-1}(:, i))^T P_k(:, i)$
        $n_k(i) = (SP_k(:, i))^T P_k(:, i)$
        $p_k(i) = (Z_{k-1}(:, i))^T P_k(:, i)$
        $q_k(i) = (P_k(:, i))^T P_k(:, i)$
        $r_k(i) = (SZ_{k-1}(:, i))^T Z_{k-1}(:, i)$
        $a_0 = m_k(i) - p_k(i) r_k(i)$
        $a_1 = q_k(i) r_k(i) - n_k(i)$
        $a_2 = n_k(i) p_k(i) - m_k(i) q_k(i)$
        $\alpha_k(i) = \frac{a_1 + \sqrt{a_1^2 - 4a_0 a_2}}{2a_2}$
        $Z_k(:, i) = Z_{k-1}(:, i) + \alpha_k(i) P_k(:, i)$
        $SZ_k(:, i) = SZ_{k-1}(:, i) + \alpha_k(i) SP_k(:, i)$
    $D_k = Z_k^T SZ_k$
    $E_k = Z_k^T Z_k$
    Solve generalized eigenproblem $D_k U_k = E_k U_k \Lambda_k$
    $Z_k = Z_k U_k$
    $SZ_k = SZ_k U_k$
$X = G^T Z_{nit}$
Orthonormalize $X$
```

Appendix B

Updating Preconditioner

Chapter 8 discusses a test problem representing a small-size wafer-slip model, where the displacements are computed via an implicit scheme which is solved using the Newton method. This requires solving at each iteration i of time step n the system $Ax = b$, where A depends on the solution of the previous Newton step $i-1$ of the current time step n, hence changes per step. Chapter 8 focusses on improving the first Newton step, where the preconditioner has to be computed from scratch. This can be done prior to the computation so is allowed to take more time than the computation of preconditioners in later steps that have to be done during the simulation. In the following Newton steps, the coefficient matrix changes slowly and it is wasteful to recompute the entire preconditioner. The same preconditioner could be used for all the steps if the matrix does not change too much over the whole computation. When this is not the case (i.e. when the small matrix changes per step accumulate to a significant change) the initial preconditioner will not give good results for later steps. Therefore, we look at ways of updating the preconditioner for slowly varying coefficient matrices. If the Jacobi preconditioner is used, it can easily be updated. Namely, just modify the diagonal entries that have changed in the corresponding entries of the coefficient matrix. However, it was shown that Jacobi requires too large a number of iterations to be usable. Thus, updating methods for more different preconditioners – like the incomplete Cholesky factorisation – must be considered.

The Sherman–Morrison–Woodbury formula in B.1 can be used to update the preconditioner for low-rank changes in the coefficient matrix.

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{B.1}$$

Assume that the known preconditioner from the previous step approximates the coefficient matrix adequately, i.e. $M^{-1} \approx A^{-1}$, and that the change in A given by $\Delta A = UCV$ (the singular value decomposition of the matrix update) is of low rank k. In this wafer-slip test model each coefficient matrix is SPD, so $V = U^T$ and the Sherman–Morrison–Woodbury formula simplifies to the Equation B.2, where A is previous coefficient matrix and $\Delta A = UCU^T$ the singular value decomposition of the matrix update [32].

$$(A + UCU^T)^{-1} \approx M^{-1} - M^{-1}U(C^{-1} + U^T M^{-1}U)^{-1}U^T M^{-1} \tag{B.2}$$

If the initial preconditioner approximates A^{-1} well, then this method is expected to yield good results. If this is not the case, for instance if a shift is used in the incomplete Cholesky computation, it might not guarantee improvement. Moreover, M^{-1} is not computed explicitly for incomplete Cholesky, but inverse multiplication is done via triangular solves (via L and L^T). Thus, the updated inverse is only computed when multiplying with a vector, and cannot be efficiently stored. This causes problems after a number of steps as the accumulated matrix update with respect to the initial matrix must be used and hence increases in rank.

Another option is using the algorithms suggested in [33]. This article examines ways of updating preconditioners for slowly varying matrices. It investigates methods for computing incremental incomplete LU factorisations, based on approximate inverses and alternating techniques. The minimal energy residual descent for LU (MERLU) algorithm uses techniques based on sparse approximate inverses. Its advantage is that it guarantees that the Frobenius norm of the residual is non-increasing. However, the disadvantage is the cost of executing the algorithm. In this thesis not incomplete LU but incomplete Cholesky preconditioners have been used, hence the modification of the MERLU algorithm for updating incomplete Cholesky factorisations is given in Algorithm 8. Another approach is to exploit alternating procedures, as done in the iterative threshold alternating lower-upper correction (ITALU) algorithm. The ITALU method performed quite well in the experiments conducted in [33]. The drawback of the method is that breaks down when a singular factor U is encountered. Note, similar problems occur in computing incomplete LU factorisations with dropping. Moreover, this breakdown happens rarely in practice. The incomplete Cholesky modification of ITALU is given in Algorithm 9.

Algorithm 8: Minimal Energy Residual descent for Cholesky (Cholesky modification of MERLU [33]).
Input: coefficient matrix A, initial (upper triangular) Cholesky factor U

```
for $k = 1, \dots$ do
    $R = A - U^T U$
    $X_U = triu(UR)$
    Apply numerical dropping to $X_U$
    $C = U^T G$
    $\alpha = \frac{Tr(C^T R)}{||C||_F^2}$
    $U = U + \alpha X_U$
end
```

Algorithm 9: Cholesky modification of ITALU (Iterative Thershold Alternating Lower-Upper correction [33]).
Input: coefficient matrix A, initial (upper triangular) Cholesky factor U

```
for $k = 1, \ldots$ do
    $R = A - U^T U$
    $X_U = triu(U^{-T} R)$
    Apply numerical dropping to $X_U$
    $U = U + X_U$
    if $det(U) = 0$ then
        Abort "singular $U$"
    end
end
```