Delft University of Technology
Faculty Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Enhancing iterative solution methods for general FEM computations using rigid body modes.

A thesis submitted to the
Delft Institute of Applied Mathematics
as part of

the degree of

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

ALEX SANGERS

Delft, the Netherlands
June 2014

MSc THESIS APPLIED MATHEMATICS

"Enhancing iterative solution methods for general FEM computations using rigid body modes."

ALEX SANGERS

Delft University of Technology

Daily supervisor

Dr.ir. M.B. van Gijzen

Responsible professor

Prof. dr.ir. C. Vuik

Other thesis committee members

Dr.ir. G.M.A. Schreppers (TNO DIANA)

Dr. J.L.A. Dubbeldam

June 2014

Delft, the Netherlands

i Abstract

The demand for large nonlinear finite element models in the field of Civil engineering grows every year. The finite element software has to facilitate the analysis of larger models in less time. One of the computationally most intensive parts of a finite element analysis is the solution of one or more systems of linear equations. Iterative solution methods have proved to be efficient for some classes of applications.

DIANA (DIsplacement ANAlyzer) is a general finite element software package that can be used to analyze a wide range of problems arising in Civil engineering. This thesis focuses on improving the convergence of iterative solution methods of general finite element applications. The convergence of iterative solvers stagnates if some eigenvalues are relatively small. Large stiffness jumps in the underlying model, such as significant material differences, cracking or other nonlinear behavior, can result in such harmful eigenvalues.

The considered remedy is based on the approximate rigid body modes of the model. To identify the approximate rigid body modes in a finite element application we propose a generally applicable method based on element stiffness matrices. Furthermore, we propose a strategy for reusing the rigid body modes in a nonlinear iteration loop. The rigid body modes are used for deflation and coarse grid correction and the performance is compared in a sequential and parallel environment with various number of threads. Both the symmetric case (Conjugate Gradient) and the non-symmetric case (restarted GMRES) are considered.

The proposed methods are implemented in the commercial software package DIANA and are extensively tested. Cases with sudden stiffness jumps of a factor 10^3 or higher can be significantly improved by using approximate rigid body modes. We find that coarse grid correction is more robust than deflation and it also scales better with the number of concurrent threads.

ii Notation and definitions

Below some common mathematical notation and definitions are introduced.

Definition 1. *Let $x \in \mathbb{R}^n$ be a vector and $A \in \mathbb{R}^{n\times n}$ be a matrix. Then A is defined:*

$$\begin{aligned} \textit{Symmetric if} \quad & A = A^T. \\ \textit{Positive definite if} \quad & x^T Ax > 0, \quad \forall x \neq 0. \end{aligned}$$

Definition 2. *Let $x, y \in \mathbb{R}^n$ be vectors and let $A \in \mathbb{R}^{n\times n}$ be a positive definite matrix. Then the following inner products are defined as*

$$\begin{aligned} \langle x, y\rangle_2 &= \sum_{i=1}^{n} x_i y_i = x^T y, \\ \langle x, y\rangle_A &= x^T A y. \end{aligned}$$

Inner products can induce *norms*.

Definition 3. *Let $x \in \mathbb{R}^n$ be a vector, $A \in \mathbb{R}^{n\times n}$ be a positive definite matrix and $p \in \mathbb{N} \cup \{\infty\}$. Then the following commonly used norms are defined by*

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \qquad \|x\|_A = \sqrt{x^T Ax},$$

$$\|A\|_p = \max_{x\in\mathbb{R}^n\setminus\{0\}} \frac{\|Ax\|_p}{\|x\|_p} \qquad \|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2}.$$

The $\|\cdot\|_p$ is often chosen as $\|\cdot\|_1$, $\|\cdot\|_2$ or $\|\cdot\|_\infty$, where

$$\|x\|_\infty := \max_{i=1,\dots,n} |x_i|.$$

The $\|\cdot\|_2$ is called the Euclidean norm or L^2-norm and for the Euclidean matrix norm holds that $\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)}$. The $\|\cdot\|_F$-norm is the Frobenius norm and is often used in numerical computations since it is relatively cheap to compute.

Any norm induced by an inner product satisfies $\|x\|_* = \sqrt{\langle x, x\rangle_*}$. The Euclidean norm is induced by the Euclidean inner product and for positive definite matrices, the A-norm is induced by the A-inner product.

Definition 4. *Let $A \in \mathbb{R}^{n\times n}$ be a matrix. The condition number κ_* of A is defined as*

$$\kappa_*(A) = ||A||_* ||A^{-1}||_*.$$

If there is no ambiguity we write $\kappa(A) := \kappa_2(A)$. Furthermore, if A is symmetric positive definite, this reduces to

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}.$$

The effective condition number of the symmetric (semi-)positive definite matrix A is defined by the following ratio:

$$\kappa_{\mathit{eff}}(A) = \frac{\lambda_{\max}(A)}{\lambda_{m+1}(A)},$$

where $\lambda_{m+1}(A)$ is the smallest positive eigenvalue of A.

Definition 5. *A matrix (or any operator) A is self-adjoint if*

$$\langle Ax, y\rangle = \langle x, Ay\rangle.$$

Note that any symmetric matrix is self-adjoint.

Definition 6. *Let the function f be defined on domain V. Let $k \in \mathbb{N} \cup \{\infty\}$ and $p \in \mathbb{N}$* [1]*. Then the following function spaces are defined as*

$$\begin{aligned}
C(V) &:= \{f \to \mathbb{C} \mid f \textit{ is continuous}\},\\
C^k(V) &:= \{f \to \mathbb{C} \mid f \textit{ is } k\textit{-times continuously differentiable}\},\\
L^p(V) &:= \{f \to \mathbb{C} \mid \int_V |f|^p \, dV < \infty\},\\
H(V) &:= \{f \to \mathbb{C} \mid ||f|| = \sqrt{\langle f, f\rangle} \textit{ is well-defined}\},\\
H^1(V) &:= \{f \in L^2(V) \mid f \textit{ has a weak derivative}\},\\
H^n(V) &:= \{f \in H^1(V) \mid f' \in H^{n-1}(V)\}.
\end{aligned}$$

The function space H is called a Hilbert space and the function space H^n is called a Sobolev space. Any space with a well-defined inner product is a Hilbert space. The mentioned weak derivative above will be elaborated later. In addition, any lower index 0, such as $f \in C_0^1(V)$, indicates that the corresponding function f is zero at the boundary Γ of V.

[1] Typically, $k = \{1, 2, \infty\}$ and $p = \{1, 2\}$.

iii Acknowledgments

This paper presents the results of my Master thesis. The graduation project was performed to obtain my masters degree in Applied Mathematics at the Delft University of Technology and carried out at TNO DIANA in Delft.

I would like to thank some people who supported and helped me during my graduation project. Firstly, I want to thank the graduation committee members, Martin van Gijzen, Gerd-Jan Schreppers, Kees Vuik and Johan Dubbeldam for their support and advice throughout the course of my graduation project.

I would like to thank my colleagues at TNO DIANA for their help with the DIANA software. In particular, I would like to thank Gerd-Jan Schreppers for giving me the opportunity to work on this project and Wijtze Pieter Kikstra, Jonna Manie and Jos Jansen for their know-how advice about DIANA. I would also like to thank Erik Jan Lingen for his feedback on parallel computing and Jantine van Steenbergen for all sorts of practical problems.

The guidance of my supervisor Martin van Gijzen was instructive and cooperating with him was very pleasant.

Finally, I want to thank my family and my girlfriend Iris for their continued support and being a listening ear in the hard-to-understand mathematical world.

Delft, June 2014

Alex Sangers

Contents

1 Introduction

In the field of Civil engineering there is a growing demand for large three-dimensional nonlinear finite element models. One of the computationally most intensive parts of a finite element analysis is the solution of one or more systems of linear equations. Large three-dimensional problems lead to systems with millions of degrees of freedom. To solve these systems a number of direct and iterative solution methods are available [34]. Iterative solution methods have proved to be able to solve these systems in a reasonable time and require less memory than the direct methods. A major drawback is that iterative solvers are not always robust, i.e., convergence can be slow or they may not converge at all. Several techniques are available to increase the robustness of iterative solvers, such as preconditioning.

DIANA (DIsplacement ANAlyzer) is a general finite element software package that can be used to analyze a wide range of problems arising in Civil engineering including structural, geotechnical, tunneling, earthquake disciplines and oil and gas engineering [40]. The purpose of this research is to get more insight in the iterative solvers of DIANA and how to improve them. This thesis addresses the following research question:

> *How can the iterative solution methods of* DIANA *be improved by incorporating the physical properties of the model?*

The solvers are implemented in a general commercial finite element software package, which requires that non-specialist users should be able to use it. Therefore, any improvements of the iterative solvers should be easy to use and generally applicable. Furthermore, the implementation should be easy to understand and easy to maintain.

The convergence of the iterative solvers stagnates if some eigenvalues are relatively small. Large stiffness jumps in the underlying model, such as material differences, cracking or other nonlinear behavior, can result in such harmful eigenvalues. The considered remedy is based on the stiff parts in a model, the so-called approximate rigid body modes [20]. This approach aims to increase the robustness and the convergence of an iterative solution method in case large jumps in stiffness occur. Examples are different material layers in a geotechnical model or nonlinear material behavior in tunneling models. Furthermore, the rigid body modes can improve global convergence in the context of parallel computing [28].

The rigid body modes can be used for coarse grid correction or for deflation. Coarse grid correction is an algebraic multigrid method to enhance the convergence of low frequency modes of a model. Deflation is a projection technique similar to preconditioning, with the purpose to increase the robustness and the convergence speed of an iterative solver. Deflation is very suitable in combination with a preconditioner and has shown to be successful in the application of structural mechanics and composite materials [20–23, 43].

This thesis builds on the work of Lingen [28], who implemented coarse grid correction in DIANA based on the subdomains of a domain decomposition. This works also strongly relates to the work of Jönsthövel [20], who described the rigid body modes deflation technique applied to composite materials. Some theoretical background of the finite element method is based on Bathe [2] and Zienkiewicz [52] and the information available in the DIANA manuals [40–42]. Background theory of iterative solution methods can be found in Saad [34] and a starting point for deflation theory is Frank et al. [12].

The outline of the remainder of this thesis is as follows:

Section 2: The finite element method. The displacement based finite element method is introduced.

Section 3: Iterative solution methods for linear systems. The iterative solution methods, preconditioners and other solution techniques that are currently available in DIANA are described.

Section 4: Domain decompositions. The domain decomposition techniques substructuring (Schur complement) and parallel Schwarz domain decomposition are explained.

Section 5: Deflation. The deflation technique to improve the iterative solver is introduced.

Section 6: Identifying rigid bodies. This section explains the idea of (approximate) rigid bodies in the model and describes how to identify and reuse the rigid bodies.

Section 7: Rigid bodies within the solution method. Both deflation and coarse grid correction can take advantage of the rigid body modes.

Section 8: Implementation. Some implementation considerations are illustrated and motivated.

Section 9: Results. This section shows the results for the iterative solution methods when applying the proposed techniques. A comparison is made with the current solvers in DIANA.

Section 10: Conclusions. The summary, conclusions and recommendations of this thesis are presented.

2 The finite element method

The finite element method (FEM) is a multi-purpose numerical method to solve involved partial differential equations (PDEs). Finite element procedures are very widely used in engineering analysis and an important advantage of the FEM over other numerical methods is its broad applicability. A general but extensive introduction to the FEM can be found in the historically significant books of Bathe [2] and Zienkiewicz [52].

Partial differential equations arising from physics can often be written as a minimization problem [46]. Such minimization problems typically aim to minimize the underlying potential energy or seek the shortest path. The advantage of minimization problems is that they admit a larger solution class than the corresponding PDE formulation; they require fewer boundary conditions. The boundary conditions explicitly described in the minimization problem are called *essential* boundary conditions. Other boundary conditions that are present in the PDE but are absent in the minimization problem are called *natural* boundary conditions. These natural boundary conditions are only implicitly present in the formulation of the minimization problem. As a rule of thumb, for second order PDEs all boundary conditions regarding to the displacement vector u are essential and all boundary

conditions regarding to the spatial derivative of u are natural.

Another, more general approach, is the weak formulation. The weak formulation as well as the minimization problem allow a larger solution class than the corresponding PDE formulation. The weak formulation is identical to the minimization problem *if* the minimization problem can be formulated. Although of historical relevance and linked with physical meaning, the minimization problem formulation is only applicable in specific cases. For more information on the minimization problem formulation, please refer to Van Kan et al. [46]. This thesis will mainly focus on the weak formulation.

2.1 The weak formulation

The goal of the weak formulation is to admit a larger solution class by using the concept of weak derivatives.

Definition 7. *The function $g \in L^2(V)$ is the weak derivative of $f \in L^2(V)$ if g satisfies*

$$\int_V g(s)\lambda(s)\ ds = -\int_V f(s)\lambda'(s)\ ds, \quad \forall \lambda \in C_0^1(V). \tag{2.1}$$

Note that the (strong) derivative f' of f is also the weak derivative and that the weak derivative g of f is also the (strong) derivative f' of f *if it exists*.
An example of a function with a weak derivative without a strong derivative is $f = |x|$ on $V = [-1, 1]$. The strong derivative f' of f does not exist in $\{0\}$, while the weak derivative g is given by

$$g(x) = \begin{cases} -1 & \text{if } x \in (-1, 0], \\ 1 & \text{if } x \in (0, 1). \end{cases}$$

The function g is well-defined and the choice of the value of g at $x = 0$ is irrelevant, since the weak derivative is only defined up to point-wise almost-everywhere equivalence [19]. The function g is equal to the strong derivative of f on $V\backslash\{0\}$. Furthermore, since the function space $C^1(V)$ is $||.||_{H^1}$-dense in the function space $H^1(V)$ [16], Equation (2.1) also holds for all $\lambda \in H_0^1(V)$.

The strength of the weak formulation can be illustrated by considering the following Poisson problem on V with $f \in \mathrm{L}^2(V)$:

$$\begin{cases} -\bigtriangledown^2 u = f, & \text{on } V, \\ u = 0, & \text{on } \Gamma = \partial V. \end{cases} \tag{2.2}$$

The PDE notation demands that u is twice differentiable. Now consider the weak formulation approach:

$$-\bigtriangledown^2 u = f \tag{2.3}$$

$$\int_V -\bigtriangledown^2 u\ \lambda\ dV = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V) \tag{2.4}$$

$$\int_V \bigtriangledown u \cdot \bigtriangledown \lambda - \bigtriangledown \cdot (\bigtriangledown u\ \lambda)\ dV = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V)$$

$$\int_V \bigtriangledown u \cdot \bigtriangledown \lambda\ dV - \oint (\bigtriangledown u\ \lambda) \cdot \mathbf{n}\ d\Gamma = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V) \tag{2.5}$$

$$\int_V \bigtriangledown u \cdot \bigtriangledown \lambda\ dV = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V). \tag{2.6}$$

Equation (2.4) results from multiplying with a test function $\lambda \in H_0^1(V)$, which satisfies the essential boundary conditions of u, and integrating on the whole domain. This is equivalent to Equation (2.3) by the extension of DuBois-Reymond's lemma [46]. Equation (2.5) follows by Gauss divergence theorem and Equation (2.6) follows from the boundary conditions of λ.

The weak formulation approach results in a lower order problem (in derivatives) with the same unique solution as the original PDE. It admits a larger solution class by only demanding that $u \in H_0^2(V)$. The weak formulation is a generalization of the corresponding PDE; a solution of the PDE is also a solution of the weak formulation, but not necessarily vice versa.

2.2 Solving the weak formulation

The finite element method solves Equation (2.6) by approximating u by a linear combination of so-called test functions, i.e.,

$$u \approx u^n = \sum_{j=1}^{n} u_j \lambda_j,$$

where λ_j are test functions. The domain V is divided into n_{el} elements with each element consisting of a number of (shared) nodes. The number of nodes per element varies with the specific choice of test functions. The choice of the test functions $\lambda_j \in H_0^1(V)$ strongly determines the sparsity of the resulting linear system of equations. In order to preserve the underlying model sufficiently accurate, the test functions λ_j always satisfy $\lambda_j(x_i) = \delta_{ij}$. This ensures that $u(x_i) = u_i$ in the nodes.

Reconsider the Poisson problem in Equation (2.2). Independent of the specific choice of λ_j, the approximation u^n in our example leads to:

$$\begin{aligned}
\int_V \bigtriangledown u^n \cdot \bigtriangledown \lambda_i \; dV &= \int_V f \lambda_i \; dV \\
\int_V \bigtriangledown (\sum_{j=1}^{n} u_j \lambda_j) \cdot \bigtriangledown \lambda_i \; dV &= \int_V f \lambda_i \; dV \\
\sum_{j=1}^{n} u_j \int_V \bigtriangledown \lambda_j \cdot \bigtriangledown \lambda_i \; dV &= \int_V f \lambda_i \; dV.
\end{aligned}$$

Assume the number of elements to be n_{el} and denote the elements by e_m. Let us introduce

$$\begin{aligned}
K_{ij} &= \int_V \bigtriangledown \lambda_j \cdot \bigtriangledown \lambda_i \; dV \\
&= \sum_{m=1}^{n_{el}} \int_{e_m} \bigtriangledown \lambda_j \cdot \bigtriangledown \lambda_i \; dV = \sum_{m=1}^{n_{el}} K_{ij}^{e_m}, \\
f_i &= \int_V f \lambda_i \; dV \\
&= \sum_{m=1}^{n_{el}} \int_{e_m} f \lambda_i \; dV = \sum_{m=1}^{n_{el}} f_i^{e_m}.
\end{aligned} \tag{2.7}$$

By choosing test functions, this notation leads to:

$$\sum_{j=1}^{n} K_{ij} u_j = f_i, \quad \forall i = \{1, \ldots, n\},$$
$$K\underline{u} = f. \tag{2.8}$$

The solution $\underline{u}$ of the linear system in Equation (2.8) can be found by a direct or iterative solution method. The solution u of the original PDE in Equation (2.2) is approximated by the solution $\underline{u} = \begin{pmatrix} u_1 & \ldots & u_n \end{pmatrix}^T$ found in Equation (2.8) by $u \approx u^n = \sum_{j=1}^{n} u_j \lambda_j$.

2.3 Application of the finite element method to structural problems

In a structural problem the displacements u, strain ε and stress σ are the unknowns of interest. The displacements can be directly analyzed in a finite element method as in DIANA. The formulation of the displacement based finite element method is extensively described in Bathe [2].

Strain is a two-dimensional tensor and it is a measure of deformation, representing the displacements between particles relative to a reference length. It is therefore a dimensionless quantity. Strain is often expressed as an array (engineering notation) for convenience [49]. It consists of normal components (diagonal terms in the tensor) and shear components (off-diagonal terms). Strain is expressed as a function of the derivative of the displacements.

Stress is also a two-dimensional tensor and is a measure of the internal forces per area that particles exert on each other. Stress is therefore of dimension force per area. Any strain generates a stress in the linear elastic case, as a reaction on the deformation. Stress can also occur due to the environment, for example when a solid vertical bar supports a hanging weight. Stress may even exist when strain is absent, or when no external forces occur (e.g. with so-called pre-stress). Stress is expressed in the stress-strain relation. Often, the stress is expressed as an array (engineering notation) for convenience [49].

Consider a static structural problem. The local strain ε can be calculated by $\varepsilon = B_m u$, with B_m the local strain-displacement (differential) matrix. The stress corresponding to ε are given by $\sigma = D_m(\varepsilon)$, with D_m the local stiffness relation. Assuming linear elastic behavior, this can be written as $\sigma = D_m \varepsilon$, with D_m the local rigidity matrix. This rigidity matrix depends on material properties such as Young's modulus E and Poisson's ratio ν.

In element formulations the displacements u, strain ε and stress σ are locally formulated for each element using the interpolation matrix N_m. This matrix N_m is determined by the test function λ as introduced in Equation (2.4). The local matrices B_m depend on N_m and vary from element to element. The matrix T_m maps the local element numbering to the global numbering. In conclusion, the local stiffness matrices K^{e_m} and the global stiffness matrix K are formed by:

$$K^{e_m} = \int_{e_m} B_m^T D_m B_m \, dV,$$
$$\Rightarrow K = \sum_{m=1}^{n_{el}} T_m^T K^{e_m} T_m,$$

where n_{el} is the number of elements. In essence this matrix formulation is a special case of the general weak formulation approach.

2.4 Elements

Elements model local physical behavior. This behavior can consist of extensive material properties, friction, contact between faces and more [41]. Therefore, a large number of element types and element shapes exist. All elements consist of a number of nodes and corresponding degrees of freedom. Nodes in three dimensional models consists up to: three translational degrees of freedom, three rotational degrees of freedom and a number of scalar degrees of freedom. These scalar degrees of freedom can for example be temperature, pressure or Lagrange multipliers. Geometrically speaking, DIANA offers: nodal point, line, triangle, quadrilateral, pyramid, wedge and brick elements. For structural analysis DIANA provides a number of structural elements, such as continuum, interface, spring, and mixture elements.

2.4.1 Structural elements

Three-dimensional structural elements usually consist of three translational degrees of freedom per node. Standard type structural elements consists of a geometry and a material, the latter at least described by Young's modulus E and Poisson's ratio ν [42]. Young's modulus E indicates the material elasticity property. The Young's modulus of a material can be used to calculate the stress it exerts under specific strain. Poisson's ratio ν indicates the ratio of material deformation in the plane perpendicular to the direction of the exerting compression or stretching. Assuming linear elasticity, the following three-dimensional relation holds

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{zx} \\ \sigma_{xy} \end{pmatrix} = D_m \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{yz} \\ \varepsilon_{zx} \\ \varepsilon_{xy} \end{pmatrix}, \tag{2.9}$$

where the stress σ and strain ε are two-dimensional tensors expressed as arrays and where the entries of the rigidity matrix D_m depend on e.g. Poisson's ratio ν and Young's modulus E. Equation (2.9) is the three-dimensional generalization of Hooke's law for linear elastic material. In general, there are 36 matrix components. In many applications the components of matrix D_m may lose independence. This reduces the number of independent components to 21 in the symmetric case (anisotropic), 9 (orthotropic), 5 (transverse isotropic) or 2 (isotropic) [52]. Considering the one-dimensional case, Hooke's law reduces to:

$$\begin{aligned} \sigma &= E\epsilon, \\ F &= A\sigma = \frac{EA}{L}\Delta u, \end{aligned} \tag{2.10}$$

where F is the force, A is the cross-sectional area through which the force is applied, L is the original length of object and Δu is the relative displacement.

In structural mechanics it is common to use finite elements such as beam, plate and shell elements. These elements are introduced in situations where classical elements perform poorly, e.g., the underlying problem is governed by fourth-order equations. Therefore, the

shape of the elements, the degrees of freedom and the test function λ have to be adapted. These elements restrict the local rigidity matrix D_m by assumptions on the stress-strain relation.

Rotational degrees of freedom are typical for special elements, such as shell elements [41]. The essence of shell elements is that they are planar (although they may be curved in that plane). In general two hypotheses hold:

- *Straight-normals.* Particles that are originally on a straight line remain on a straight line during deformations.
- *Zero-normal-stresses.* The stress through the thickness of the shell is zero.

In each node of a shell element occur five (or six) degrees of freedom: three translational degrees of freedom and two (or, if drilling rotations are included, three) rotational degrees of freedom. Many principals of shells are described in Zienkiewicz [52].

2.4.2 Interface elements for structural analysis

DIANA offers three families of interface elements, namely structural interfaces, contact elements and fluid-structure interfaces. Interface elements are placed between nodes, lines and/or planes with special properties. Typical applications of structural interface elements are elastic bedding, nonlinear elastic bedding, discrete cracking, bond-slip along reinforcement, friction between surfaces, joints in rock, masonry and so on [42]. Moreover , structural interface elements can be pre-stressed.

Contact elements model zones of possible contact. There are two types of contact elements: surface containing contact elements and surface containing target elements. Contact elements can result in poor performance of the solution method and are avoided as much as possible by using structural interface elements when possible.

At fluid-structure interface elements the additional pore pressure potential degrees of freedom are one-side added. The fluid-structure interface elements are used in fluid-structure interaction analysis, coupling the fluid domain to the structure via pressure of the fluid and the normal displacement of the structure.

The behavior of interface elements is nonlinear in general. For example, in cracking the interface elements will act linearly at the beginning, but as cracking starts to take place, the nonlinear behavior will become dominant. The transition of this behavior is hard to compute, and in general more iterations per nonlinear loop and smaller increments are required during the initiation of a crack.

The input for DIANA for interface elements are not Young's modulus and Poisson's ratio, but the elastic rigidity D and depending on the application, the stiffness can be specified per direction and can depend on maturity, temperature, friction, etc. [42]. The diagonal entries of D need always to be specified. Assuming linear elasticity and an uncoupled system, the following three-dimensional relation is given:

$$\begin{pmatrix} \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} = \begin{pmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{pmatrix} \begin{pmatrix} \Delta_x u \\ \Delta_y u \\ \Delta_z u \end{pmatrix}, \tag{2.11}$$

where τ is the traction vector (equivalent to the stress σ), D is the stiffness relation and Δu is the relative displacement. Equation (2.11) is called Hooke's law for linear elastic material. Considering the one-dimensional case, Hooke's law reduces to

$$\begin{aligned} \tau &= D\Delta u, \\ F = \tau A &= DA\Delta u, \end{aligned} \tag{2.12}$$

where F is the force and A is the cross-sectional area through which the force is applied. Comparing the one-dimensional stiffness of a classical structural element in Equation (2.10) with an interface element in Equation (2.12) gives the following one-dimensional relation:

$$\begin{aligned} F &= \left(\frac{EA}{L}\right)\Delta u = (DA)\Delta u, \\ \Rightarrow \frac{E}{L} &= D. \end{aligned} \tag{2.13}$$

Thus, to compare the two different types of elements, one should compare the interface element's stiffness D with the classical structural element's Young's modulus and original length.

2.4.3 Spring elements

Spring elements act as continuous dampers in the finite element model between two nodes or model the interaction of the finite element model with its environment [42]. Therefore, a spring element can consist of one or two nodes. A spring element requires the spring 'constant' k, which may depend on the relative displacements. Spring elements can model springs and/or dashpots in both translational or rotational direction. A spring element often models one-dimensional elasticity and for linear static analysis the following relation holds (Hooke's law):

$$F = k\Delta u,$$

where F is the force, k is the spring stiffness and Δu is the relative displacement.
Comparing the spring stiffness with classical structural element stiffness gives the following one-dimensional relation:

$$\begin{aligned} F = k\Delta u &= \left(\frac{EA}{L}\right)\Delta u, \\ \Rightarrow k &= \frac{EA}{L}. \end{aligned} \tag{2.14}$$

Thus, to compare the two different types of elements, one should compare the spring element's spring stiffness with the classical structural element's Young's modulus, the area through which the force is applied and its original length.

2.4.4 Mixture elements

If deformation affects the pore pressures, one may extend a structural element with pore pressure potential degrees of freedom. These elements are called mixture elements. All DIANA 's regular plane strain, axisymmetric and solid structural elements can be extended to mixture elements, adding a scalar pore pressure potential degree of freedom to each element node. Also interface elements can be mixture elements.

In static analysis, the time derivatives are zero, yielding only a single-sided coupling between stress and flow (flow influences stress only). In a dynamic analysis there is a two-sided coupling. The pressure degrees of freedom are often of a different order of magnitude than the translational degrees of freedom. Details of mixture elements can be found in DIANA User's Manual, Analysis Procedures [40], Section 60.2.

2.5 Element integration

The element integrals K_{ij} and f_i as in Equation (2.7) can be calculated using exact or numerical integration. Often exact integration cannot be done. Numerical integration is typically done by Newton-Cotes, composite Simpson, Lobatto or Gauss integration [40] in the following way:

$$\int_{e_m} f \; dV = \sum_{i=1}^{n_\xi} w_{\xi_i} f(\xi_i),$$

where ξ_i are the integration points, w_{ξ_i} is the weight function of the integration scheme and n_ξ is the number of integration points. The number and location of required integration points depends on the used integration scheme and the order of the test function.

2.6 Nonlinear analysis

In nonlinear finite element analysis the relation between the force vector f and the vector u is no longer linear. The general behavioral description $F(u) = 0$ cannot be reformulated to $Ku = f$ as in the linear case. A common approach to solve $F(u) = 0$ is Newton's method. Newton's method takes an initial guess u^0 and then determines an improved solution u^{k+1} by a Taylor expansion in the neighborhood of u^k.

$$F(u^{k+1}) = F(u^k) + J(u^k)(u^{k+1} - u^k) + \mathcal{O}((u^{k+1} - u^k)^2),$$

where $J = \frac{\partial F}{\partial u}$. Setting $F(u^{k+1}) = 0$ and neglecting second order terms results in the following iteration scheme:

$$\begin{aligned} J(u^k)v^k &= -F(u^k), \\ u^{k+1} &= u^k + v^k. \end{aligned} \tag{2.15}$$

Note that u, v and F are vectors and J is the Jacobian matrix. Forming and solving the linear system in Equation (2.15) is the hard part.

Newton's method is effective, but nonlinear behavior can results in very small eigenvalues of the stiffness matrix and furthermore, the computation of the Jacobian J is very time consuming. A number of alternative iterative approximations are available in DIANA, namely Modified Newton, Quasi-Newton and linear and constant stiffness. Furthermore, continuation and line search are used to speed up these nonlinear iterative methods.

Modified Newton uses only the initial Jacobion matrix $J(u^0)$ so that each nonlinear iteration is cheap. Of course, in general more iterations are needed with Modified Newton. Quasi-Newton methods, such as BFGS [4] and Crisfield [6], use information of previous iterations to achieve better approximations than Modified Newton. The linear stiffness method uses the initial linear stiffness matrix all the time (also for successive states, e.g. in time)

and is therefore very cheap per iteration (using a direct solution method) but yields slow convergence in general. The constant stiffness method uses the constructed stiffness matrix of another method, keeping it constant from that point on (also for successive states). The constant stiffness method also yields very cheap iterations but slow convergence in general.

Speeding up these iterative methods can be done by continuation and line search. Continuation assumes relative continuous deformation, so that the previous increment is a first prediction of the current increment. The line search algorithm is useful if the prediction is far from the equilibrium, e.g., if strong nonlinearities take place. The line search algorithm determines the amplification factor of the direction of the nonlinear iterative method.

In DIANA a nonlinear analysis is performed by using load or time stepping. In essence these two types of stepping are similar: they both define a sequence of states. The following problem illustrates how to solve a nonlinear problem using stepping. The following equation represents a nonlinear spring satisfying Hooke's law.

$$k(u)u = f, \tag{2.16}$$

where $k(u) = (1 - u)k_0$. The solution u of Equation (2.16) requires solving the nonlinear problem

$$F(u) = -k_0u^2 + k_0u - f = 0.$$

In the light of load stepping, this means that the first load step (right-hand side vector) of Equation (2.15) is initialized at zero and increased with $-F(u^0)$. If the right-hand side vector is too large or if the model is strongly nonlinear, the nonlinear iterations could converge slowly or not at all. This can be solved by applying several load steps to incrementally increase the right-hand side f of Equation (2.16).

In many applications several nonlinear PDE's need to be solved. The solutions can also affect subsequent solutions in next time/load steps, e.g., material elasticity can change after deformation as in Equation (2.16). This results in a nonlinear stress-strain relation. Suppose the solution vector u_m^k at time/load step k is converged after m nonlinear iterations. By applying the continuation technique, the solution vector u_m^k does not need to be reset after convergence of the nonlinear iteration method, but can be scaled and used as initial solution vector u_0^{k+1} at time/load step $k + 1$.

3 Iterative solution methods for linear systems

The analysis of a finite element analysis requires to solve at least one linear system of equations $Ku = f$. In general, this system of equations is too large to be solved by directly computing K^{-1}. Two classes of solution methods exist to solve a system effectively, namely direct and iterative solution methods, also called direct and iterative solvers. This thesis focuses on the iterative solution methods due to its attractive properties for large three-dimensional problems. Two category of iterative solution methods for solving $Ku = f$ exist.

The first category of iterative solvers is called Basic Iterative Methods and they are based on a splitting $K = P - N$, followed by the iteration scheme

$$u_{m+1} = u_m + P^{-1}r_m, \tag{3.1}$$

with $r_m = f - Ku_m$ the residual and u_0 is an initial guess. The matrix P should resemble K in some way and it should be easy to solve $Px = y$. Typical resulting methods are (damped) Jacobi, Gauss-Seidel and SOR(ω). For increasing size of K, the Basic Iterative Methods can converge very slowly [48].

The second category of iterative solvers is called Krylov subspace methods. This category is widely popular and often more effective than the Basic Iterative Methods.

3.1 Krylov subspace methods

Krylov subspace methods can be used to solve large systems of linear equations or to find eigenvalues, without performing matrix-matrix operations. Many different iterative solution methods are based on Krylov subspaces. This thesis mainly focuses on (the derivation of) the well-known methods Conjugate Gradient (CG) and Generalized Minimal Residual (GMRES) [34], since these methods are currently available in DIANA.

Krylov subspace methods yield two m-dimensional subspaces: $\mathcal{K}^m$, which is also called the solution space, and $\mathcal{L}^m$, which is called the constraints space. A Krylov subspace method uses the span of the vectors in subspace $\mathcal{K}^m$ to reduce the residual $r_m = f - Ku_m$, while r_m should be orthogonal to $\mathcal{L}^m$. Reformulated, this implies

$$\text{Find } u_m \in u_0 + \mathcal{K}^m \text{ such that } f - Ku_m \perp \mathcal{L}^m. \tag{3.2}$$

The exact solution u is approximated by $u_m \in \mathcal{K}^m$. In order to find an approximation u_m satisfying Equation (3.2), the bases V_m for $\mathcal{K}^m$ and W_m for $\mathcal{L}^m$ have to be constructed. The approximation u_m can be computed by $u_m = u_0 + V_m y_m$ such that $W_m^T(f - Ku_m) = 0$.

The Krylov subspace $\mathcal{K}^m$ is based on a polynomial of degree $m-1$. The choice for the polynomial approximations strongly determines the success of the Krylov method. The Krylov subspace is defined as $\mathcal{K}^m(K; r_0) = \text{span}\{r_0, Kr_0, \ldots, K^{m-1}r_0\}$, which is the mth-order Krylov subspace generated by matrix K with starting vector r_0. If no ambiguity occurs, this is shortly denoted by $\mathcal{K}^m$.

Arnoldi's procedure is an algorithm for building an orthonormal basis of the Krylov subspace $\mathcal{K}^m(K; v_1)$ [34].

Algorithm 1. ***Arnoldi***

Choose a vector v_1, such that $\|v_1\|_2 = 1$
For $j = 1, 2, \ldots, m$ Do:
  $h_{i,j} = \langle Kv_j, v_i \rangle$ *for* $i = 1, 2, \ldots, j$
  $w_j = Kv_j - \sum_{i=1}^{j} h_{ij} v_i$
  $h_{j+1,j} = \|w_j\|_2$
  If $h_{j+1,j} = 0$ *then Stop*
  $v_{j+1} = w_j / h_{j+1,j}$
EndDo

At iteration j this algorithm multiplies the vector v_j with K and orthonormalizes the resulting vector w_j with respect to all previous v_i by a Gram-Schmidt procedure. The Arnoldi algorithm stops if $w_j = 0$. The resulting vectors $v_1, v_2, \ldots, v_m$ are equal to the orthonormalized (with respect to each other) vectors $v_1, Kv_1, \ldots, K^{m-1}v_1$. This orthonormal property is very useful, which will be elaborated later. This version of Arnoldi uses a

Gram-Schmidt procedure, but due to rounding errors often a more stable method is used, such as modified Gram-Schmidt or Householder reflection.

Let the entries of matrix $\bar{H}_m$ be given by h_{ij} at the m-th iteration in Algorithm 1. The resulting matrix $\bar{H}_m \in \mathbb{R}^{(m+1)\times m}$ is an upper-Hessenberg matrix. This is a matrix with only non-zero entries h_{ij} for $j = i-1, i, \ldots, m$. Let us also define $V_m = [v_1 \cdots v_m]$, and H_m obtained from $\bar{H}_m$ by deleting its last row, so

$$H_m = \begin{pmatrix} h_{11} & \cdots & \cdots & \cdots & h_{1m} \\ h_{21} & h_{22} & \cdots & \cdots & h_{2m} \\ & h_{32} & \ddots & \cdots & h_{3m} \\ & & \ddots & \ddots & \vdots \\ & & & h_{m,m-1} & h_{mm} \end{pmatrix}.$$

The following equalities hold:

$$KV_m = V_m H_m + w_m e_m^T \tag{3.3}$$
$$= V_{m+1}\bar{H}_m \tag{3.4}$$
$$V_m^T K V_m = H_m \tag{3.5}$$

From Algorithm 1 follows that

$$v_{j+1}h_{j+1,j} = w_j = Kv_j - \sum_{i=1}^{j} h_{ij}v_i,$$
$$\Rightarrow Kv_j = \sum_{i=1}^{j+1} h_{ij}v_i.$$

Rewriting in matrix formulations leads to Equation (3.4). Equation (3.3) follows by step 4 in Algorithm 1, where w_j is orthogonal with respect to all previous v_i. By premultiplying Equation (3.3) with V_m^T and using orthonormality of its columns follows Equation (3.5).

The following subsections describe different iterative solution methods that can be derived from Arnoldi's procedure or its symmetric variant, the Lanczos procedure. The first described method is the Full Orthogonalization Method (FOM). It is not used in DIANA, but acts as an introduction to the Generalized Minimimal Residual method (GMRES) and the Conjugate Gradient method (CG). The FOM solves non-symmetric problems by orthogonalizing the residuals with respect to each other. The solution and constraints spaces are chosen to be $\mathcal{K}^m = \mathcal{L}^m$. The CG method applies the same strategy for symmetric problems. GMRES solves non-symmetric problems by minimizing the residual and takes $\mathcal{K}^m = K \cdot \mathcal{L}^m$. The Conjugate Residual method (CR) applies the same strategy for symmetric problems, but this method is less popular and will not be further discussed in this report. The methods are graphically classified in Figure 1.

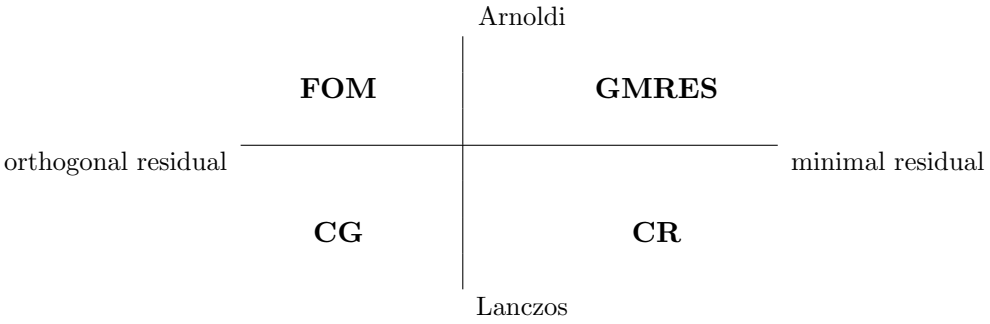


Figure 1: Krylov-based iterative solution methods.

More Krylov-based iterative solution methods have been developed and some of them will be shortly addressed.

3.1.1 Full Orthogonalization Method

The Full Orthogonalization Method uses the Arnoldi procedure with solution space and constraints space $\mathcal{K}^m = \mathcal{L}^m$. The Arnoldi procedure becomes particularly interesting if Algorithm 1 is initialized with $v_1 = r_0/||r_0||_2 := r_0/\beta$. Now, for any vector $u_m \in u_0 + \mathcal{K}^m(K; r_0)$ there is a vector y_m of appropriate length such that $u_m = u_0 + V_m y_m$.

The challenge is to find y_m such that the residual corresponding to the calculated u_m satisfies $r_m \perp \mathcal{L}^m$. It follows from $KV_m = V_{m+1}\bar{H}_m$ that

$$\begin{aligned} r_m = f - Ku_m &= f - K(u_0 + V_m y_m) \\ &= r_0 - KV_m y_m \\ &= \beta v_1 - V_{m+1}\bar{H}_m y_m \\ &= V_{m+1}(\beta e_1 - \bar{H}_m y_m). \end{aligned} \tag{3.6}$$

The residual is orthogonalized with respect to the current Krylov subspace $\mathcal{K}^m(K; r_0)$. The approximate solution u_m can be found by solving

$$\begin{aligned} y_m &= H_m^{-1}\beta e_1, \\ u_m &= u_0 + V_m y_m. \end{aligned} \tag{3.7}$$

To determine whether the solution u_m is sufficiently accurate, Equation (3.6) is reduced to

$$\begin{aligned} r_m = f - Ku_m &= \beta v_1 - V_{m+1}\bar{H}_m y_m \\ &= \beta v_1 - V_m H_m y_m - h_{m+1,m} e_m^T y_m v_{m+1} \\ &= -h_{m+1,m} e_m^T y_m v_{m+1}, \end{aligned} \tag{3.8}$$

by $H_m y_m = \beta e_1$. Taking the norm of Equation (3.8) yields $||r_m||_2 = |h_{m+1,m} y_m(m)|$, which is cheap to evaluate.

Furthermore, as a consequence of Arnoldi's procedure on r_0/β, all residuals r_m are mutually

orthogonal,

$$\begin{aligned} & r_m = f - Ku_m = -(h_{m+1,m} e_m^T y_m) v_{m+1} \\ \Rightarrow\ & r_m \in \text{span}\{v_{m+1}\} \\ \Rightarrow\ & r_m \perp \text{span}\{v_1, \ldots, v_m\} \in \mathcal{K}^m \\ \Rightarrow\ & r_m \perp r_i, \quad \forall i \neq m. \end{aligned}$$

The FOM subsequently orthogonalizes all residuals and computes u_m by Equation (3.7).

3.1.2 Generalized Minimal Residual Method

The full GMRES procedure can be motivated by the FOM and the Arnoldi procedure [35]. Its strategy is to set $\mathcal{L}^m = K \cdot \mathcal{K}^m$. Theorem 1 shows that this is equivalent to minimizing the residual [1].

Theorem 1. *The condition $f - Ku_m = r_m \perp \mathcal{L}^m = K \cdot \mathcal{K}^m$ is equivalent to*

$$\|r_m\|_2 = \min_{u \in u_0 + \mathcal{K}^m} \|f - Ku\|_2 = \min_{r \in r_0 + \mathcal{L}^m} \|r\|_2.$$

Proof. Write $r = r_0 + l$ with $l \in \mathcal{L}^m$ and define P_m as the orthogonal projector onto $\mathcal{L}_m$ and $Q_m = I - P_m$. Then we can decompose the initial residual r_0 according to $r_0 = (P_m + Q_m) r_0$, so r can be decomposed as

$$r = r_0 + l = (P_m r_0 + l) + Q_m r_0.$$

Taking the Euclidean norm and applying Pythagoras yields

$$\|r\|_2^2 = \|P_m r_0 + l\|_2^2 + \|Q_m r_0\|_2^2.$$

The minimum r_m is attained for $l = -P_m r_0$ resulting in

$$r_m = r_0 - P_m r_0 = Q_m r_0 \perp \mathcal{L}^m.$$

Reversed reasoning implies that if $r \perp \mathcal{L}^m$, then $r = Q_m r$ so that $\|r\|_2^2 = \|Q_m r\|_2^2 = \|Q_m r_0\|_2^2$.

□

In the light of Equation (3.6), let us define the following operator

$$J(y_m) = ||f - Ku_m||_2 = ||f - K(u_0 + V_m y_m)||_2.$$

To solve the system $Ku = f$ it is clear that minimizing the Euclidean norm of the residual, $J(y_m)$, could be an advantageous strategy. Recall that at iteration m holds $f - Ku_m = V_{m+1}(\beta e_1 - \bar{H}_m y)$ from Equation (3.6). Taking the norm yields by orthonormality

$$J(y) = ||\beta e_1 - \bar{H}_m y||_2. \tag{3.9}$$

The GMRES method computes after convergence the solution u_m of the minimization problem in Equation (3.9) by

$$\begin{aligned} y_m &= \underset{y}{\text{argmin}} \, ||\beta e_1 - \bar{H}_m y||_2 \\ u_m &= u_0 + V_m y_m. \end{aligned}$$

The Hessenberg matrix H_m has eigenvalues that approximate the eigenvalues of the matrix K. These eigenvalues are called the Ritz values. Since H_m typically is much smaller than K, only the extreme eigenpairs are approximated. Thereafter, the corresponding approximate eigenvectors can be computed. This information can be reused in the nonlinear loop by using deflation of the smallest eigenvectors, as will be discussed in Section 5.

Full GMRES has optimal properties based on $\mathcal{K}^m(K;.)$, but has long recurrences. The iteration process is ceased as soon as the solution has converged. Meanwhile, each Krylov vector has to be stored and is used in each iteration, resulting in more CPU time. Furthermore, the required computer memory can grow to an enormous amount. *Restarted GMRES* or GMRES(s) bounds the number of iterations up to s iterations and thereafter $u_0 := u_s$ is used to restart GMRES. Restarted GMRES computes the approximate solution after convergence or if the memory requirements of the Krylov vectors exceed a certain threshold.

In DIANA the restarted GMRES method is available with a modified Gram-Schmidt orthogonalization procedure. GMRES is restarted if 50% of the memory used by the system matrix and the preconditioner is used for the Krylov vectors.

3.1.3 On other non-symmetric iterative methods

For general (non-symmetric) matrices K there exist many iterative solution algorithms based on Krylov subspaces. A small selection of popular choices is Bi-CGSTAB, IDR(s), GMRES and restarted GMRES [34]. For non-symmetric matrices it is impossible to combine the advantageous properties optimality and short-recurrence. The Bi-CGSTAB, IDR(s) and restarted GMRES algorithm are not optimal (they do not guarantee to decrease the residual over an always-increasing subspace) and full GMRES has long recurrences. This implies that GMRES is preferable if the solution converges relatively fast, but as soon as a restart is required due to memory issues, another short-recurrence, non-optimal method could be preferable.

In Appendix D the IDR(s) method [38, 45] is described as an alternative for restarted GMRES. IDR(s) is short-recurrent and can be a valuable addition to the methods currently available in DIANA.

3.1.4 The Conjugate Gradient Method

The Conjugate Gradient (CG) method is a popular choice for symmetric positive definite (SPD) matrices [17]. SPD matrices yield some favorable properties, such as short-recurrence, optimality and orthogonal residuals based on $\mathcal{K}^m$. Looking at Arnoldi's procedure, note that by symmetry of K it follows from Equation (3.5) that

$$H_m = V_m^T K V_m = V_m^T K^T V_m = H_m^T,$$

which implies that the Hessenberg matrix is a symmetric tridiagonal matrix T_m, so

$$H_m := T_m = \begin{pmatrix} t_{11} & t_{12} & & & \\ t_{12} & t_{22} & t_{23} & & \\ & t_{23} & \ddots & \ddots & \\ & & \ddots & \ddots & t_{m-1,m} \\ & & & t_{m-1,m} & t_{mm} \end{pmatrix}.$$

Referring to Algorithm 1, this property results in a short-recurrence algorithm, since each additional column of T_m only consist of two unique non-zero entries. Therefore, in Algorithm 1 step 3 only t_{jj} has to be calculated. In step 4 only $t_{j-1,j}$ and t_{jj} can be unequal to zero. Adapting to common notation, let us introduce $\beta_j := ||w_{j-1}||_2$ and $h_{jj} := \alpha$. This yields the Lanczos procedure and can be viewed as a special (symmetric) case of Arnoldi's procedure [34].

Algorithm 2. ***Lanczos***

1 *Choose a vector* v_1, *such that* $||v_1||_2 = 1$
2 *For* $j = 1, 2, \ldots, m$ *Do:*
3 $w_j = Kv_j - \beta_j v_{j-1}$
4 $\alpha_j = \langle w_j, v_j \rangle$
5 $w_j = w_j - \alpha_j v_j$
6 $\beta_{j+1} = \|w_j\|_2$. *If* $\beta_{j+1} = 0$ *then Stop*
7 $v_{j+1} = w_j / \beta_{j+1}$
8 *EndDo*

Note that due to symmetry the β_j is reused in the update of w_j. Also note that by short-recurrences this version of Arnoldi is using the Modified Gram-Schmidt procedure, since v_{j+1} is orthogonalized with respect to all previous relevant predecessors.

Similar as with the non-symmetric case there are two popular strategies, namely orthogonalizing residuals ($\mathcal{L}^m = \mathcal{K}^m$) or minimizing the residual ($\mathcal{L} = K \cdot \mathcal{K}^m$) [34]. The strategy for CG is that the residuals r_m are orthogonalized with respect to each other. The SPD matrix K yields an easy-to-invert matrix T_m, which can be decomposed by a direct LU decomposition of $T_m = L_m U_m$, where L_m is a lower triangular matrix and U_m an upper triangular matrix. The bandwidth of T_m is only two, resulting in

$$T_m = L_m U_m = \begin{pmatrix} 1 & & & \\ \lambda_2 & 1 & & \\ & \ddots & \ddots & \\ & & \lambda_m & 1 \end{pmatrix} \cdot \begin{pmatrix} \eta_1 & \beta_2 & & \\ & \eta_2 & \ddots & \\ & & \ddots & \beta_m \\ & & & \eta_m \end{pmatrix}.$$

Consider Equation (3.5), which can be reduced in the symmetric case to

$$\begin{aligned} V_m^T K V_m &= T_m = L_m U_m, \\ V_m^T K V_m U_m^{-1} &= L_m, \\ U_m^{-T} V_m^T K V_m U_m^{-1} &= U_m^{-T} L_m. \end{aligned} \tag{3.10}$$

Define $P_m = V_m U_m^{-1}$, then Equation (3.10) reduces to

$$P_m^T K P_m = U_m^{-T} L_m. \tag{3.11}$$

The right-hand-side of Equation (3.11) results in a symmetric and lower triangular matrix and therefore a diagonal matrix. The columns p_j of P_m are called the search direction vectors or conjugate vectors. From the resulting diagonal matrix in Equation (3.11) follows that the conjugate vectors p_j are K-orthogonal, i.e., $\langle p_i, K p_j \rangle = 0$, $\forall i \neq j$.

The consequence of symmetry is that the residuals r_j and K-conjugate p_j can be constructed in a recurrence of only two vectors, while the non-symmetric FOM requires an additional

vector for every iteration. Furthermore, the approximating solution vector u_m can be updated every iteration. The resulting Conjugate Gradient algorithm applicable for SPD matrices K is shown in Algorithm 3, adapted to common notation.

Algorithm 3. ***Conjugate Gradient***

Compute $r_0 = f - Ku_0$, $p_0 = r_0$.
For $j = 0, 1, \ldots,$ *until convergence, Do:*
  $\alpha_j = \langle r_j, r_j \rangle / \langle p_j, Kp_j \rangle$
  $u_{j+1} = u_j + \alpha_j p_j$
  $r_{j+1} = r_j - \alpha_j K p_j$
  $\beta_j = \langle r_{j+1}, r_{j+1} \rangle / \langle r_j, r_j \rangle$
  $p_{j+1} = r_{j+1} + \beta_j p_j$
EndDo

For a full derivation of the CG method please refer to Saad [34]. In [34] is also shown that the coefficients in Algorithm 3 can be used to directly compute T_m as

$$T_m = \begin{pmatrix} \frac{1}{\alpha_0} & \frac{\sqrt{\beta_0}}{\alpha_0} & & & \\ \frac{\sqrt{\beta_0}}{\alpha_0} & \frac{1}{\alpha_1} + \frac{\beta_0}{\alpha_0} & \frac{\sqrt{\beta_1}}{\alpha_1} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} \\ & & & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} & \frac{1}{\alpha_{m-1}} + \frac{\beta_{m-2}}{\alpha_{m-2}} \end{pmatrix}. \tag{3.12}$$

Corresponding eigenvalues of T_m in Equation (3.12) are called Ritz values and they approximate the extreme eigenvalues of the matrix K. The QR algorithm [11] could be used to determine the Ritz vectors, where Q is an orthogonal matrix and R an upper triangular matrix. This information can be useful in a nonlinear iteration loop using eigenvector deflation.

The CG algorithm is the most popular choice for SPD matrices, combining optimality and short-recurrence. To be precise, CG minimizes $||u - u_m||_K$ using orthogonal residuals $r_m = f - Ku_m$. The convergence behavior of the CG method is determined by the condition number (for SPD matrices) $\kappa = \lambda_{\max}/\lambda_{\min}$ as by Definition 4 in Section ii. The following bound for the CG method is well-known:

Theorem 2. *Let K be a symmetric positive definite matrix. Then the error $u - u_m$ of the CG method at iteration m is bounded by*

$$\|u - u_m\|_K \leq 2 \left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right]^m \|u - u_0\|_K.$$

This implies that a small condition number $\kappa \geq 1$ results in fast convergence. The proof can be found in e.g. Saad [34].

3.2 Preconditioning

For any Krylov subspace method it is important to have a preconditioner to improve its performance. Preconditioning means that the system is multiplied with a well-chosen matrix called the preconditioner. The newly acquired system should have better convergence properties than the original one. An effective preconditioner P should resemble K in some way and it should be easy to solve $Px = y$. Preconditioners are often inspired by the

matrix P where $K = P - N$ as in Equation (3.1), or by direct methods. Preconditioning can be applied in different ways. Left-preconditioning is shown in Equation (3.13), central-preconditioning in Equation (3.14) and right-preconditioning in Equation (3.15).

$$P^{-1}Ku = P^{-1}f, \tag{3.13}$$

$$P = LU; \qquad L^{-1}KU^{-1}x = L^{-1}f; \qquad u = U^{-1}x, \tag{3.14}$$

$$KP^{-1}x = f; \qquad u = P^{-1}x. \tag{3.15}$$

3.2.1 Preconditioned CG

Preconditioning CG directly effects the convergence by Theorem 2. Central-preconditioning preserves symmetry by $P = LL^T$. This is advantageous for SPD matrices K, since CG can directly be applied on the symmetric system $L^{-1}KL^{-T}$.

Left- and right-preconditioning can destroy the symmetry of the system, even when P^{-1} is symmetric. Yet, there is a solution to circumvent this by using other inner products than the standard Euclidean inner product in CG iterations. Note that the left-preconditioned system $P^{-1}K$ is self-adjoint if the P-inner product is used:

$$\langle P^{-1}Kx, y\rangle_P = \langle Kx, y\rangle_2 = \langle x, Ky\rangle_2 = \langle x, P(P^{-1}K)y\rangle_2 = \langle x, P^{-1}Ky\rangle_P.$$

This implies that left-preconditioning combined with the P-inner product preserves symmetry. Note that the right-preconditioned system KP^{-1} is self-adjoint if the P^{-1}-inner product is used

$$\langle KP^{-1}x, y\rangle_{P^{-1}} = \langle P^{-1}KP^{-1}x, y\rangle_2 = \langle x, P^{-1}KP^{-1}y\rangle_2 = \langle x, KP^{-1}y\rangle_{P^{-1}}.$$

This implies that right-preconditioning combined with the P^{-1}-inner product preserves symmetry. Moreover, rewriting the CG algorithm for left-preconditioning with the P-inner product results in the same algorithm as rewriting the CG algorithm for right-preconditioning with the P^{-1}-inner product. In other words, the left-preconditioned CG algorithm with the P-inner product is mathematically equivalent to the right-preconditioned CG algorithm with the P^{-1}-inner product [34]. The central-preconditioning can also be rewritten to the same algorithm.

3.2.2 Preconditioned GMRES

GMRES does not require a symmetric system. Therefore, preconditioning GMRES can be done straightforwardly. Left-preconditioning results in computing the initial residual at the start of GMRES as

$$r_0 = P^{-1}(f - Ku_0).$$

Right-preconditioning yields computing the solution at the end of GMRES as

$$x_m = x_0 + P^{-1}V_m y_m.$$

Central-preconditioning $P = LU$ is a combination of both by $r_0 = L^{-1}(f - Ku_0)$ and $u_m = u_0 + U^{-1}V_m y_m$.

When comparing left-, right- and central-preconditioning for GMRES, observe that the spectra of the three associated operators $P^{-1}K$, KP^{-1} and $L^{-1}KU^{-1}$ are identical. In practice however, the convergence behavior differs. Left-preconditioning minimizes the residual norm $||P^{-1}(f-Ku_m)||_2$, but preserves the original iterations u_m. Right-preconditioning preserves the original residual norm, but requires to calculate $u_m = P^{-1}x_m$ after convergence. Although all norms on a finite space are equivalent, it still means that ill-conditioned systems yield different convergence behavior due to numerical issues. DIANA applies right-preconditioning for restarted GMRES.

3.2.3 Diagonal scaling

The simplest preconditioner P for $K \in \mathbb{R}^{n\times n}$ is diagonal scaling or Jacobi preconditioning. The Jacobi preconditioner P has non-zero entries

$$P_{ii} = K_{ii}, \quad i = 1, \ldots, n.$$

The advantage of this preconditioner is that is cheap to construct, it is cheap to solve $Px = y$ and easy to parallelize. The disadvantage is that it does not resemble K very accurately in general, resulting in only slightly less iterations.

3.2.4 Incomplete decompositions

The full Cholesky decomposition is applicable for SPD matrices K, which decomposes the matrix $K = LL^T$, where L is a lower triangular matrix. The non-symmetric variant is a full LU decomposition, which decomposes $K = LU$, where L is a lower triangular matrix and U an upper triangular matrix. Full decompositions can be used as a direct solution method, where a forward and backward substitution are performed to solve $Ku = f$. Performing a full decomposition on a large sparse matrix can be too expensive to be applied directly. The so-called bandwidth of the matrix K results in fill-in and a full decomposition is not always competitive.

The Incomplete Cholesky (IC) decomposition for symmetric systems and the Incomplete LU (ILU) decomposition for non-symmetric systems are based on the idea to do the decomposition of the matrix incompletely. These incomplete decompositions can be used as preconditioners for an iterative solution method. The preconditioners $P = LU$ for non-symmetric systems and $P = LL^T$ for symmetric systems should approximate the stiffness matrix K to some extent. Note that preconditioning by diagonal scaling is actually a special case of an incomplete decomposition, restricting the fill-in of P to the diagonal of K.

The standard preconditioner in DIANA for non-symmetric systems is the ILU decomposition without fill-in ILU(0).

$$\begin{aligned} (L+U)_{ij} &= 0 \qquad \text{if } A_{ij} = 0, \\ (LU)_{ij} &= A_{ij} \qquad \text{otherwise.} \end{aligned} \tag{3.16}$$

Replacing in Equation (3.16) matrix U by L^T gives the IC decomposition without fill-in, IC(0). As a result, the sparsity pattern of K is unchanged, which saves memory and CPU time.

The ILU and IC decompositions can also be applied in a block-wise approach, e.g. applicable in a parallel environment. In that case the following holds:

$$\begin{aligned} (L+U)_{[i,j]} &= 0 && \text{if } A_{[i,j]} = 0, \\ (LU)_{[i,j]} &= A_{[i,j]} && \text{otherwise.} \end{aligned}$$

This block-wise approach is used in DIANA for the parallel domain decomposition.

If the iterative solution method fails to converge using this preconditioner, then a threshold τ for fill-in is set up. These preconditioners are abbreviated by ICT(τ) and ILUT(τ). The threshold τ for fill-in is decreased until the iterative solver converges within a specified number of iterations. Decreasing τ results in a more accurate and more expensive approximation of K. Note that the exact factorization is obtained if the drop tolerance is small enough, e.g. $\tau = 0$. Details of the IC and ILU decomposition can be found in e.g. Saad [34] or Van der Vorst [44].

3.2.5 Other preconditioners

DIANA also offers substructuring in the form of a preconditioner (although technically, it is a Schur domain decomposition). In the context of Schwarz domain decomposition an additive Schwarz preconditioner and a coarse grid correction are available. These domain decomposition techniques are explained in Section 4. The coarse grid correction is a specific case of algebraic multigrid.

3.3 Multigrid

Multigrid methods are efficient iterative methods for the solution of linear systems [34]. Two types of multigrid can be distinguished, namely geometric and algebraic multigrid. The advantage of geometric multigrid is its efficiency, however, it can only be applied when the geometric grid and underlying PDEs are explicitly known. Algebraic multigrid is more adaptive and only requires information from the matrix and underlying connectivity itself, although the costs per iteration are slightly higher than with geometric multigrid. For generic problems, algebraic multigrid is more robust than geometric multigrid.

Multigrid can be implemented as a preconditioner, but this is only implicit. Multigrid actually *corrects* the residual vector in the following way:

$$u_{i+1} = u_i + P^{-1}(f - Ku_i).$$

This computation is not directly performed. Multigrid uses two complementary processes, namely relaxation and coarse grid correction. In the relaxation phase an iteration is performed to damp the low frequencies in the error. Thereafter, the coarse grid correction damps the high frequencies by projecting the grid on a restrictive coarse grid. The following procedure is followed

1. Restrict the residual
$$\tilde{r}_i = Y^T r_i.$$

2. Compute the coarse solution
$$Y^T K Z \Delta \tilde{u}_i = \tilde{r}_i.$$

3. Expand the coarse solution
$$\Delta u_i = Z \Delta \tilde{u}_i.$$

4. Correct the solution
$$u_{i+1} = u_i + \Delta u_i.$$

where Y^T the restriction operator and Z the interpolation operator. In algebraic multigrid it often holds that $Y = Z$. The coarse problem of the algebraic multigrid is formulated with the coarse matrix (also: Galerkin matrix)

$$E = Z^T K Z.$$

This technique can be implemented as a stand-alone solution algorithm, where the coarsening process is repeatedly applied, see e.g. Notay [33]. Furthermore, algebraic multigrid can be used in a Krylov method. In DIANA an analogue technique called coarse grid correction is implemented in the Schwarz domain decomposition method.

4 Domain decompositions

Domain decomposition methods aim at the divide-and-conquer strategy. A domain can be described by a collection of subdomains, which can be solved separately with correct coupling. Domain decomposition methods attempt to solve the problem on the entire domain

$$\Omega = \bigcup_{i=1}^{n_d} \Omega_i,$$

by constructing solutions on the subdomains Ω_i [34].

Domain decomposition methods have several advantages. Historically, one of the most important features was memory management. The idea is to partition the original structure into n_d subdomains, where each subdomain fits in memory. Thereafter, the subdomain solutions can be used to compute the global solution. A more recent trend in domain decomposition methods is parallel computing.

Two types of domain decomposition methods exist: the Schur and Schwarz domain decomposition method. DIANA supports a sequential Schur domain decomposition method, also called substructuring, and a parallel Schwarz domain decomposition method. Any domain decomposition method requires partitioning.

4.1 Partitioning

A simple partitioning is illustrated in Figure 2. An efficient partitioner should have three objectives: minimize the number of so-called interface degrees of freedom, minimize the variation in subdomain sizes and group together the degrees of freedom that have similar properties [28, 34].

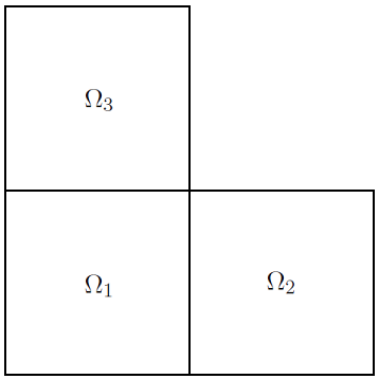


Figure 2: The domain Ω divided in three subdomains without overlapping elements.

1. Minimize the number interface degrees of freedom. The so-called interface degrees of freedom can be loosely described as degrees of freedom that occur in multiple domains. This first objective minimizes communication between subdomains. In the context of parallel processes this improves parallelism and in the context of sequential computing this speeds up the computation of the solution in the interfaces.

2. Minimize the variation in subdomain sizes. The second objective ensures balanced computing time for the subdomains. This is mainly important for optimal parallel computations.

3. Group together degrees of freedom with similar properties. The third objective is based on the observation that preconditioners tend to be more effective if the subdomain is fairly regular in terms of geometry, linearity, underlying PDEs or material properties [28, 34].

A variety of domain decompositions have been developed and applied. It is hard to satisfy the three above objectives all together and a preference has to be made. DIANA supports two types of domain decompositions in the form of substructuring (Section 4.2) and parallel domain decomposition (Section 4.3). The partitioning for substructuring focuses on objectives 1 and 3. The partitioning for parallel domain decomposition focuses on objectives 1 and 2. The parallel domain decomposition partitioning uses Metis [26], a graph partitioning open software package. Metis partitions the elements of the model, so that no element is split up. This implementation does make sure that the partitioning is balanced, based on the underlying connectivity of the elements and it also minimizes the interface degrees of freedom. However, no other information, such as material properties, element types or stiffness is used to determine the partitioning.

In principal, a partition does not need to contain any overlapping elements; all elements can be assigned to exactly one subdomain. In substructuring no overlapping elements occur. In the parallel domain decomposition some elements are assigned to multiple subdomains to improve the convergence of an iterative solution method.
Degrees of freedom are originally mapped to one subdomain; these are called internal degrees of freedom. The additional degrees of freedom are called interface degrees of freedom. Each degree of freedom is an internal degree of freedom in exactly one subdomain and an interface degree of freedom in a subdomain is also an internal degree of freedom in exactly

one other subdomain.

4.2 Substructuring

Substructuring or Schur domain decomposition is one of the two types of domain decomposition methods. The idea is to divide the whole domain into n_d substructures (subdomains) of similar properties. The most important example in DIANA is to put elements that behave linearly in a nonlinear model in a substructure. The coefficients of such a substructure will not change throughout the nonlinear iterations and the decomposition can be reused by a direct solver.

The partitioning in substructures is based on element properties. Thereafter, the degrees of freedom are divided in internal degrees of freedom and interface degrees of freedom. Internal degrees of freedom belong uniquely to a substructure, whereas interface degrees of freedom belong to multiple substructures. The degrees of freedom corresponding to the n_d substructures can be reordered such that the stiffness matrix K can be written as

$$K \sim \begin{pmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_{n_d} & B_{n_d} \\ B_1^T & B_2^T & \dots & B_{n_d}^T & C \end{pmatrix}.$$

The A_i correspond to internal degrees of freedom of substructure i and the coupling between the substructures is described by the interface degrees of freedom in B_i and B_i^T. The reordered matrix K can easily be factorized as

$$\begin{pmatrix} A_1 & & & & \\ & A_2 & & & \\ & & \ddots & & \\ & & & A_{n_d} & \\ B_1^T & B_2^T & \dots & B_{n_d}^T & C^* \end{pmatrix} \begin{pmatrix} I & & & & A_1^{-1}B_1 \\ & I & & & A_2^{-1}B_2 \\ & & \ddots & & \vdots \\ & & & I & A_{n_d}^{-1}B_{n_d} \\ & & & & I \end{pmatrix}.$$

The local solution of the internal degrees of freedom in each substructure can be computed independently. Assuming all A_i are SPD, the local solution is computed by $B_i^T A_i^{-1} B_i$ using a Cholesky factorization. The computation of the so-called Schur complement is given by

$$C^* = C - \sum_{i=1}^{n_d} B_i^T A_i^{-1} B_i,$$

which can be the most time consuming part. The system of equation $C^* u = f$ is in general a dense system and it can be solved by a solver of the user's choice. Substructuring can be effective, but has some disadvantages. If the ratio interface degrees of freedom to internal degrees of freedom is high, then substructuring can be ineffective due to the density of C^*.

Substructuring in DIANA is implemented as a preconditioning technique without overlapping subdomains.

4.3 Schwarz domain decomposition

The purpose of Schwarz domain decomposition is to divide the domain into a number of subdomains for parallel processing. The parallel Schwarz domain decomposition at DIANA

is designed by Erik Jan Lingen and the implementation has strong similarities with his later work [28]. The DIANA user can specify the number of available threads when using the iterative solver. This number is equal to the number of subdomains the parallel iterative solver will use.

Let us define the boolean left and right restriction operators of the i-th subdomain [28]. The left restriction operator L_i correspond to the internal degrees of freedom of the i-th subdomain, while the right restriction operator R_i corresponds to internal and interface degrees of freedom of the i-th subdomain. Note that non-zero columns of $(R_i - L_i)$ indicate the interface degrees of freedom of the i-th subdomain.

The matrix K can be expressed in terms of the subdomain matrices K_i (which may be overlapping) by

$$K = \sum_{i=1}^{n_d} L_i^T K_i R_i.$$

The domain decomposition uses a two-level Schwarz preconditioner. The first preconditioner is an additive Schwarz (AS) preconditioner and the second is a coarse grid correction. The AS preconditioner is used to combine the local preconditioners of each subdomain. The coarse grid correction aims to provide global communication at each iteration in order to make the convergence rate independent of the problem size and the number of subdomains.

The AS preconditioner P^{-1} preserves symmetry (in case of symmetric subdomain preconditioners P_i^{-1}) by ignoring overlap and is constructed as follows:

$$P^{-1} = \sum_{i=1}^{n_d} R_i^T P_i^{-1} R_i,$$

where P_i^{-1} are the subdomain preconditioners, such as an ILU decomposition preconditioner. If the additive Schwarz preconditioner fails, the more effective restricted additive Schwarz (RAS) preconditioner is being used, given by

$$P^{-1} = \sum_{i=1}^{n_d} L_i^T P_i^{-1} R_i.$$

The RAS preconditioner, however, is non-symmetric and forces the use of GMRES(s) or another non-symmetric iterative solver.

The second preconditioner is a coarse grid preconditioner. It is constructed in a similar way to classical algebraic multigrid, described in Section 3.3, only the coarsening is extreme [37]. The coarse grid correction without other preconditioner is obtained in the following way:

$$P_C = I + Z(Z^T K Z)^{-1} Z^T = I + Z E^{-1} Z^T, \tag{4.1}$$

where Z is given by the rigid body modes of the n_d subdomains.

In multigrid terminology Z^T is the restriction operator and Z the interpolation operator. The restriction and interpolation are based on the approximate null space of the domains as described in Section 6. Each domain is described by (restricted to) six vectors, namely three translation and three rotation vectors per domain. The coarse system $Ex = y$ can then be

solved by e.g. a direct solution method. The coarse solution x is thereafter interpolated by applying Z. The extended solution vector can be used to correct the iterative solution vector.
The computation of E^{-1} is computationally most expensive and is done in DIANA by a QR-decomposition

$$E = QR,$$

with Q an orthonormal matrix and R an upper triangular matrix. This process can be performed in parallel (as implemented in DIANA) by applying a Gram-Schmidt orthonormalization procedure to the columns of E. For details we refer to Lingen [28].

Coarse grid correction can be applied as a stand-alone preconditioner as in Equation (4.1) or in combination with another preconditioner. Combining preconditioners can be done in an additive or a multiplicative way. The additive way is simply the addition:

$$P_{C,P^{-1}} = P^{-1} + ZE^{-1}Z^T. \tag{4.2}$$

A more effective strategy is the multiplicative way [28]. The techniques are computed as follows:

$$\begin{array}{ll} \text{Additive} & \text{Multiplicative} \\ y_1 = P^{-1}x, & \tilde{y} = P^{-1}x, \\ y_2 = ZE^{-1}Z^Tx, & \tilde{r} = x - K\tilde{y}, \\ y = y_1 + y_2. & y = ZE^{-1}Z^T\tilde{r}. \end{array} \tag{4.3}$$

Although more effective, the multiplicative technique requires an extra matrix-vector multiplication including additional communication between threads.

4.4 Substructuring versus parallel domain decomposition

The two domain decomposition methods in DIANA seem similar but are very different. This section provides a short overview of (the implementation of) substructuring and parallel domain decomposition in Table 1.

Substructuring	**Parallel domain decomposition**
Sequential computations	Parallel computations
Partitioning: minimum number of interface degrees of freedom and similar type	Partitioning: minimum number of interface degrees of freedom and balanced work
Home-made partitioning in `SOLVE` building block	Metis partitioning in `DOMDEC` building block
Non-overlapping	Overlapping as well as non-overlapping
Cholesky decomposition per substructure Solve Schur complement by method of user's choice	Additive Schwarz preconditioner Coarse grid correction
Implemented as preconditioner	Implemented as solution method

Table 1: Differences between DIANA's Schur (left) and Schwarz (right) domain decomposition methods.

5 Deflation

Deflation is a technique to improve an iterative solution method. The main difference between preconditioning and deflation is that deflation is a projection and hence not invertible. Deflation is particularly suited to eliminate eigenvectors corresponding to small eigenvalues of a matrix. These eigenvalues are projected out of the system of equations. Deflation has been developed by Nicolaides [32] and Dostál [8] and different deflation techniques have been developed, improved and exploited by other authors [9, 12, 25].

Deflation has some analogies with algebraic multigrid methods, as deflation also uses the restriction operator Y^T and interpolation operator Z. Deflation is based on two projections $\Pi^{\in}$ and $\Pi^{\perp}$, which are constructed by Y^T and Z [12].

The solution u is split into two parts: one part to be solved directly and one part to be solved iteratively. The splitting of solution u can be written as follows:

$$u = u^{\in} + u^{\perp}. \tag{5.1}$$

Let the interpolation operator $Z \in \mathbb{R}^{n \times m}$ be a basis for the $\mathcal{Z}$ and the restriction operator $Y^T \in \mathbb{R}^{n \times m}$ be a basis for $\mathcal{Y}$ with $m \ll n$. The part of the solution u in $\mathcal{Z}$, $u^{\perp}$, can be written as a linear combination of Z, implying $u^{\perp} = Zy$. The residual $r^{\perp} = f - Ku^{\perp}$ is orthogonalized with respect to Y, i.e., $r^{\perp} \perp Y$. This requirement [34] can be written as

$$\begin{aligned} Y^T r^{\perp} &= 0, \\ Y^T (f - Ku^{\perp}) &= 0, \\ Y^T (f - KZy) &= 0. \end{aligned} \tag{5.2}$$

Define the coarse matrix or Galerkin operator $E = Y^T KZ$. Equation (5.2) requires $y = (Y^T KZ)^{-1} Y^T f = E^{-1} Y^T f$. Substituting this into $u^{\perp}$ results in

$$\begin{aligned} u^{\perp} &= Zy = ZE^{-1}Y^T f, \\ &= ZE^{-1}Y^T K u. \end{aligned} \tag{5.3}$$

Defining the projector $\Pi^{\in} = I - ZE^{-1}Y^T K$ yields

$$u^{\perp} = (I - \Pi^{\in})u.$$

Note that for projector $\Pi^{\in}$ indeed holds $(\Pi^{\in})^2 = \Pi^{\in}$. Furthermore, the solution $u^{\perp}$ can be calculated directly as in the first statement of Equation (5.3). In general the matrix Z consists of m columns with $m \ll n$, implying that this part is relatively easy to solve.

Equation (5.1) can also be written as

$$u = (I - \Pi^{\in})u + \Pi^{\in}u. \tag{5.4}$$

The projector $\Pi^{\perp}$ can be constructed by finding a solution for $u^{\in} := \Pi^{\in}u$. For this purpose $u^{\in}$ is premultiplied by K, resulting in

$$\begin{aligned} Ku^{\in} &= K\Pi^{\in}u, \\ K\Pi^{\in}u &= K(I - ZE^{-1}Y^T K)u, \\ &= f - KZE^{-1}Y^T Ku, \\ &= (I - KZE^{-1}Y^T)f. \end{aligned}$$

Defining the projector $\Pi^{\perp} = I - KZE^{-1}Y^T$ yields

$$K\Pi^{\in}u = \Pi^{\perp}f. \tag{5.5}$$

Note that indeed holds $(\Pi^{\perp})^2 = \Pi^{\perp}$. Using the identity $\Pi^{\perp}K = K\Pi^{\in}$ in Equation (5.5) implies

$$\Pi^{\perp}K\tilde{u} = \Pi^{\perp}f. \tag{5.6}$$

The solution $\tilde{u}$ of Equation (5.6) is introduced to avoid ambiguity. The solution $\tilde{u}$ is the computational difficult part and can be solved iteratively. Deflation can in the light of Equation (5.6) be seen as a left preconditioner (although it is a projection). This projection results in a singular system. The singularity of this system is not necessarily a problem, as long as the corresponding right-hand side $\Pi^{\perp}f$ is in the range of $\Pi^{\perp}K$ (Equation (5.6) should be consistent) [3, 24]. The projection $\Pi^{\perp}$ is applied at the right-hand side as well, thus it still holds that $f = K\tilde{u}$ for some $\tilde{u}$.

The solution $u = u^{\in} + u^{\perp}$ can be computed by combining the solutions $u^{\perp}$ and $\tilde{u}$ of Equation (5.3) respectively Equation (5.6) into Equation (5.4) as

$$u = ZE^{-1}Y^T f + \Pi^{\in}\tilde{u}. \tag{5.7}$$

Note that the extreme case where the complete space is deflated, $Z = Y = I$, results in a direct solution method and reduces Equation (5.7) to $u = K^{-1}f$.

In case that matrix K is symmetric, it is advantageous to preserve symmetry in the application of deflation. Note that to preserve symmetry, only $\Pi^{\perp}K = K\Pi^{\in}$ is not sufficient,

but the requirement $\Pi^{\perp}K = K(\Pi^{\perp})^T$ should hold. This implies $\Pi^{\in} = (\Pi^{\perp})^T$ and thus $Y = Z$.

The non-symmetric case allows more freedom for the choice of Y and Z. Often the choice $Y = Z$ is made anyway. This choice allows us to draw some conclusions about the robustness of deflation. An SPD matrix K and full rank matrices $Y = Z$ ensure a nonsingular coarse matrix E. The robustness and (non)singularity concerning E is discussed in Section 5.2. Another advantage is that only one set of vectors need to be determined and stored. From this point it is assumed that $Y = Z$.

Deflation is very suitable in combination with a preconditioner. Typically, deflation could deal with the smallest eigenvalues, while a preconditioner deals with the largest eigenvalues. The choice of Z strongly influences the effectiveness of deflation. A good choice for Z is an important but not so obvious part of deflation. Typical options for Z are elaborated in Sections 5.3, 5.4 and 5.5.

A variant of the Deflated Preconditioned Conjugate Gradient method [39] is given in Algorithm 4:

Algorithm 4. ***Deflated Preconditioned Conjugate Gradient***

Choose u_0. *Compute* $r_0 = f - Ku_0$. *Set* $r_0 := \Pi^{\perp} r_0$.
Solve $Pz_0 = r_0$ *and set* $p_0 = z_0$.
For $j = 0, 1, \ldots$ *until convergence, Do*
  $z_j = \Pi^{\perp} K p_j$
  $\alpha_j = \langle r_j, z_j \rangle / \langle p_j, z_j \rangle$
  $u_{j+1} = u_j + \alpha_j p_j$
  $r_{j+1} = r_j - \alpha_j z_j$
  Solve $Pz_{j+1} = r_{j+1}$
  $\beta_j = \langle r_{j+1}, z_{j+1} \rangle / \langle r_j, z_j \rangle$
  $p_{j+1} = z_{j+1} + \beta_j p_j$
EndDo
$u = ZE^{-1}Z^T f + \Pi^{\in} u_{j+1}$

The algorithm for deflation applied to GMRES(s) is given as Algorithm 5:

Algorithm 5. ***Deflated right-preconditioned GMRES(s)***

Choose u_0. *Compute* $r_0 = f - Ku_0$.
Set $r_0 := \Pi^{\perp} r_0$, $\beta = \|r_0\|_2$ *and* $v_1 = r_0/\beta$.
For $j = 0, 1, \ldots, s$, *Do*
  Compute $w = \Pi^{\perp} K P^{-1} v_j$
  For $i = 1, \ldots, j$, *Do*
    $h_{i,j} = \langle w, v_i \rangle$
    $w := w - h_{i,j} v_i$
  EndDo
  $h_{j+1,j} = \|w\|_2$ *and* $v_{j+1} = w / h_{j+1,j}$
  $V_s = [v_1, \ldots, v_s]$ *and* $\bar{H}_s = \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le s}$
EndDo
Compute $y_s = \text{argmin}_y \|\beta e_1 - \bar{H}_s y\|_2$ *and* $u_s = u_0 + P^{-1} V_s y_s$
$u = ZE^{-1}Z^T f + \Pi^{\in} u_s$

5.1 Convergence of deflation

The convergence of an iterative method is influenced by the condition number of the system of equations. The condition number is defined as $\kappa(K) = ||K^{-1}||_2 \cdot ||K||_2$, which yields for symmetric positive definite matrices $\kappa = \lambda_{\max}/\lambda_{\min}$.

Deflation involves multiplying matrix K from the left by the projection matrix $\Pi^{\perp}$. By (effective) deflation some small eigenvalues are projected out of the system of equations (projected to zero). This decreases κ and often increases the rate of convergence. The following two theoretical bounds on the effective condition number (Definition 4) of $\Pi^{\perp}K$ for SPD matrices K are given [12].

Theorem 3. *Let K be symmetric positive definite, let $\Pi^{\perp} = I - KZ(Z^TKZ)^{-1}Z^T$, $Z \in \mathbb{R}^{n\times m}$, and suppose we can split $K = C + R$ such that C and R are symmetric positive semidefinite with $\mathcal{N}(C) = span\{Z\}$ the null space of C. Then*

$$\lambda_i(C) \leq \lambda_i(\Pi^{\perp}K) \leq \lambda_i(C) + \lambda_{\max}(\Pi^{\perp}R). \tag{5.8}$$

The effective condition number of $\Pi^{\perp}K$ is bounded by

$$\kappa_{\mathit{eff}}(\Pi^{\perp}K) \leq \frac{\lambda_{\max}(K)}{\lambda_{m+1}(C)}. \tag{5.9}$$

Proof. Note that $\Pi^{\perp}K$ is symmetric. Since Z is in the null space of C, it follows that $\Pi^{\perp}C = C$ is also symmetric. Matrix R is positive semidefinite, $\Pi^{\perp}$ a projection and $\Pi^{\perp}R = \Pi^{\perp}K - C$ is symmetric. Define $y = (\Pi^{\perp})^T x$, then:

$$\begin{aligned} x^T\Pi^{\perp}Rx &= x^T(\Pi^{\perp})^2Rx \\ &= x^T\Pi^{\perp}R(\Pi^{\perp})^Tx \\ &= ((\Pi^{\perp})^Tx)^TR((\Pi^{\perp})^Tx) \\ &= y^TRy \geq 0, \end{aligned}$$

so $\Pi^{\perp}R$ is symmetric positive semidefinite. This implies $\lambda_{\min}(\Pi^{\perp}R) \geq 0$. The bounds in Inequality (5.8) follow from Theorem 8.1.5. of Golub et al. [13].

$$\lambda_i(\Pi^{\perp}C) + \lambda_{\min}(\Pi^{\perp}R) \leq \lambda_i(\Pi^{\perp}K) \leq \lambda_i(\Pi^{\perp}C) + \lambda_{\max}(\Pi^{\perp}R).$$

This results in $\lambda_i(\Pi^{\perp}K) \geq \lambda_i(C)$. Furthermore, since $\Pi^{\perp}K$ is also positive semi-definite, Theorem 8.1.5 of [13] gives $\lambda_{\max}(\Pi^{\perp}K) \leq \lambda_{\max}(K)$. Combining these bounds gives Inequality (5.9). □

The preconditioned variant of Theorem 3 with a split Cholesky preconditioner $P = LL^T$ is the following theorem [12]:

Theorem 4. *Assume the conditions of Theorem 3 and let $P = LL^T$ be a symmetric positive definite Cholesky based preconditioner. Then the effective condition number of $L^{-1}\Pi^{\perp}KL^{-T}$ is bounded by*

$$\kappa_{\mathit{eff}}(L^{-1}\Pi^{\perp}KL^{-T}) \leq \frac{\lambda_{\max}(L^{-1}KL^{-T})}{\lambda_{m+1}(L^{-1}CL^{-T})}.$$

Proof. Define $\hat{K} = L^{-1}KL^{-T}$, $\hat{C} = L^{-1}CL^{-T}$, $\hat{R} = L^{-1}RL^{-T}$, $\hat{Z} = L^{T}Z$ and

$$\hat{\Pi^{\perp}} = I - \hat{K}\hat{Z}(\hat{Z}^{T}\hat{K}\hat{Z})^{-1}\hat{Z}^{T} = L^{-1}\Pi^{\perp}L.$$

Note that $\hat{\Pi^{\perp}} = \hat{\Pi^{\perp}}^{2}$ is a projection, $\hat{\Pi^{\perp}}\hat{K}$ is symmetric and it still holds that $\hat{\Pi^{\perp}}\hat{C} = \hat{C}$. Now, apply Theorem 3 on the deflated matrix $\hat{\Pi^{\perp}}\hat{K}$ and substitute back all original variables.

□

Enlarging the size of the full rank matrix $Z \in \mathbb{R}^{n\times m}$, thus increasing m, is equivalent to enlarging the null space of C. This means that the effective condition number of the deflated system will in general improve for increasing dimensions of full rank matrix Z. For symmetric matrices this means that the convergence improves. For non-symmetric matrices this is not necessarily true, although if the non-symmetric part is not too dominant, similar convergence behavior may be expected.

5.2 Robustness

The coarse matrix $E = Z^{T}KZ$ must be nonsingular. For SPD matrices K it is sufficient to have full rank Z by the following theorem [12].

Theorem 5. *Assume $K \in \mathbb{R}^{n\times n}$ to be a symmetric positive definite matrix and let $Z \in \mathbb{R}^{n\times m}$ be a full rank matrix. Then $E = Z^{T}KZ$ is nonsingular.*

Proof. It is sufficient to show that E is also SPD. Since K is SPD, by definition holds that:

$$x^{T}Kx > 0, \quad \forall x \in \mathbb{R}^{n}, x \neq 0.$$

Since Z is full rank, we can always write $x = Zy \neq 0$, $\forall y \in \mathbb{R}^{m}, y \neq 0$. Now, $\forall y \in \mathbb{R}^{m}, y \neq 0$ holds

$$y^{T}Ey = y^{T}Z^{T}KZy = (Zy)^{T}K(Zy) = x^{T}Kx > 0,$$

which shows that E is positive definite. The theorem follows by $E = Z^{T}KZ = (Z^{T}KZ)^{T} = E^{T}$. □

This theorem does not hold for non-symmetric matrices K. In Yeung et al. [51] it is proven that E is nonsingular under the assumption that the columns of Z form a K-invariant subspace. This is certainly true for eigenvectors of K, but in general needs to be checked. For practical implications concerning the robustness of deflation please refer to Section 8.3.

5.3 Eigenvector deflation

An obvious choice for Z is the span of the eigenvectors corresponding to the smallest eigenvalues. Such an approach certainly is effective, although the smallest eigenvectors are not always known beforehand. However, in some iterative methods such as GMRES(s), the (approximate) eigenvectors can be computed relatively cheap. These vectors can be used as a deflation space in the restart of GMRES(s) or in the nonlinear iterative loop [9, 29]. Another limitation is the effectiveness in case of a large number of independent small eigenvectors. Deflating a large amount of small eigenvectors means that the dimension of Z grows beyond its effectiveness.

Eigenvector deflation has shown to be effective. The Ritz vectors can be computed and used in CG and GMRES(s). In Erhel et al. [9] and Morgan [29] the restart of GMRES(s)

is augmented or deflated using approximated eigenvectors and in Gosselet et al. [14,15] the CG method is augmented in nonlinear structural analysis problems. In DIANA these applications of eigenvector deflation can be an effective technique to speed up the convergence process.

5.4 Subdomain deflation

Let us divide the domain $\Omega = \bigcup_{j=1}^{n_d} \Omega_j$ into n_d subdomains and choose

$$Z_{ij} = \begin{cases} 1 & \text{if } i \in \Omega_j, \\ 0 & \text{otherwise,} \end{cases}$$

resulting in $Z \in \mathbb{R}^{n \times n_d}$. If it is advantageous to decouple these domains (for similarity in properties or parallel computations), then the convergence of an iteration method could speed up by using subdomain deflation. DIANA uses a coarse grid correction in its Schwarz domain decomposition, which has the same purpose as subdomain deflation. Subdomain deflation can be extended to rigid body modes deflation.

5.5 Rigid body modes deflation

The idea of rigid body modes deflation is to treat a collection of elements as a rigid body. This can be advantageous due to physical properties or for parallel computations. The stiffness of a collection of elements can be relatively large and therefore it approximately acts as a rigid body. A true rigid body would result in a singular matrix and eigenvalues equal to zero. The so-called approximate rigid bodies show similar behavior, resulting in eigenvalues close to zero. Approximate rigid bodies are a collection of stiff and interconnected elements and these typically cause the matrix K to be ill-conditioned, as the discontinuities in the physical properties result in large jumps in the coefficients of K. Deflating the rigid body modes of these collections of elements would improve the condition of K.

Ignoring scalar degrees of freedom for the moment, the rigid body modes of a three-dimensional node, element or collection of elements can be described by six modes, namely three translational modes and three rotational modes. Consider a single node consisting of three translational degrees of freedom u_x, u_y and u_z. The rigid body modes of this nodes are given by

$$\begin{matrix} u_x \\ u_y \\ u_z \end{matrix} \begin{pmatrix} 1 & 0 & 0 & 0 & -z & y \\ 0 & 1 & 0 & z & 0 & -x \\ 0 & 0 & 1 & -y & x & 0 \end{pmatrix}.$$

Consider a three-dimensional problem consisting of two materials; a connected rigid material and a not-so-rigid material. Let us split the stiffness matrix $K = C + R$, with C containing one independent singular submatrix corresponding to the rigid material and R corresponding to the not-so-rigid material. Using rigid body modes deflation, the subspace $\mathcal{Z}$ is equal to the span of the null space of C, i.e., $\mathcal{Z} = \mathcal{N}(C) = \text{span}\{Z\}$ with $Z = \{z^1, \ldots, z^6\}$ the six base vectors corresponding to the six rigid body modes of the rigid material. This results in $Z \in \mathbb{R}^{n \times 6}$.

Assume the number of bodies to be n_b. Splitting $K = C + R$ provides us the possibility to decouple the matrix K into disjoint matrices C_i with $C = \bigcup_i^{n_b} C_i$ and mutual couplings R_i, with $R = \bigcup_i^{n_b} R_i$. If we choose matrices C_i on basis of material properties, then the

matrices C_i do not have irregular jumps in their coefficients.

$$K = C + R = \begin{pmatrix} \boxed{C_1} & & & \\ & \boxed{C_2} & & \\ & & \ddots & \\ & & & \boxed{C_{n_b}} \end{pmatrix} + R.$$

Recall Theorem 3 and 4 for the bounds on the effective condition number of the deflated system $\Pi^{\perp}K$ and the preconditioned deflated system $L^{-1}\Pi^{\perp}KL^{-T}$. The split of $K = C+R$ is here explicitly given.

Section 6 explains the identification of rigid bodies. More details on the rigid body modes and how to apply the modes can be found in Section 7.

6 Identifying rigid bodies

An approximate rigid body is a collection of connected elements in a finite element mesh that acts to a certain extent as a rigid body. The approximate rigid body modes of a model can be used in deflation or in coarse grid correction. These modes (vectors) can negatively influence the convergence of a Krylov subspace method. This section proposes a method applicable for general finite element packages to identify the approximate rigid bodies in a model.

6.1 Motivation

One-dimensional Poisson case
Consider the following illustrative one-dimensional Poisson problem on $V = \bigcup_{i=1}^{3} V_i$, where V_2 seperates V_1 and V_3:

$$\begin{aligned} -\frac{d}{dx}\left(c\frac{du}{dx}\right) &= f, \quad x \in V, \\ u &= 0, \quad x \in \partial V, \end{aligned} \tag{6.1}$$

where

$$c = \begin{cases} c_1 & \text{if } x \in V_1 \cup V_3 \\ c_2 & \text{if } x \in V_2 \end{cases},$$

u the unknown displacements and f the given external forces. Equation (6.1) can be discretized with the finite element method. Linear test functions and regular elements of size h results in the following discretization:

$$Ku = hf \tag{6.2}$$

where

$$K = \begin{pmatrix} 2c_1 & -c_1 & & & & & & & \\ -c_1 & \ddots & \ddots & & & & & & \\ & \ddots & 2c_1 & -c_1 & & & \emptyset & & \\ & & -c_1 & c_1+c_2 & -c_2 & & & & \\ & & & -c_2 & \ddots & \ddots & & & \\ & & & & \ddots & 2c_2 & -c_2 & & \\ & & & & & -c_2 & c_1+c_2 & -c_1 & \\ & & \emptyset & & & & -c_1 & \ddots & \ddots \\ & & & & & & & \ddots & 2c_1 & -c_1 \\ & & & & & & & & -c_1 & 2c_1 \end{pmatrix}.$$

Let us take $c_1 = 10$, $c_2 = 10^7$, $h = 1/40$ and V_i such that

$$c = \begin{cases} 10 & \text{if } x \in \{el_1, \ldots, el_{19}, el_{26}, \ldots, el_{40}\} \\ 10^7 & \text{if } x \in \{el_{20}, \ldots, el_{25}\} \end{cases}.$$

Note the jump in the coefficient of c around the 20-th and 25-th element.

Equation (6.2) can be solved by the Preconditioned Conjugate Gradient method, preconditioned by a Jacobi preconditioner (diagonal scaling) P with entries $p_{ii} = k_{ii}$. The spectrum of $P^{-1}K$ is illustrated in blue crosses in Figure 3.

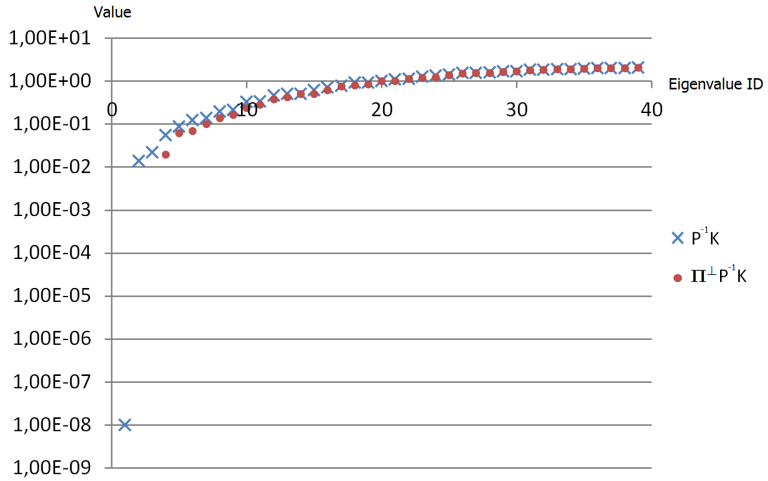


Figure 3: The eigenvalue spectrum of $P^{-1}K$ and $\Pi^{\perp}P^{-1}K$ for the one-dimensional Poisson problem.

The eigenvalues of $P^{-1}K$ are despite of diagonal scaling strongly diverse due to one relatively small eigenvalue. Recall that the convergence of Conjugate Gradient is influenced by the effective condition number $\kappa_{\text{eff}}(K) = \lambda_{\max}/\lambda_{\min}$ by Theorem 2. This basically means that the small eigenvalue of $P^{-1}K$ in Figure 3 results in slow convergence of CG. The convergence of Preconditioned CG (PCG) is illustrated in the dotted blue line in Figure 4. This figure shows that PCG converges just within the theoretical iteration bound (39) of

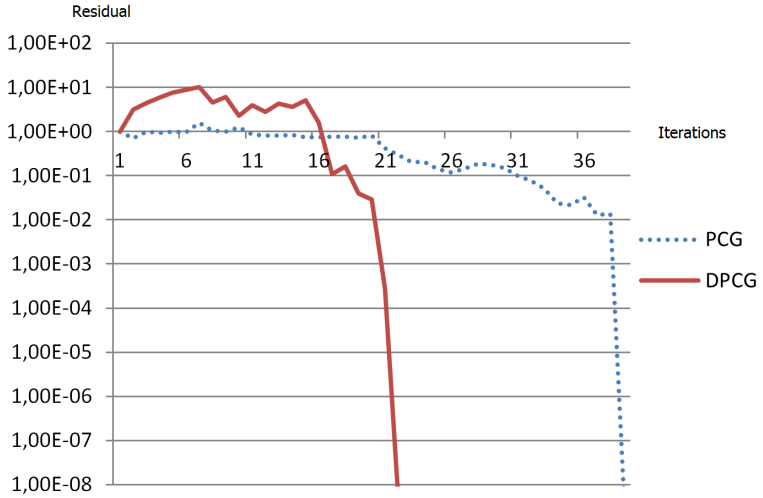


Figure 4: The convergence of PCG and DPCG for the one-dimensional Poisson problem.

the size of the problem [17].

The jumps in c have a bad influence on the eigenvalues of $P^{-1}K$. Deflation based on the rigid body modes as discussed in Section 5.5 can significantly improve this. By choosing the three uniform parts of the model as approximate rigid bodies, we can decouple the system into parts with constant coefficients c. The three corresponding one-dimensional rigid body modes are:

$$Z = \begin{pmatrix} 1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 1 & 0 & \vdots \\ 0 & 1 & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & 1 & 0 \\ \vdots & 0 & 1 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix}, \tag{6.3}$$

which in practice will be normalized. As can be seen in Equation (6.3), in one-dimensional problems rigid body modes deflation is equivalent to subdomain deflation. Figure 3 shows the eigenvalue spectrum of $\Pi^{\perp}P^{-1}K$ in red dots, excluding the three eigenvalues equal to zero[2] due to deflation. Note that the small eigenvalues of $P^{-1}K$ and $\Pi^{\perp}P^{-1}K$ are significantly different, but the large eigenvalues are not. The convergence of Deflated Preconditioned CG (DPCG) is illustrated in the solid red line in Figure 4.

The effective condition number decreases from $\kappa_{\text{eff}}(P^{-1}K) \approx 2 \cdot 10^8$ to $\kappa_{\text{eff}}(\Pi^{\perp}P^{-1}K) \approx 1 \cdot 10^2$. The number of CG iterations decreases from 39 to 22 iterations.

[2] Actually, these computed eigenvalues are of order 10^{-16}.

***SphereInCube* case**

Consider the three-dimensional *SphereInCube* case consisting of a stiff sphere located in a low stiffness cube, as illustrated in Figure 5. This case is modeled, evaluated and solved in DIANA.

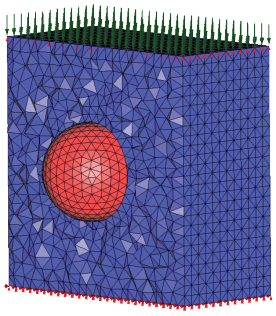


Figure 5: A section of the *SphereInCube* case with high stiffness material (red sphere) in low stiffness material (blue cube).

The *SphereInCube* case is a linear elastic model and built with tetrahedron elements. It has 36.129 degrees of freedom of which 33.425 degrees of freedom are really free (not constrained). The inner sphere is 10^6 stiffer than the surrounding cube. The cube is fixed at the bottom and at the top edges in the normal direction. A uniform load is applied on the top plane of the cube. The inner sphere with high stiffness acts, by approximation, as a rigid body in the low stiffness cube. The convergence of PCG using preconditioner IC(0) is shown as the dotted blue graph in Figure 6.

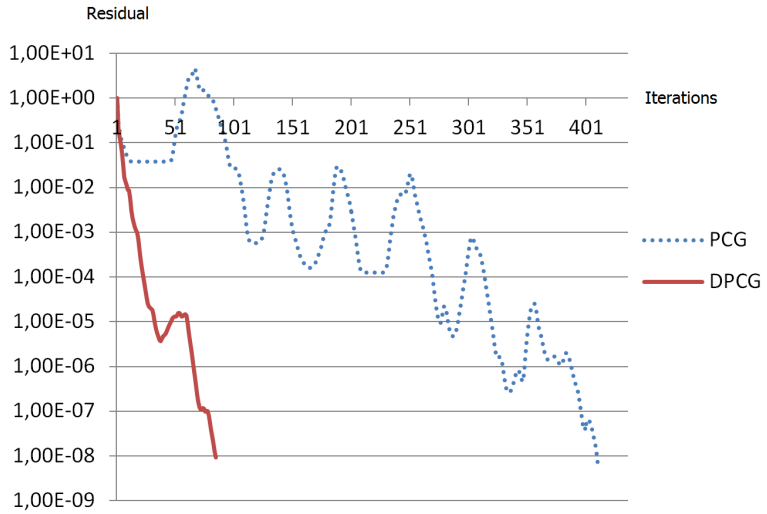


Figure 6: Convergence of PCG and DPCG for the case *SphereInCube*.

Figure 6 shows that PCG has six 'difficulties' in the convergence process. These jumps indicate the six modes in the model that are typically hard to find in Krylov subspace methods. Typically, Krylov subspace methods have difficulties with reducing the residual in the span of small eigenvectors.

The sphere is not a true rigid body in the model, but due to the high stiffness of the sphere, the corresponding modes are dominant in the model. By choosing the two uniform parts of the model as approximate rigid bodies, namely the sphere and the cube, we can decouple the system into parts with more or less constant coefficients in the stiffness matrix. The convergence of Deflated PCG (DPCG), preconditioned by IC(0) and deflated by the approximate rigid body modes, is shown as the red graph in Figure 6. The curious little jump in convergence for DPCG could be due to numerical rounding errors.

The number of iterations reduces from 410 to 84 and the residual decreases more gradual. Furthermore, the computation time reduces from 11.2 to 4.7 seconds, including the identification time for the approximate rigid bodies. The above analysis indicates that approximate rigid bodies can result in slow convergence of an iterative solution method. Identification of these bodies can be advantageous, since two techniques (deflation and coarse grid correction) can be used to improve this behavior.

The real rigid body modes of an unsupported model are eigenvectors corresponding to eigenvalue zero. Such vectors indicate a singular system. Approximate rigid bodies occur if a part of the model approximately floats free in the remainder of the model. This happens if a group of elements is mutually strongly coupled, but weakly coupled with its environment. This behavior can be caused by large stiffness jumps and by attempts to decouple parts of the model, e.g. for the purpose of domain decomposition. In general, domain decomposition methods result in subdomains that lack boundary conditions [20, 28]. The boundary conditions can be imposed (or corrected for) by using the rigid body modes of a complete subdomain.

The remainder of this section explains how to identify the approximate rigid bodies in a FEM model. Please note that the terms 'rigid body' and 'approximate rigid body' are used interchangeable for convenience if there is no ambiguity.

6.2 The coloring algorithm

The approximate rigid bodies are actually a collection of connected elements with a comparable stiffness. A classical method to identify the bodies is the so-called coloring algorithm. Each element consists of a material and the coloring algorithm groups together the connected elements of the same material. A relevant classical example of the coloring algorithm is described in Jönsthövel et al. [20], similar to Algorithm 6.

Algorithm 6. ***Coloring algorithm to identify rigid bodies***

Given the FE mesh with d materials and elements $\Lambda = \{\Lambda_1, \ldots, \Lambda_{n_{el}}\}$.
Define element i as $\Lambda_i = \{\chi_{\Lambda_i}, \theta_{\Lambda_i}, \gamma_{\Lambda_i}\}$, with

- *neighboring elements, χ_{Λ_i}, containing indices of neigboring elements of element Λ_i.*
- *material type, θ_{Λ_i}.*
- *rigid body, γ_{Λ_i}.*

```
1   For j = 1, ..., d, Do
2     set d_j = 0
3     For i = 1, ..., n_el, Do
4       select element Λ_i with χ_Λi, θ_Λi and γ_Λi.
5       assign_element_to_body(Λ_i) {
6       If θ_Λi = j, Then
7         If γ_Λi = 0, Then
8           d_j = d_j + 1, set γ_Λi = d_j
9         EndIf
10        For all Λ_k ∈ χ_Λi, Do
11          select element Λ_k with χ_Λk, θ_Λk and γ_Λk.
12          If θ_Λk = j, Then
13            If γ_Λk = 0, Then
14              set γ_Λk = γ_Λi,
15              put element Λ_k on the heap of element Λ_i.
16            EndIf
17          EndIf
18        EndFor
19      EndIf
20      }
21      For all Λ_k on the heap of Λ_i, Do
22        assign_element_to_body(Λ_k)
23      EndFor
24    EndFor
25  EndFor
```

Algorithm 6 starts with defining an element as a rigid body and seeks all connected elements of the same material by sequentially checking all neighbors of those elements. If all connected elements of the same material are identified and labeled as a rigid body, an unlabeled element is defined as a new rigid body. Thereafter, all connected elements of the same material as the new rigid body are identified and labeled as a rigid body. This process repeats itself until all elements are contained in a rigid body.

On one hand, it is advantageous to eliminate all stiffness jumps in the corresponding system of equations to improve the convergence of an iterative solution method. On the other hand, increasing the number of rigid bodies results in larger matrices Z and E. This can increase the computation time required for one iteration significantly. Therefore, a proper balance between stiffness jumps and the number of rigid bodies is required for optimal performance.

For this reason the number of bodies should be limited to a maximum. Let this maximum

number of allowed bodies be n_b bodies. In case of domain decomposition, each subdomain may contain n_b bodies. Actually, no domain decomposition is equivalent with one subdomain. Each subdomain combines neighboring bodies found by Algorithm 6, based on their material properties, until n_b bodies remain for each subdomain. The assembled bodies per subdomain form the complete subdomain. In other words, the approximate rigid body caused by the decoupling of the subdomain can be constructed by the assembly of all bodies in the subdomain.

Figure 7 presents a model with two non-overlapping subdomains divided by the dashed line. The complete model Ω consists of four bodies and each of the subdomains Ω_1 and Ω_2 consists of three bodies. Bodies Ω_1^a, Ω_1^b and Ω_1^c make up for the complete subdomain Ω_1 and bodies Ω_2^a, Ω_2^b and Ω_2^c make up for the complete subdomain Ω_2. If in this case the maximum number of bodies $n_b = 3$, then all bodies can be used by the iterative solver.

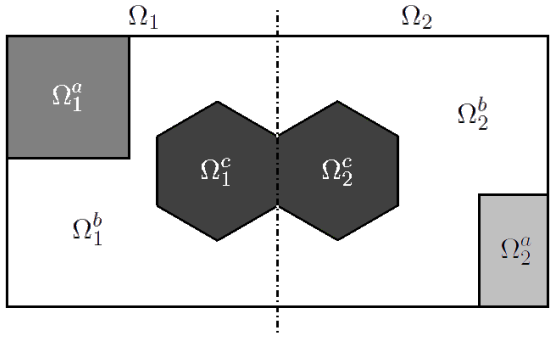


Figure 7: A model consisting of four bodies divided into two subdomains, each consisting of three bodies.

Unfortunately, Algorithm 6 is insufficiently applicable in a general FEM software package. The most important short-coming of the classical coloring algorithm is a proper criterion to classify the different 'materials' in the model, since a wide variety of elements and materials exist.

Element can be described by various parameters, such as Young's modulus, time, previous (local) stresses, temperature, pressure, etc. This implies that one material can lead to large behavioral differences throughout the model. This also implies that two different materials can actually behave similarly.

Furthermore, in some cases it is not possible to compare elements on the previously mentioned parameters. It is not obvious how to compare a classical structural element with an interface element, which can model various phenomena, such as elastic bedding, cracking, bond-slip along reinforcements, friction between surfaces, joints in rocks, contact and fluid-structure relations. In general, the relation between e.g. a structural element and an interface element depends on the element sizes and the cross-sectional area through which the force is applied are also involved, as described in Section 2 in Equation (2.13) and (2.14).

It may be possible to incorporate every relevant parameter that determines the behavior of an element. Nevertheless, for general FEM computations it is important to identify the rigid bodies in an easy and broad applicable way. The rigid body identification should easy

to understand and easy to maintain.

6.3 Generalizing the coloring algorithm

This section describes how to compare elements in any model and how to use this comparison to identify rigid bodies in the model.

6.3.1 Comparing elements

The issue that relates to approximate rigid bodies in the iterative solution method is the large stiffness jumps in the global stiffness matrix K. A group of elements corresponding to an approximate rigid body will have a dominant contribution to the matrix K compared to its surrounding. This relatively large contribution can be caused by various relevant parameters, such as the Young's modulus of an isotropic material, a so-called dummy stiffness D_m of an interface element or the spring constant of a spring element.

As an illustration, consider the rigidity matrices D_m^{iso} of an isotropic structural element and D_m^{int} of an interface element. The rigidity matrix D_m^{iso} for a three-dimensional structural element modeling isotropic behavior equals

$$D_m^{\text{iso}} = \frac{E}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & (1-2\nu)/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1-2\nu)/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & (1-2\nu)/2 \end{pmatrix},$$

where E is Young's modulus and ν is Poisson's ratio. The rigidity matrix D_m^{int} for an interface element equals

$$D_m^{\text{int}} = \begin{pmatrix} D_{11} & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & D_{33} \end{pmatrix},$$

where D_{ii} is the user's input.

All relevant parameters will eventually result in an element stiffness matrix. The element stiffness matrix is always present if the element contributes to the global stiffness matrix. The element stiffness matrices are formed as follows:

$$K^{e_m} = \int_{e_m} B_m^T D_m B_m \, dV.$$

These matrices K^{e_m} can be mutually compared in a fair and relatively easy way. In general the diagonal of the element stiffness matrix is leading for the stiffness of the element.

An attempt to describe an element stiffness matrix with one value is to take the maximum, the sum or the average of the element stiffness diagonal. The sum of the element stiffness diagonal is the trace by definition.

Theorem 6. *Let K^{e_m}, $m = 1, \ldots, n_{el}$ denote the element stiffness matrices of the global stiffness matrix $K \in \mathbb{R}^{n \times n}$ such that $\sum_{m=1}^{n_{el}} K^{e_m} = K$. Then the following holds:*

$$\sum_{m=1}^{n_{el}} tr(K^{e_m}) = \sum_{j=1}^{n} \lambda_i(K).$$

Proof. Note that the sum of the assembled element stiffness matrices is the global stiffness matrix.

$$\begin{aligned}\sum_{m=1}^{n_{el}} \mathrm{tr}(K^{e_m}) &= \sum_{m=1}^{n_{el}} \sum_{i} K_{ii}^{e_m}, \\ &= \sum_{i=1}^{n} K_{ii}, \\ &= \mathrm{tr}(K).\end{aligned}$$

The characteristic polynomial of K can be written as $p(\lambda) = (-\lambda)^n + tr(K)\lambda^{n-1} + \ldots + \det(K)$, and also as $p(\lambda) = (-1)^n(\lambda - \lambda_1)\cdots(\lambda - \lambda_n)$. Comparing the λ^{n-1} coefficients of both expressions yields $\sum_{i=1}^{n} \lambda_i = tr(K)$, which concludes the theorem. □

This means that the trace of an element stiffness matrix K^{e_m} directly contributes to the sum of the eigenvalues of the global stiffness matrix K. We suggest to take the average of the element stiffness diagonal, since this also relates to the classical approach based on Young's modulus.

6.3.2 Identifying bodies

An effective approach is to iteratively define the rigid bodies based on the relative stiffness difference of neighboring elements. The parameter δ indicates when two neighboring elements are put in the same body; if their stiffness relatively differs less then $\delta \in \mathbb{R}$, then the two elements are put in the same body. The parameter δ is initialized based on the minimum and maximum stiffness occurring in the model. A minimum value for the parameter δ is set (e.g. $\delta \geq 100$). If the number of identified bodies is too large, then δ is increased and the rigid body identification is re-applied until a reasonable amount of bodies is identified. The steps are summarized in the illustrative Algorithm 7.

Algorithm 7. ***Extended coloring algorithm to identify rigid bodies***

1 *Compute the average of the element stiffness diagonals.*
2 *Estimate the δ parameter.*
3 *Apply a variant of Algorithm 6: Combine neighboring elements to bodies if their stiffness differs less than factor δ.*
4 *If the number of bodies is much larger than n_b, then increase δ and goto 3.*
5 *Combine identified bodies until n_b bodies per subdomain remain.*

This approach is intuitively clear; the stiffness of an approximate rigid body should be much larger than the stiffness of its neighboring elements, which implies a "stiffness jump" at the boundary of the rigid body. The internal coupling of the degrees of freedom in the approximate rigid body is significantly stronger then the coupling of the degrees of freedom between the rigid body and its surrounding elements. Therefore, the rigid body floats relatively free, if no other boundary conditions are attached to the body itself. In that case,

the surrounding elements effectively put Dirichlet boundary conditions on the rigid body.

An alternative (less effective) approach to identify rigid bodies is based on "stiffness ranges" in the model. More information can be found in Appendix C.

6.4 Limitation of rigid bodies

Although the approximate rigid body approach is often effective, some limitations should be addressed.

1. **Gradual versus sudden stiffness increases.** A simple preconditioning matrix such as Jacobi can deal with gradual stiffness increases throughout the model. The elements of the stiffness matrix can be properly scaled in that case. Convergence problems do occur if stiffness jumps are induced by the model. Figure 8 illustrates how a gradual stiffness increase (as the extension of the coloring algorithm detects it) can ultimately lead to a large stiffness difference. For the case in Figure 8, it could be wise to define two or three bodies.

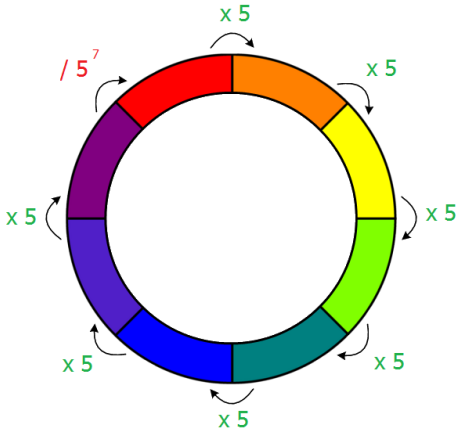


Figure 8: An annular finite element model where gradual stiffness increases can result in large stiffness jumps.

2. **Scalar degrees of freedom of another order of magnitude**. The rigid body modes approach in general can identify and improve systems of equations with a small number of approximate rigid bodies. This approach is unsuitable for models where stiffness differences occur throughout a large part of the model.

   Let us illustrate this limitation by a model with pressure degrees of freedom that occur throughout a large part of the model. These scalar degrees of freedom interact with other (translational) degrees of freedom, which have other units and order of magnitude. All these scalar degrees of freedom can be seen as rigid bodies, as each pressure degree of freedom implies a large stiffness jump in the matrix. As the number of pressure degrees of freedom can be very large, it is unwise to define each of them as an approximate rigid body, since the size of Z will grow beyond its effectiveness.

   In short, scalar degrees of freedom with another order of magnitude than the degrees of freedom that they interact with, should not defined as a separate rigid body. The approximate rigid body approach cannot improve convergence for those cases.

6.5 Reusing rigid bodies in nonlinear iterations

The system of equations in a nonlinear iteration loop may change only slightly or not at all. For a preserved system, e.g. constant stiffness, it is obvious how to reuse information. A direct solver can reuse the matrix decomposition and an iterative solver can reuse the preconditioner and deflation operator. It is less obvious if and how to reuse information in a slightly changing system of equations, for example in a relative linear Newton iteration.

The extended coloring algorithm described by Algorithm 7 starts with computing the average trace of each element stiffness matrix. Let n_{el} be the number of elements, $\gamma \in \mathbb{R}$ the nonlinear tolerance parameter and let $y_i \in \mathbb{R}^{n_{el}}$ be a vector. The vector y_i contains the average trace of each element stiffness matrix in nonlinear iteration i. By storing vector y_i, each element can be compared with y_{i+1} in the next nonlinear iteration. The previously identified rigid bodies can be reused if the element stiffness difference is small:

$$\frac{|y_{i+1}(j) - y_i(j)|}{\min(y_{i+1}(j), y_i(j))} < \gamma, \qquad \forall j = 1, \ldots, n_{el}, \tag{6.4}$$

provided that y_{i+1} and y_i do not contain zero entries. Inequality (6.4) checks whether all vector entries differ less than factor $1 + \gamma$. Some numerical experiments suggest $\gamma = 1$, resulting in a bound of factor-two relative stiffness change.

7 Rigid bodies within the solution method

7.1 The rigid body modes

The rigid body modes are spanned by the kernel base vectors of the corresponding element stiffness matrix. The rigid body modes are the eigenvectors corresponding to eigenvalue zero. The null space of the element matrices can therefore be approximated by the rigid body modes of the element matrices. The rigid body motions (in three dimensions) are given by three translations and three rotations plus any scalar degrees of freedom types, such as temperature or pressure.

Equation (7.1) shows the eight rigid body modes of a node with eight degrees of freedom. The node consists of three translation degrees of freedom u_x, u_y and u_z, three rotational degrees of freedom ϕ_x, ϕ_y and ϕ_z and two scalar degrees of freedom p and T corresponding to pressure and temperature. Translation and rotational degrees of freedom are not necessarily coupled in the model, but they are coupled in the rigid body rotational modes.

$$\begin{array}{c} u_x \\ u_y \\ u_z \\ \phi_x \\ \phi_y \\ \phi_z \\ p \\ T \end{array} \begin{pmatrix} 1 & 0 & 0 & 0 & -z & y & 0 & 0 \\ 0 & 1 & 0 & z & 0 & -x & 0 & 0 \\ 0 & 0 & 1 & -y & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.1}$$

For the sake of completeness, all rigid body modes should be correctly oriented with respect to the orientation of the nodes (which could differ from $\{(1,0,0),(0,1,0),(0,0,1)\}$). Each column represents the null vector corresponding to the translation in x-, y- and z-direction,

linearized rotation in x-, y- and z-direction and pressure and temperature. Each row represents one of the eight degrees of freedom of the node. Each body (collection of elements) imply $6 + s$ deflation vectors in three dimensions, as each body has three translational and rotational degrees of freedom [20] plus s types of scalar degrees of freedom.

The rigid body modes of a single element are the combination of the vectors in Equation (7.1) for all nodes contained in that element. Sets of elements make up the bodies. The rigid body modes of a collection of elements is equal to the assembly of the rigid body modes of the individual elements, ignoring their overlap. Nodes that do not uniquely belong to one element are assigned to the stiffest element, so that nodes that belong to two bodies will be assigned to the stiffest body.

7.2 Deflation versus coarse grid correction

The coarse grid correction described in Section 4.3 and rigid body mode deflation described in Section 5.5 are analogous to a large extent. Recall that the stand-alone coarse grid preconditioner is $P_C = I + ZE^{-1}Z^T$ as in Equation (4.1). The preconditioned coarse grid preconditioner is $P_{C,P^{-1}} = P^{-1} + ZE^{-1}Z^T$ as in Equation (4.2). Recall that the deflation operator is $\Pi^{\perp} = I - KZE^{-1}Z^T$.

The coarse grid correction in DIANA takes the rigid body modes of the subdomains as base for Z. This can be extended in such a way that it not only takes the subdomains as bodies, but that each subdomain contains up to n_b bodies. This means that the matrix Z of coarse grid correction and deflation can be chosen *equal*.

Algorithm 8 provides an algorithm where one can choose deflation **[D]** (DPCG) or coarse grid correction in the form of a second preconditioner **[P]** (PPCG) to use the rigid body modes. If deflation **[D]** is applied, then the coarse grid correction **[P]** should be skipped and vice versa[3]. The preconditioners are additively applied and can be replaced by a multiplicative version as in Equation (4.3).

[3]Some numerical experiments suggest that applying both deflation and coarse grid correction is possible, but without gain.

Algorithm 8. ***PCG menu: DPCG [D] or PPCG [P]***

	1	*Choose* u_0. *Compute* $r_0 = f - Ku_0$.
[D]	2	*Set* $r_0 := \Pi^{\perp} r_0$.
	3	*Solve* $Pz_0 = r_0$ *and set* $p_0 = z_0$.
[P]	4	*Compute* $\tilde{z}_0 = ZE^{-1}Z^T r_0$ *and set* $p_0 := p_0 + \tilde{z}_0$.
	5	*For* $j = 0, 1, \ldots$ *until convergence, Do*
	6	$z_j = Kp_j$
[D]	7	$z_j := \Pi^{\perp} z_j$
	8	$\alpha_j = \langle r_j, z_j \rangle / \langle p_j, z_j \rangle$
	9	$u_{j+1} = u_j + \alpha_j p_j$
	10	$r_{j+1} = r_j - \alpha_j z_j$
	11	*Solve* $P_1 z_{j+1} = r_{j+1}$
[P]	12	*Compute* $\tilde{z}_{j+1} = ZE^{-1}Z^T r_{j+1}$ *and set* $z_{j+1} := z_{j+1} + \tilde{z}_{j+1}$.
	13	$\beta_j = \langle r_{j+1}, z_{j+1} \rangle / \langle r_j, z_j \rangle$
	14	$p_{j+1} = z_{j+1} + \beta_j p_j$
	15	*EndDo*
[D]	16	$u = ZE^{-1}Z^T f + \Pi^{\in} u_{j+1}$

Algorithm 8 shows at what moment deflation and coarse grid correction are applied in PCG. GMRES(s) can be modified analogously. The application of the techniques is as follows:

Deflation	Coarse grid correction
$y = x - KZE^{-1}Z^T x$	$y = ZE^{-1}Z^T x$

In both techniques the coarse matrix $E = Z^T KZ$ acts as a representation of the rigid bodies in the subdomains and in both techniques the coarse system should be efficiently solved. In Nabben et al. [30, 31] a comparison is made between coarse grid correction and deflated preconditioning. The following theorem and proof are from [30]:

Theorem 7. *Let* $K \in \mathbb{R}^{n \times n}$ *be symmetric positive definite and* $Z \in \mathbb{R}^{n \times m}$ *be full rank such that span*$\{Z\} = \mathbb{N}(C)$, *where* $C = K - R$ *is positive semidefinite. Let* $\Pi^{\perp} = I - KZE^{-1}Z^T$ *be the deflation operator and* $P_C = I + ZE^{-1}Z^T$ *be the coarse grid preconditioner. Then the following holds:*

$$\lambda_1(\Pi^{\perp} K) = \ldots = \lambda_r(\Pi^{\perp} K) = 0, \tag{7.2}$$

$$\lambda_n(\Pi^{\perp} K) \leq \lambda_n(P_C K), \tag{7.3}$$

$$\lambda_{r+1}(\Pi^{\perp} K) \geq \lambda_1(P_C K). \tag{7.4}$$

Proof. By the proof of Theorem 3, $\Pi^{\perp} K$ is positive semidefinite. All eigenvalues of $\Pi^{\perp} K$ are therefore real and non-negative. Since $\Pi^{\perp} KZ = KZ - KZE^{-1}Z^T KZ = 0$, Equation (7.2) holds. Consider

$$K^{\frac{1}{2}} P_C K^{\frac{1}{2}} - \Pi^{\perp} K = KZE^{-1}Z^T K + K^{\frac{1}{2}} ZE^{-1}Z^T K^{\frac{1}{2}}.$$

The right-hand side is positive semidefinite, so for the left-hand side holds that $\lambda_i(K^{\frac{1}{2}} P_C K^{\frac{1}{2}}) \geq \lambda_i(\Pi^{\perp} K)$ by Corollary 7.7.4 in Horn et al. [18]. Since the spectra of $P_C K$ and $K^{\frac{1}{2}} P_C K^{\frac{1}{2}}$ are equal, we obtain

$$\lambda_i(P_C K) = \lambda_i(K^{\frac{1}{2}} P_C K^{\frac{1}{2}}) \geq \lambda_i(\Pi^\perp K),$$

which proves Inequality (7.3). Now consider

$$\begin{aligned} P_C K P_C - \Pi^\perp K &= K + ZE^{-1}Z^T K + KZE^{-1}Z^T + ZE^{-1}Z^T KZE^{-1}Z^T \\ &\quad - K + KZE^{-1}Z^T K \\ &= ZE^{-1}Z^T K + KZE^{-1}Z^T + ZE^{-1}Z^T + KZE^{-1}Z^T K \\ &= (K+I)ZE^{-1}Z^T(K+I). \end{aligned}$$

This shows that $P_C K P_C - \Pi^\perp K$ is symmetric and of rank m. By Theorem 4.3.6 in Horn et al. [18] follows that

$$\lambda_{r+1}(\Pi^\perp K) \geq \lambda_1(P_C K P_C) = \lambda_1(P_C^2 K).$$

Since $P_C - I$ is positive semidefinite, $P_C^2 - P_C$ and $K^{\frac{1}{2}} P_C^2 K^{\frac{1}{2}} - K^{\frac{1}{2}} P_C K^{\frac{1}{2}}$ are also positive semidefinite. Hence,

$$\lambda_i(P_C^2 K) = \lambda_i(K^{\frac{1}{2}} P_C^2 K^{\frac{1}{2}}) \geq \lambda_i(K^{\frac{1}{2}} P_C K^{\frac{1}{2}}) = \lambda_i(P_C K).$$

Combining the inequality above gives

$$\lambda_{r+1}(\Pi^\perp K) \geq \lambda_1(P_C^2 K) \geq \lambda_1(P_C K).$$

□

From Theorem 7 follows that

$$\kappa_{\text{eff}}(\Pi^\perp K) \leq \kappa(P_C K),$$

so deflated CG converges faster than CG combined with coarse grid correction, for arbitrarily full rank $Z \in \mathbb{R}^{n \times m}$. Theorem 7 can extended to a preconditioned version.

Theorem 8. *Let $K \in \mathbb{R}^{n \times n}$ and P^{-1} be symmetric positive definite and $Z \in \mathbb{R}^{n \times m}$ be full rank. Let $\Pi^\perp = I - KZE^{-1}Z^T$ be the deflation operator and $P_{C,P^{-1}} = P^{-1} + ZE^{-1}Z^T$ be the coarse grid preconditioner. Then the following holds:*

$$\lambda_n(P^{-1}\Pi^\perp K) \leq \lambda_n(P_{C,P^{-1}} K), \tag{7.5}$$

$$\lambda_{r+1}(P^{-1}\Pi^\perp K) \geq \lambda_1(P_{C,P^{-1}} K). \tag{7.6}$$

The proof is given in Nabben et al. [30]. From Theorem 8 follows that

$$\kappa_{\text{eff}}(P^{-1}\Pi^\perp K) \leq \kappa(P_{C,P^{-1}} K),$$

so deflated PCG (DPCG) converges faster than PCG combined with coarse grid correction (PPCG).

Table 2 gives an overview of the strong and weak characteristics of deflation and coarse grid correction.

	Deflation	Coarse grid correction
Cost per iteration	+[4]	+
Scalability with # threads	– [4]	+
Effectiveness	+	–
Sensitivity to numerical issues	–	+

Table 2: Comparison of the weak and strong characteristics of deflation and coarse grid correction.

The costs per iteration for deflation are larger due to an extra vector addition and a multiplication with K. This extra multiplication can be performed beforehand by pre-computing $K_Z = KZ$, which considerably reduces the extra costs per iteration. Appendix B.2 indicates that pre-computing KZ is efficient, especially for a small number of threads (< 5). On the other hand, in parallel computing this strategy poorly scales with the number of threads. For each additional thread i, *each* thread has to store the contribution K_Z^i of thread i. Details on parallel deflation follow in Section 8.2.

Deflation is more powerful than coarse grid correction, if properly used. This can be seen by the effective condition number of the system and by the number of iterations of an iterative solver. It is be proven that, with arbitrary full rank matrix Z, the effective condition number for deflation is always below the condition number of the system preconditioned by the coarse grid correction [30].

A disadvantage of deflation is that it is more sensitive to numerical errors. The projection operation in deflation has to be very accurate. If a small part of a deflation vector is not completely projected out of the system, then it will never leave the Krylov subspace. In that case, the iterative method possibly never converges at all. The coarse grid correction allows for inaccuracy, only the correction will be less effective. The next iterations are likely to improve this ineffective correction.

For a performance comparison of deflation and coarse grid correction in DIANA, please refer to Section 9. Additional theoretical background can be found in Nabben et al. [30, 31].

8 Implementation

DIANA-specific implementation details can be found in Appendix A.1.

8.1 Parallel computers

A parallel computer is a collection of processors that is able to work cooperatively to solve a computational problem [10]. Three classes of parallel computers exist: shared memory computers, distributed memory computers and virtual shared memory computers.

A shared memory computer or Symmetrical Multi-Processor (SMP) consists of multiple processors sharing the same memory. Each processor can access all memory addresses through a bus, as illustrated in the upper left figure in Figure 9 acquired from Lingen [27]. The memory access time increases with the number of processors due to the capacity of

[4] if KZ is pre-computed as $KZ = K_Z$.

the bus. Therefore, each processor has its own fast cache memory. Typically, the number of processors ranges from 2 to 32.

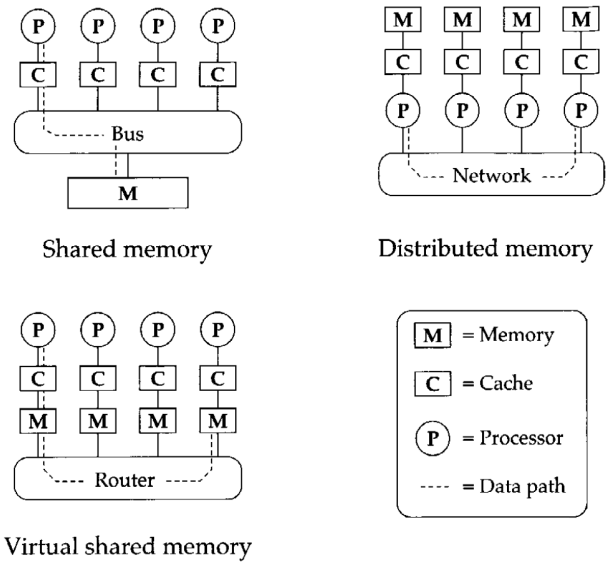


Figure 9: The three classes of parallel computers.

A distributed memory computer consists of multiple processors, each having its own memory. The processors interact with each other through a communication network, as illustrated in the upper right figure in Figure 9. The performance of a distributed memory computer is strongly influenced by the performance of the communication network. This can be expressed in terms of latency and bandwidth. Latency is the time required to start an interaction between two processors and the bandwidth is the number of bytes that can be transferred over the network within one second. The number of processors can be very large, since no restrictive bus is required.

A virtual shared memory computer combines the properties of a shared memory computer with that of a distributed memory computer. It has a memory architecture that is physically distributed, but the memory can be accessed by all processors. Each processor can directly access the memory of any other processor through a routing network, as illustrated in the lower left figure in Figure 9.

The implementation in DIANA targets shared memory computers. This architecture represents most modern desktop computers and is relatively easy to implement within existing software. In the shared memory programming model, a parallel program consists of concurrent threads, each executing its own sequential program. In the case of DIANA, all tasks execute the same sequential program, which classifies the implementation of DIANA as 'Single Program, Multiple Data' (SPMD). This model supports all shared memory and virtual shared memory computers, as long as each computer provides the same shared memory programming language or software libraries.
Two standard software libraries are POSIX Threads software library [5] and OpenMP [7]. The customized software library in DIANA is implemented by Lingen [27] and is based on OpenMP.

8.2 Parallel computations

Recall the left and right restriction operators L_i and R_i as introduced in Section 4.3 with

$$K = \sum_{i=1}^{n} L_i^T K_i R_i.$$

Both L_i and R_i are the same size with the number of columns equal to the global size and the number of rows equal to the local size (of subdomain i). The right restriction operator can be used to extract subdomain matrices/vectors from the global matrices/vectors:

$$\begin{aligned} K_i &= R_i K R_i^T, \\ x_i &= R_i x. \end{aligned}$$

The left operator can be used to assemble the local vectors into a global vector, which is useful for inner products:

$$x = \sum_{i=1}^{n} L_i^T x_i.$$

Parallel computations involve three different types of data exchange. The different types of data exchange will be explained through three examples: an inner product, a matrix-vector product involving stiffness matrix K and a matrix-vector product involving approximate null space Z.

Parallel: inner product
A parallel inner product yields

$$s = x^T y = \sum_{i=1}^{n} x_i^T L_i L_i^T y_i \tag{8.1}$$

The inner product computation requires to evaluate matrix $L_i L_i^T$. This is a boolean diagonal matrix containing non-zero values for the unique set of degrees of freedom in the i-th subdomain. In other words, $L_i L_i^T$ only preserves the internal degrees of freedom and disregards all overlapping degrees of freedom.
This is in alignment with the intuitive interpretation that the multiplication of all entries in the global vectors is evaluated exactly once. After the local inner products, the local sums are synchronized (added up) to obtain the global inner product.

Parallel: matrix-vector product with stiffness matrix
A parallel matrix-vector product with K yields

$$\begin{aligned} y_j &= R_j y = R_j K x \\ &= R_j \sum_{i=1}^{n} L_i^T K_i R_i x \\ &= \sum_{i=1}^{n} R_j L_i^T K_i x_i \end{aligned}$$

The matrix $R_j L_i^T$ is non-zero if subdomain i shares degrees of freedom with subdomain j. In other words, $R_j L_i^T$ communicates and contributes between subdomains in one direction: only domain i can influence domain j, but not the other way around.
The computation is performed in two steps. Firstly, each local thread computes $K_i x_i$. Secondly, the contributions $K_j x_j$ of all domains are exchanged, corresponding to the shared degrees of freedom between subdomains. This data exchange is one-directional.

Parallel: the approximate null space matrix

Following notation of Lingen et al. [28], the coarse restrictor operators $C_i \in \mathbb{R}^{6\times(6\cdot n_d)}$ extracts a coarse subdomain vector from a global coarse vector by applying $x_i = C_i x$. Furthermore, just as the left restriction operator L_i, C_i has a unique set of non-zero columns such that

$$C_i C_j^T = \begin{cases} 0 & \text{if } i \neq j, \\ I & \text{if } i = j. \end{cases}$$

Now, the matrix Z can be written as

$$Z = \sum_{i=1}^{n} R_i^T Z_i C_i,$$

where Z_i is the local approximate null space of K_i, i.e., the local rigid body modes. A parallel matrix-vector product with Z yields

$$\begin{aligned} y_j &= R_j y = R_j Z x \\ &= R_j \sum_{i=1}^{n} R_i^T Z_i C_i x \\ &= \sum_{i=1}^{n} R_j R_i^T Z_i x_i \end{aligned}$$

The matrix $R_j R_i^T$ is non-zero if the subdomain i and j share common degrees of freedom. In other words, $R_j R_i^T$ indicates symmetric communication between subdomains.
The computation is performed in two steps. Firstly, each local thread computes $Z_i x_i$. Secondly, the contributions $Z_i x_i$ of all domains are exchanged, corresponding to the shared degrees of freedom between subdomains. This data exchange is symmetric and thus two-directional.

Let us illustrate the consequence of domain decomposition on the (local) null spaces. Assume two domains of uniform material, resulting in two null spaces Z_1 and Z_2 which are non-zero for the corresponding domains. In order to compute the correct null space of each domain, we scale the border of the subdomain Z by the diagonal of K. This approach is also called weighted overlap [47]. To be precise, define K_i the stiffness matrix of domain i and K_i^α the diagonal element of K_i at border degree of freedom α. Consider border degree of freedom α, present in domains i and j, such that $K^\alpha = K_i^\alpha + K_j^\alpha$. Then (until so far globally determined) Z_i^α and Z_j^α are scaled in the following way:

$$\begin{aligned} Z_i^\alpha &:= \frac{K_i^\alpha}{K_i^\alpha + K_j^\alpha} Z_i^\alpha, \\ Z_j^\alpha &:= \frac{K_j^\alpha}{K_i^\alpha + K_j^\alpha} Z_j^\alpha. \end{aligned} \tag{8.2}$$

Due to overlapping domains (in terms of nodes), note that the above scaling results in the correct global assembled null space Z, as all overlapping nodes add up to one.

8.2.1 Deflation

Since the multiplication with KZ is required each iteration, it is advantageous to compute KZ in advance. A performance comparison is discussed in Appendix B.2. The following steps are analogue to the approach of Lingen [28]. Let us compute b_i^j, column j of KZ.

$$\begin{aligned} b_i^j &= KZe^j \\ &= R_i \sum_{l=1}^{n} L_l^T K_l R_l \sum_{k=1}^{n} R_k^T Z_k C_k \; e^j. \end{aligned}$$

1. Form the coarse subdomain vector $e_k^j = C_k e^j$.
2. Compute $\tilde{a}_k^j = Z_k e_k^j$.
3. Exchange data to obtain $a_l^j = \sum_{k=1}^{n} R_l R_k^T \tilde{a}_k^j$.
4. Compute $\tilde{b}_l^j = K_l a_l^j$.
5. Exchange data to obtain $b_i^j = \sum_{l=1}^{n} R_i L_l^T \tilde{b}_l^j$.

Each thread stores all partial columns of KZ. The vector $y = KZx$ is computed by data exchange after the local computations

$$y_i = b_i x.$$

Data exchange is required for each thread, resulting that the matrix KZ grows *with each thread*. This is a severe disadvantage if a large number of threads is used.

Deflation also requires to evaluate $y = Z^T K x$, one time after convergence. In the symmetric case this can be done by $Z^T K = (KZ)^T$. The resulting vector $y = (KZ)^T x$ is in fact a number of inner products, which can be computed analogue to Equation (8.1):

$$y(j) = (b^j)^T x = \sum_{i=1}^{n} (b_i^j)^T L_i L_i^T x_i.$$

In the non-symmetric case, $y = Z^T K x$ is just computed by first multiplying with K and thereafter with Z^T.

The coarse matrix $E \in \mathbb{R}^{(6 \cdot n_d) \times (6 \cdot n_d)}$ can be computed by pre-multiplying b^j with Z^T, so that each thread i stores the partial columns $E_i \in \mathbb{R}^{6 \times (6 \cdot n_d)}$.

8.3 The condition number of the coarse matrix

The robustness of both deflation and coarse grid correction strongly depend on the quality of the rigid bodies and the quality of the coarse matrix $E = Z^T K Z$. To successfully project or correct for the rigid body modes, the coarse matrix E must be nonsingular.

In case of deflation, the projection needs to be very accurate to completely project the rigid body modes out of the system. If only the slightest span of a projected vector remains in the system, then this part can freely influence the solution vector. An unsuccessfully projected vector will never leave the Krylov subspace and therefore, it is unlikely that the iterative solution method will ever converge.

In case of coarse grid correction, the correction is allowed to be less accurate. The correction may lose accuracy, but as long as the main direction is correct, there is no serious harm in using coarse grid correction. Unlike deflation, the coarse grid correction can be seen as a preconditioner, where accuracy is of lesser importance.

8.3.1 Improving the condition number of the coarse matrix

The condition number of $E = Z^T K Z$ can be effectively improved by adjusting the approximate null space Z. It is clear that if Z is not full rank, then this results in a singular matrix E. The choice of non-overlapping bodies per domain ensures that no problem can occur on each domain. On the other hand, due to overlapping elements in the domain decomposition, some small bodies located in more than one domain can result in (almost) linearly dependent vectors in Z. Such overlapping bodies can dramatically increase the condition number $\kappa(E)$.

Figure 10 shows a two-dimensional model with overlapping partitioning. The global model Ω consists of two rigid bodies Ω^a and Ω^b. The partitioning provides two subdomains Ω_1 and Ω_2, each consisting of two rigid bodies. Note that due to overlap, body Ω_1^b and Ω_2^b are the same body, but regarded by the iterative solver as two bodies.

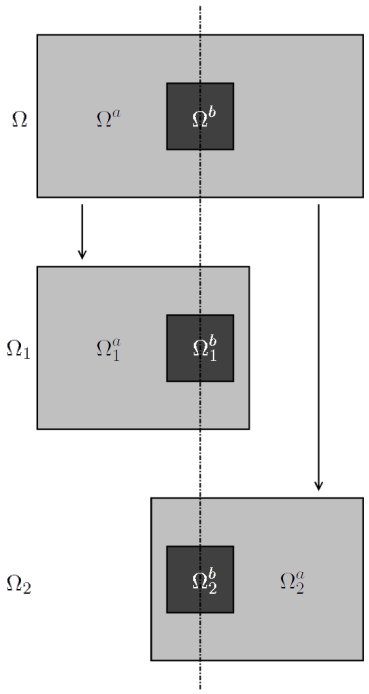


Figure 10: A model with two rigid bodies is partitioned into two overlapping subdomains.

The nonzero elements of the globally assembled null space matrix Z of the model in Figure 10 are illustrated in Equation (8.3).

$$
Z = \begin{pmatrix} Z_1 & \\ & Z_2 \end{pmatrix}
= \begin{pmatrix} Z_1^a & & & \\ & Z_2^a & & \\ & Z_1^b & & Z_2^b \\ Z_1^a & & & \\ & & Z_2^a & \end{pmatrix}. \tag{8.3}
$$

The matrix Z in Equation (8.3) is scaled at the overlapping parts by the diagonal of K, as described in Equation (8.2). Nevertheless, it is clear that bodies Ω_1^b and Ω_2^b result in (almost) linearly dependent vectors in Z.

One strategy to deal with (almost) linearly dependent vectors in Z is to apply orthogonalization. Resulting vectors with a relative small norm are disregarded, other vectors can be normalized and are linear independent. The disadvantage of this approach is that the orthogonalization results in coupled vectors throughout several domains. An identified rigid body on the edge, located in two overlapping domains, results in an orthogonalized vector with values in previously non-overlapping parts in both domains. This approach couples the approximate null vectors, so that the implementation is less parallelizable.

Another strategy is to directly disregard (almost) linearly dependent vectors in Z. The quality of the vectors can be checked by orthogonalization. Resulting vectors with a relative small norm are disregarded. The original ('not-orthogonalized') vectors of good quality are preserved. Note that the quality check can be done by orthogonalizing the *nodes* of the bodies instead of Z, which is much cheaper (for example, reducing six degree-of-freedom-vectors to one node-vector results in a speed up of factor 18).

Both strategies should incorporate the correct scaling of the null space as described in Section 8.2. This means that after disregarding a body, the null space should be scaled with only the present bodies.

The current implementation is based on the second strategy to enforce parallelism. As an addition, it starts with ordering the bodies on size. The bodies are orthogonalized with respect to the previous bodies, starting with the largest bodies. This ensures that large

bodies are less likely to be disregarded.

8.3.2 Computing the condition number of the coarse matrix

The QR-decomposition determines Q and R such that $E = QR$, after which R^{-1} and Q^T are computed. Each thread i computes local Q_i^T and local R_i^{-1}, the partial columns of $Q^T$and of R^{-1}. Due to the explicit inversion of R it is cheap to compute the exact condition number $\kappa(E)$ as follows:

$$\begin{aligned}\kappa_*(E) &= \|E\|_*\|E^{-1}\|_* \\ &= \|QR\|_*\|R^{-1}Q^T\|_* \\ &= \|R\|_*\|R^{-1}\|_*,\end{aligned}$$

where typically $\|.\|_* = \{\|.\|_2, \|.\|_F\}$. The last statement holds because Q is orthonormal.

The Frobenius norm is used to compute the condition number for its computational attractiveness and the bound on the Euclidean condition number.

Theorem 9. *Let $K \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix. Then holds the following:*

$$\kappa_2(K) \leq \kappa_F(K).$$

Proof. Let K^{-1} denote the symmetric positive definite inverse of K. Let $\sigma_1 \geq \ldots \geq \sigma_r > 0$ be the singular values for K. Then

$$\|K\|_2^2 = \sigma_1^2 \leq \sum_{i=1}^{r} \sigma_i^2 = \|K\|_F^2.$$

Applying the same analysis for K^{-1} concludes the theorem. □

8.3.3 Accuracy of the coarse solution

The condition number $\kappa(E)$ strongly influences the accuracy of the solution x of $Ex = y$. The following inequality holds:

$$\frac{\|\Delta x\|}{\|x\|} \leq \kappa(E)\frac{\|\Delta y\|}{\|y\|}.$$

This inequality indicates the rate of change of the solution x with a change in the right hand side vector y. Assuming a machine precision of 10^{-16} and a convergence criterion of ϵ, the solution x of the coarse system can lose up to $\log(10^{16}\epsilon)$ significant digits without any problems. However, the solution of the coarse system may lose $\log(\kappa(E))$ significant digits. Consider the two techniques deflation and coarse grid correction.

1. Deflation. Since deflation is based on a projection technique, it is important not to lose any significant digits within the convergence criterion in the solution x of $Ex = y$. If a deflated vector has a small contribution in the non-deflated part, then this contribution will never leave the Krylov subspace again. Therefore, to make sure no significant digits within the convergence criterion are lost, it is required that $\kappa(E) < 10^{16}\epsilon$.

2. Coarse grid correction. Since coarse grid correction can be seen as a preconditioning technique based on multigrid, it is allowed to lose significant digits in the solution. Some numerical experiments showed that coarse grid correction can be effective even if the condition of the coarse matrix is extremely large (see e.g. Table 12 using 8 threads). Coarse grid correction will give a warning for large condition numbers but never aborts, except if the iterative solver fails to converge.

As a result, if the condition number $\kappa(E) \geq 10^{16}\epsilon$, then we switch from deflation to coarse grid correction. Most computations, such as the approximate null space matrix Z and the coarse matrix E, can be reused, but are just differently applied. The costs for switching to coarse grid correction are therefore relatively low; in practice this costs less than 0.1 seconds.

9 Results

This section describes artificial and real-life engineering cases and the performance of the iterative solution methods for these cases.

The methods are implemented in DIANA, which is programmed in Fortan77 and C. The supporting libraries are BLAS and LAPACK in the Intel MKL library. All cases are performed on the HENDRIKS machine, a Dell PowerEdge 2900 workstation. This workstation contains 2 Quad-Core Intel Xeon X5355 processors (8 CPUs) running at 2.66GHz and 24GB DDR2 ECC memory. Maximum parallelism is gained at 8 concurrent threads running in parallel. For more details please refer to Appendix A.2.

9.1 General applicability

Apart from the described cases in this section, the implementation of the rigid body modes approach has been extensively tested by performing the standard DIANA test suites. An amount of 828 linear, 139 geomechanical and 1605 nonlinear test problems were analyzed on the HENDRIKS machine. The tests validate correct behavior on a large scale of problems. The new implementation is tested for the original iterative solver settings, rigid body modes deflation and rigid body modes coarse grid correction.

9.2 Case descriptions

An overview of all cases is given in Table 3. The first five cases illustrate the strength and weakness of the iterative solvers and the last three cases are models of real-life applications.

9.2.1 3Cubes

3Cubes is a linear static model consisting of three high-stiffness cubes inside a low-stiffness cube. The model is built with tetrahedron elements and consists of four materials [Figure 11].

	# free d.o.f.	# materials	Analysis	Symmetry
3Cubes	15.735	4	Linear	Y
BeddingByInterfaces	206.681	2	Linear	Y
BeddingBySprings	203.401	2	Linear	Y
SplittedCube	161.711	2	Linear	Y
MixtureCube	33.832	2	Linear	N
Geo	73.336	9	Linear	Y
PitExcavation	10.856	8	Nonlinear	N
Road	2.286	5	Nonlinear	Y

Table 3: Overview of the cases.

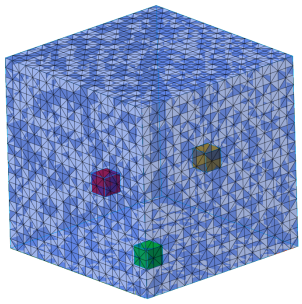


Figure 11: A graphical representation of the *3Cubes* case. Inside this cube there are three relatively stiff cubes located.

The three inner cubes are much stiffer than the outer cube as given in Table 4. The iterative solver is expected to perform poorly.

Material	Young's modulus
OuterCube	$1.0 \cdot 10^0\ kPa$
InnerCube1	$9.0 \cdot 10^5\ kPa$
InnerCube2	$6.0 \cdot 10^5\ kPa$
InnerCube3	$3.0 \cdot 10^5\ kPa$

Table 4: Young's moduli of the materials in the *3Cubes* case.

The model *3Cubes* has 17.058 degrees of freedom of which 15.735 degrees of freedom are really free (no constraints). The cube has fixed boundary conditions at the bottom. A uniform load is vertically applied on top of the large cube.

9.2.2 BeddingByInterfaces

BeddingByInterfaces is a linear static model of a cube standing on fixed interface elements, which function as a linear elastic bedding. The model is built with plane interfaces and hexahedron elements and consists of two materials [Figure 12]. The model *BeddingByInterfaces* has 211.806 degrees of freedom of which 206.681 degrees of freedom are really free (no constraints). The cube is attached to the interfaces at the bottom and supported at

two sides of the cube. The load is vertically applied at one of the top plane's corners, which physically will result in tilting the cube.

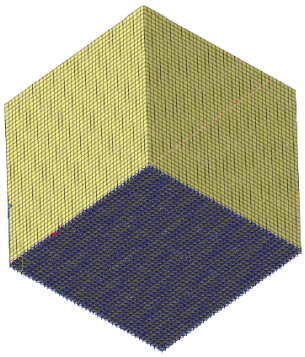


Figure 12: A graphical representation of the *BeddingByInterfaces* and *BeddingBySprings* cases. The blue bottom represents the linear elastic bedding.

9.2.3 BeddingBySprings

BeddingBySprings is similar to *BeddingByInterfaces*, but instead of interface elements, the linear elastic bedding is modeled with spring elements. The remainder of the model is built with hexahedron elements [Figure 12]. The model *BeddingBySprings* has 206.763 degrees of freedom of which 203.401 degrees of freedom are really free (no constraints). The cube is attached to the springs at the bottom and supported at two sides of the cube. The load is vertically applied at one of the top plane's corners, which physically will result in tilting the cube.

9.2.4 SplittedCube

SplittedCube is a linear static model consisting of quadratic hexahedron elements and quadratic plane interfaces. A uniform cube is divided into two parts by interface elements [Figure 13]. The interface elements are located between the green part and the yellow part. The stiffness of the interface elements is relatively low in comparison with the stiffness of

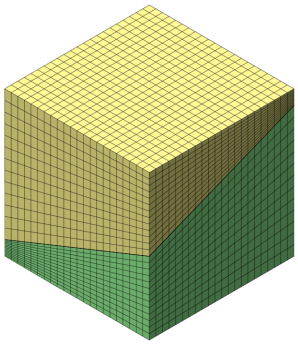


Figure 13: A graphical representation of the *SplittedCube* case. Interface elements are placed between the two parts.

the cube. The model *SplittedCube* has 170.706 degrees of freedom of which 161.711 degrees of freedom are really free (no constraints). The cube is supported in the normal direction at three neighboring planes (bottom, back and one side). A uniform load is vertically applied on top of the cube.

9.2.5 MixtureCube

MixtureCube is a non-symmetric linear static model. It consists of two materials modeled by classical and mixture hexahedron elements [Figure 14]. The classical elements form a horizontal three-element thick layer in the middle of the cube. Below and above the layer are the mixture elements. The model *MixtureCube* has 36.817 degrees of freedom of which 33.832 degrees of freedom are really free (no constraints). The cube is supported in the normal direction at two side planes and at all edges. A uniform load is vertically applied on top of the cube.

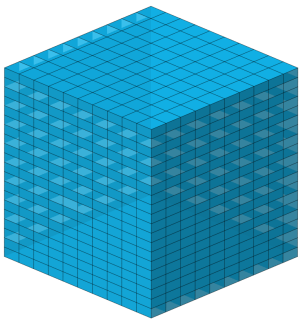


Figure 14: A graphical representation of the *MixtureCube* case. A three-thick element layer divides two mixture element blocks.

9.2.6 Geo

Geo is a linear static geotechnical model of a real-life application. The model is built with plane interfaces and hexahedron elements and consists of eight materials which roughly form layers in the model [Figure 15].

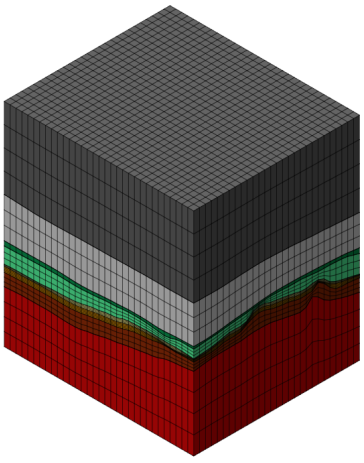


Figure 15: A graphical representation of the *Geo* case. A pressure load is applied in the center.

Young's moduli of the materials is presented in Table 5. This table gives the materials as they appear in the model from top to bottom.

Material	Young's modulus
Sediments	$1.0 \cdot 10^4$ kPa
Chalk	$5.0 \cdot 10^6$ kPa
Cromer Knoll	$1.0 \cdot 10^6$ kPa
Kimmeridge	$2.0 \cdot 10^6$ kPa
Interface	n/a
Upper Fulmar	$5.0 \cdot 10^5$ kPa
Lower Fulmar	$1.0 \cdot 10^6$ kPa
Pentland	$4.0 \cdot 10^6$ kPa
Triassic	$8.0 \cdot 10^6$ kPa

Table 5: Young's moduli of the materials in the *Geo* case.

The interface elements are located between the Kimmeridge and Upper Fulmar layer. The interface elements have $2.0 \cdot 10^3$ kN/m^3 normal stiffness and $1.0 \cdot 10^{-1}$ kN/m^3 shear stiffness moduli.
The model *Geo* has 77.517 degrees of freedom of which 73.336 degrees of freedom are really free (no constraints). The cube is supported in the normal direction at two side planes and at all edges of the model. A pressure load is applied in the center of the model; gravity is ignored. The model contains some 'ill-shaped elements', which means that the volume of the element is relatively low in comparison with the nodal distances.

9.2.7 PitExcavation

PitExcavation is a non-symmetric, nonlinear geotechnical model of a real-life application consisting of three phases: the initial state, wall installation and drainage, and excavation and loading. These phases result in five nonlinear steps. The case is modeled with plane interfaces and hexahedron elements and consists of eight materials [Figure 16].

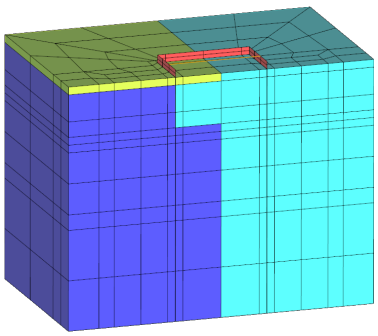


Figure 16: A graphical representation of the *PitExcavation* case. The part inside the red sheet pile wall will be excavated.

Young's moduli of the materials are presented in Table 6.

Material	Young's modulus
Sand	$2.0 \cdot 10^4\ kPa$
Sheet Pile Wall	$1.8 \cdot 10^{11}\ kPa$
Strut	$2.1 \cdot 10^8\ kPa$
Interfaces (5)	n/a

Table 6: Young's moduli of the materials in the *PitExcavation* case.

Five variants of interface elements occur in the model; between the sand and the wall and in the sand layers. The interface stiffness is of order $10^9\ kN/m^3$.
The model *PitExcavation* has a varying number of degrees of freedom, depending on the phase. In each phase one or more parts of the model is active. The number of degrees of freedom that is not constrained is between 10.856 and 11.808, depending on the phase. The cube is supported in the normal direction at three planes and at all edges of the model. There are three load cases. First, the gravity load is applied, then a load is applied on the surface and the last load is pore pressure on both sides of the wall after drainage.

9.2.8 Road

The case *Road* is a two-dimensional, nonlinear model of a real-life application. It consists of five layers, namely bitumen, a granular base, two sand layers and a soil foundation [Figure 17].

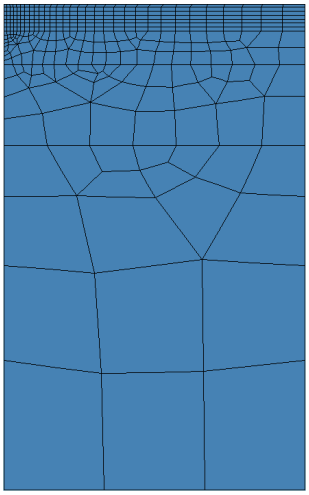


Figure 17: A graphical representation of the two-dimensional *Road* case.

Young's moduli of the top-to-bottom layers are presented in Table 7.

Material	Young's modulus
Bitumen	$4.0 \cdot 10^6\ kPa$
Granular	$8.0 \cdot 10^3\ kPa$
Sand 1	$6.5 \cdot 10^4\ kPa$
Sand 2	$1.3 \cdot 10^5\ kPa$
Soil	$6.0 \cdot 10^4\ kPa$

Table 7: Young's moduli of the materials in the *Road* case.

The two-dimensional model *Road* consists of 2.324 degrees of freedom of which 2.286 are really free (no constraint). The boundary conditions support the outside of the model in horizontal direction and the bottom in vertical direction.

A gravity load and a wheel load are applied. The wheel load is located at the upper left corner of Figure 17.

9.3 Numerical experiments

In this section the results of the cases are described. Table 8 presents the solvers that are compared.

PARDISO	
PCG	Preconditioned Conjugate Gradient
GMRES(s)	Restarted Generalized Minimal Residual method
DPCG	Deflated Preconditioned Conjugate Gradient
PPCG	Preconditioned Preconditioned Conjugate Gradient
DGMRES(s)	Restarted Deflated Generalized Minimal Residual method
PGMRES(s)	Restarted Preconditioned Generalized Minimal Residual method

Table 8: The solvers that are compared.

The PARDISO solver will be the standard solution method in DIANA version 9.6. It is a parallel direct sparse solver implemented by the Intel Math Kernel Library (Intel MKL). The solver uses a combination of left- and right-looking supernode techniques [36].

The PCG method respectively GMRES(s) method are the current implementations of the symmetric respectively non-symmetric iterative solver. The PCG and GMRES(s) methods use coarse grid correction that is *purely subdomain-based*, by the subdomains of the Schwarz domain decomposition method. The number of available threads is equal to the number of subdomains. The domain decomposition is not performed if it is specified to use one thread. In that case no coarse grid correction is applied.

The DPCG and DGMRES(s) methods are the newly implemented iterative solvers using rigid body modes deflation. The PPCG and PGMRES(s) methods are the newly implemented iterative solvers using rigid body modes in coarse grid correction. The rigid body modes are based on the *physical properties* of the model. The number of available threads is equal to the number of subdomains and each subdomain results in at least one and up to n_b rigid bodies. The rigid body modes are also used if it is specified to use one thread.

All iterative solution methods use the preconditioners IC(0) for symmetric matrices or ILU(0) for non-symmetric matrices, if not specified otherwise. Substructuring with or without rigid bodies proved not to be competitive and is therefore not included in the comparison.

Tables 9 – 16 show the performance of several solution methods in DIANA. The first column presents the number of available threads, which equals the number of subdomains. The

second column presents the condition number of the coarse matrix formed by the identified rigid bodies. The condition number is explicitly computed by

$$\kappa_F(E) = \|E\|_F \|E^{-1}\|_F = \|R\|_F \|R^{-1}\|_F.$$

The CPU times of the DPCG, PPCG, DGMRES(s) and PGMRES(s) methods *include* the rigid body identification time.

Figures 18 – 20 show the convergence of several iterative solvers. The vertical axis represents the relative residual and the horizontal axis represents the iteration number. The convergence criterion is 10^{-8}. The residual is always explicitly computed after convergence to verify the results.

9.3.1 3Cubes

Table 9 presents the results for the *3Cubes* case. The DPCG and PPCG solvers identified four global rigid bodies in 0.3 seconds. Note that the assembly of the bodies per subdomain always forms the whole subdomain. The bodies are formed by the three cubes and the remainder of the model, implying $Z \in \mathbb{R}^{n\times 24}$ for one subdomain. The number of removed bodies due to overlap increases with the number of threads. For example, four of the seventeen global bodies were removed using eight threads.

# threads	$\kappa_F(E)$	PCG		PARDISO		DPCG		PPCG	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$4 \cdot 10^3$	820	5.2	1	1.1	92	1.2	91	1.2
2	$6 \cdot 10^3$	761	3.1	1	0.8	92	0.9	100	0.8
4	$1 \cdot 10^4$	846	2.3	1	0.6	103	0.8	102	0.7
8	$4 \cdot 10^9$	746	3.6	1	0.6	63	1.0	67	0.9

Table 9: The results of the *3Cubes* case.

The PARDISO solver outperforms the iterative solvers due to the small size of the model. Figure 18 shows how the relative residual decreases with each iteration of the PCG, DPCG and PPCG methods for the *3Cubes* case.

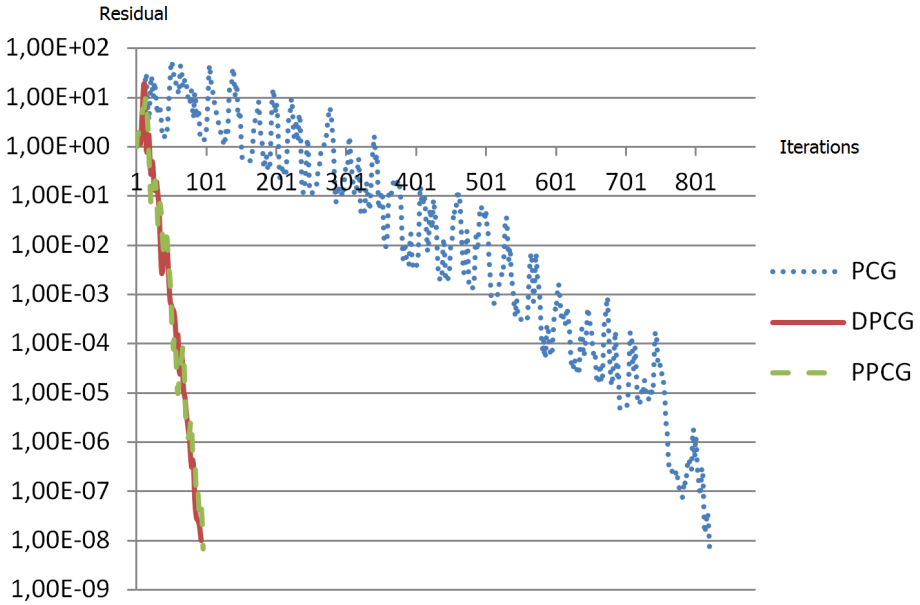


Figure 18: The convergence behavior of the *3Cubes* case using one thread.

This shows that convergence of the PCG method for the *3Cubes* case is slow and erratic. The DPCG and PPCG methods improve convergence by using the rigid body modes.

9.3.2 BeddingByInterfaces

Table 10 presents the results for the *BeddingByInterfaces* case. The DPCG and PPCG solvers identified two global rigid bodies in 0.8 seconds. These bodies are the interface elements (the bedding) and the solid elements, implying $Z \in \mathbb{R}^{n \times 12}$ for one subdomain.

# threads	$\kappa_F(E)$	PCG		PARDISO		DPCG		PPCG	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$3 \cdot 10^6$	130	25.8	1	132.3	88	21.0	104	23.8
2	$9 \cdot 10^6$	113	14.7	1	79.2	104	14.6	110	15.4
4	$2 \cdot 10^7$	123	12.8	1	44.7	113	13.4	125	13.9
8	$5 \cdot 10^7$	120	13.2	1	44.7	85	12.0	96	12.5

Table 10: The results of the *BeddingByInterfaces* case.

Figure 19 shows the convergence of the iterative solvers with sequential computations. Note that deflation and coarse grid correction improve the convergence of the PCG method. As shown, coarse grid correction results in a more erratic convergence process. Also note that the rigid bodies have a slightly negative effect on the convergence of PPCG for four threads, which is unexpected.

9.3.3 BeddingBySprings

Table 11 presents the results for the *BeddingBySprings* case. The DPCG and PPCG solvers identified one global rigid body in 0.8 seconds. The springs have a low stiffness, so that the nodes attached to the springs and the solid elements are put in the body of the solid

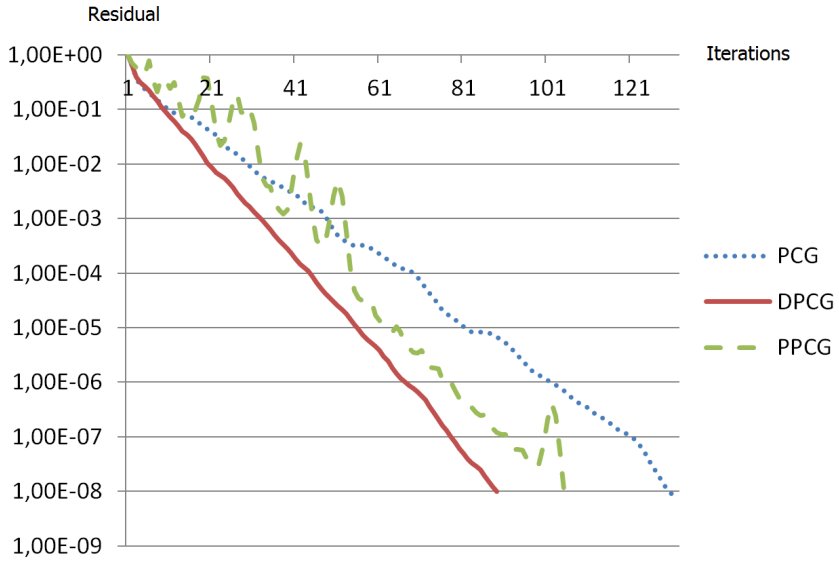


Figure 19: The convergence behavior for the *BeddingByInterfaces* case using one thread.

elements. This implies $Z \in \mathbb{R}^{n \times 6}$ for one subdomain.

# threads	$\kappa_F(E)$	PCG		PARDISO		DPCG		PPCG	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$1 \cdot 10^2$	143	26.7	1	128.9	90	21.4	95	22.9
2	$3 \cdot 10^2$	112	14.3	1	75.1	105	14.6	112	15.1
4	$8 \cdot 10^2$	135	13.5	1	43.6	127	13.9	135	14.4
8	$2 \cdot 10^3$	93	10.6	1	42.7	83	10.9	93	11.4

Table 11: The results of the *BeddingBySprings* case.

As can be seen in Table 11, the identification of rigid bodies in the model *BeddingBySprings* is not worthwhile. The result is one rigid body per subdomain, which is identical to the original PCG method. The iterations of the PCG and PPCG methods are identical when using more than one thread. The PCG method is faster than the PPCG method by saving the required time to identify one rigid body per subdomain. Note the difference between the PCG and PPCG methods in the sequential setting. The reason for this difference is that the PCG method does not use coarse grid correction in sequential computations.

9.3.4 SplittedCube

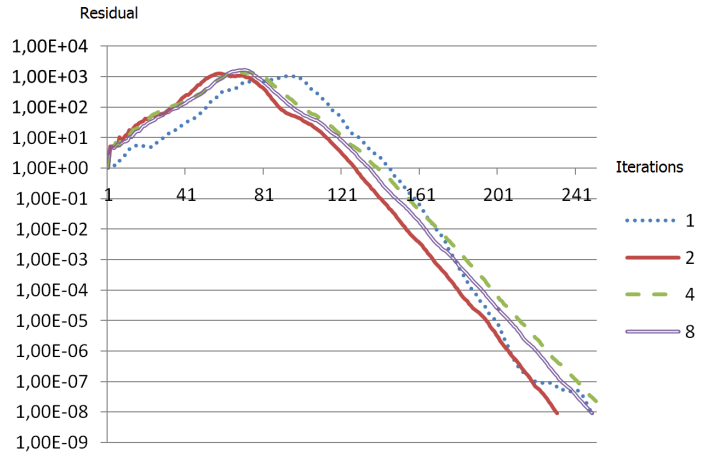
Table 12 presents the results for the *SplittedCube* case. The DPCG and PPCG solvers identified two global rigid bodies in 0.5 seconds. These bodies are the two cube parts. The interface elements have a low stiffness and the corresponding nodes are put in the bodies formed by the two cube parts. This implies $Z \in \mathbb{R}^{n \times 12}$ for one subdomain.

# threads	$\kappa_F(E)$	PCG		PARDISO		DPCG		PPCG	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$3 \cdot 10^8$	248	66.2	1	132.0	122	37.8	126	38.7
2	$6 \cdot 10^8$	229	43.1	1	78.0	n/a	n/a	144	30.7
4	$9 \cdot 10^8$	255	35.4	1	47.1	n/a	n/a	151	23.3
8	$3 \cdot 10^{19}$	248	37.5	1	47.9	n/a	n/a	128	23.4

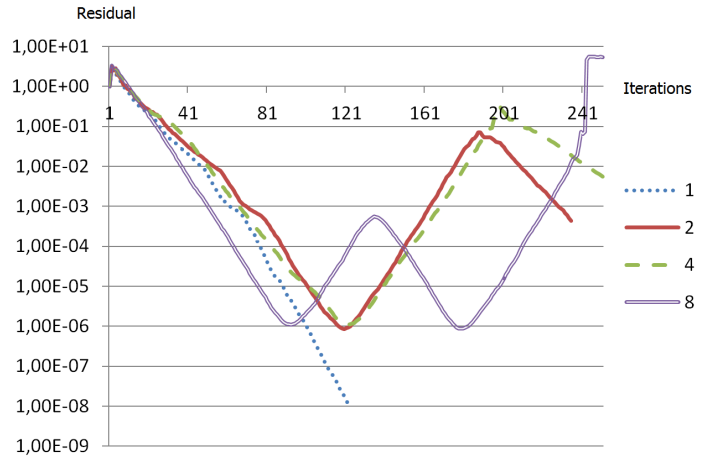
Table 12: The results of the *SplittedCube* case.

Note that deflation is too sensitive to rounding errors for 2, 4 or 8 threads, since the condition number $\kappa_F(E)$ is too large.

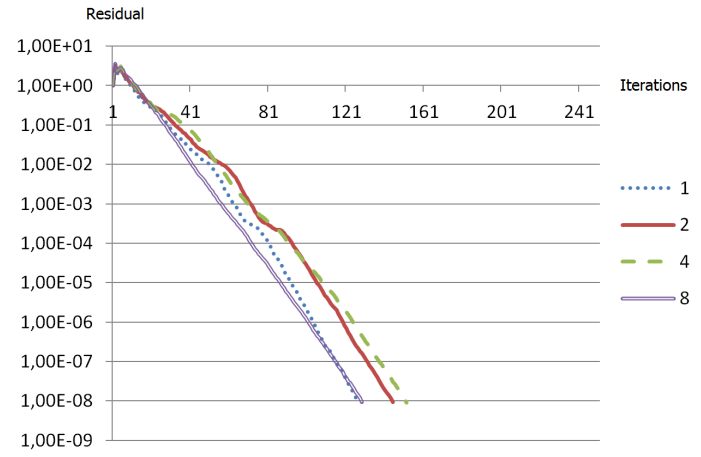
Figure 20 presents the convergence behavior of the PCG, DPCG and PPCG methods. The DPCG method does not always converge, as illustrated in Figure 20b. The convergence jumps for the PCG method in Figure 20a can be explained by the interface crack of the *SplittedCube* case. This jump is removed by correcting for the rigid body modes in Figure 20c.



(a) The PCG method: computations with one, two, four and eight threads.



(b) The DPCG method: computations with one, two, four and eight threads. Only the computation with one thread converges.



(c) The PPCG method: computations with one, two, four and eight threads.

Figure 20: The convergence behavior for the *SplittedCube* case.

9.3.5 MixtureCube

Table 13 presents the results for the *MixtureCube* case. The DGMRES(s) and PGMRES(s) solvers identified three global rigid bodies in 0.2 seconds. In two bodies extra pressure degrees of freedom occur. This implies $Z \in \mathbb{R}^{n\times(7+7+6)}$ for one subdomain. The number of removed bodies due to overlap increases with the number of threads. For example, two of the sixteen global bodies were removed using eight threads. Nevertheless, the coarse matrix is very ill-conditioned.

# threads	$\kappa_F(E)$	GMRES(s)		PARDISO		DGMRES(s)		PGMRES(s)	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$2 \cdot 10^{22}$	17	23.7	1	15.8	n/a	n/a	23	27.1
2	$2 \cdot 10^{11}$	30	210.4	1	10.3	29	221.6	30	223.7
4	$8 \cdot 10^{12}$	41	9.2	1	7.2	44	9.8	41	9.8
8	$2 \cdot 10^{11}$	42	8.8	1	7.0	44	9.5	44	10.1

Table 13: The results of the *MixtureCube* case.

Table 13 shows that the iterative solver with two threads is very inefficient. The additional time is required for the setup of the ILUT(10^{-6}) preconditioner. With other number of threads, the much cheaper ILUT(10^{-3}) preconditioner is used. Furthermore, identifying the rigid bodies does not seem not to speed up or even to slow down the convergence speed. This is caused by the ill-conditioned coarse matrix, which imply that deflation and coarse grid correction may be less effective. Using one thread and deflation results in divergence.

9.3.6 Geo

Table 14 presents the results for the *Geo* case. The DPCG and PPCG solvers identified two global rigid bodies in 0.3 seconds. These bodies are formed by the top layer Sediments and the remainder of the model. Note that the interface elements do not result in an additional rigid body. This implies $Z \in \mathbb{R}^{n\times 12}$ for one subdomain.

# threads	$\kappa_F(E)$	PCG		PARDISO		DPCG		PPCG	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$2 \cdot 10^{4}$	289	17.5	1	18.6	248	16.6	251	16.7
2	$7 \cdot 10^{4}$	255	10.3	1	11.1	255	10.9	258	10.8
4	$1 \cdot 10^{5}$	284	9.2	1	7.1	279	10.1	283	9.8
8	$4 \cdot 10^{5}$	256	9.2	1	6.5	251	10.6	255	9.9

Table 14: The results of the *Geo* case.

The convergence of all methods for the *Geo* case is acceptable. The residual decreases relatively slow, but gradually and stable. Note that the identified rigid bodies for two threads have a slightly negative effect on the convergence, which is unexpected.

9.3.7 PitExcavation

The nonlinear relation is solved by a Quasi-Newton method (see Section 2.6). This implies that the matrix does not change and hence, the decomposition can be reused in all nonlinear iterations per step. The first phase (initial state) requires two steps, involving the

solutions of nine linear systems; the second phase (wall installation and drainage) requires two steps, involving the solutions of thirteen linear systems; the third phase (excavation and loading) requires five steps, involving the solutions of thirty linear systems.

The iterative solvers GMRES(s), DGMRES(s) and PGMRES(s) use the ILUT(10^{-5}) preconditioner. The computation time of the *complete* analysis is given in Table 15. Only one thread is analyzed, since the GMRES(s) method failed to converge for a larger number of threads. The number of iterations and the condition number in Table 15 are averaged over all the solution procedures.

# threads	$\kappa_F(E)$	GMRES(s)		PARDISO		DGMRES(s)		PGMRES(s)	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$9 \cdot 10^2$	249,3	843	1	63	221,1	763	233,1	878

Table 15: The results of solving the linear systems of the *PitExcavation* case.

Note that PARDISO clearly outperforms the iterative solvers, since the decomposition can be reused every nonlinear iteration.

The DGMRES(s) and PGMRES(s) solvers identified two rigid bodies in each phase, implying $Z \in \mathbb{R}^{n \times 12}$. In the first phase the rigid bodies are the group of interface elements, where the wall will be placed and the remainder of the active part of the model. In the last two phases the rigid bodies are the wall and the remainder of the model. The stiffness matrix does not change within each phase. The DGMRES(s) and PGMRES(s) methods reuse the identified rigid bodies every nonlinear iteration in each phase.

9.3.8 Road

In the nonlinear analysis two gravity load steps and fifty wheel load steps are applied using Newton's method. The granular layer becomes stiffer as more load is applied, reducing the stiffness jump between the bitumen and the granular layer. The computation time of the *complete* analysis is presented in Table 16. Due to the small size of the model only one thread is analyzed. The number of iterations is averaged over all solution procedures that were able to solve the system using the IC(0) preconditioner. This means that fill-in solutions were neglected for the average number of iterations.

# threads	$\kappa_F(E)$	PCG		PARDISO		DPCG		PPCG	
		iter	CPU(s)	iter	CPU(s)	iter	CPU(s)	iter	CPU(s)
1	$1 \cdot 10^2$	292.2	17	1	8	239.7	18	248.5	17

Table 16: The results of the *Road* case.

The DPCG and PPCG solvers identified one rigid body in each linear system, implying $Z \in \mathbb{R}^{n \times 3}$, since the *Road* case is two-dimensional. The rigid body was reused in 75 of the 133 linear solves, since the element stiffness matrices changed not so extremely. Recomputing the new rigid body took 2 milliseconds on average.

The DPCG and PPCG methods are more effective if the stiffness difference is larger[5]. Note that PARDISO is significantly faster than any iterative solver for this small two-dimensional problem.

10 Conclusions

This section provides a summary of the theory, the conclusions from the numerical results and possible future research.

10.1 Summary of theory

The finite element method is introduced by the weak formulation of a PDE. DIANA uses a displacement application for structural problems including linear and nonlinear finite element analysis. Different types of properties are addressed as well as a variety of structural elements, such as continuum, spring, mixture and interface elements.

The resulting system of equation(s) can be solved by a number of iterative solution methods. A Krylov subspace method based on Arnoldi's procedure is the Full Orthogonalization Method (FOM), which orthogonalizes the residuals. A Krylov-based variant of the FOM is the Generalized Minimal Residual Method (GMRES), which minimizes the residual over the Krylov subspace. The symmetric version of Arnoldi's procedure is Lanzcos, resulting in the efficient short-recurrent Conjugate Gradient (CG) method.

Preconditioning a system can significantly improve convergence and robustness of an iterative solution method. Both left- and right-preconditioning are useful; left-preconditioning preserves the original iterations and right-preconditioning preserves the original residual norm. Different preconditioners are discussed such as: Jacobi, Incomplete Cholesky decomposition, Incomplete LU decomposition, Schwarz preconditioning and coarse grid correction.

DIANA provides the Schur and Schwarz domain decomposition methods. The Schur domain decomposition method, or substructuring, divides the model into parts based on their properties and computes the global solution sequentially. The Schwarz domain decomposition method divides the model into almost-equally large parts and can be used for parallel computations. Parallel computation in DIANA targets shared memory computers. All tasks execute the same sequential program (Single Program, Multiple Data). This way, the required data exchange between threads are kept to a minimum.

It is showed that the convergence of a Krylov subspace method can be improved by deflation. Thee deflated preconditioned iterations for the Conjugate Gradient and Generalized Minimal Residual Method are given. Some deflation strategies are eigenvector deflation, subdomain deflation and rigid body modes deflation.

The (approximate) rigid bodies of a model can be identified by using a coloring algorithm. The balance between the number and size of the stiffness jumps, and the number of rigid bodies determines the performance of the rigid body modes strategy. In a general finite

[5]If the stiffness difference between the bitumen and the granular layer is sufficiently increased (factor 10) or a linear analysis is performed, then the algorithm initially identifies two bodies. After applying gravity load, it identifies only one body.

element application, the coloring algorithm cannot be directly applied.

We propose to compute the average trace of the element stiffness matrices. Elements may be combined to rigid bodies based on their relative stiffness difference. Furthermore, we propose to reuse the identified rigid bodies in a nonlinear iteration loop if the average trace of the element stiffness matrices do not change significantly.

The modes corresponding to the identified rigid bodies can be used for deflation and coarse grid correction. Deflation is a more powerful technique than coarse grid correction, resulting in less iterations. On the other hand, coarse grid correction is better parallelizable and is more robust than deflation. An important factor in the robustness of deflation and coarse grid correction is the condition of the coarse (Galerkin) matrix.

We propose to remove small, substantially overlapping rigid bodies. Furthermore, we propose to compute the condition number of the coarse matrix and to efficiently switch from deflation to coarse grid correction for ill-conditioned coarse systems.

10.2 Conclusions from the results

This thesis addressed the following research question:

> *How can the iterative solution methods of* DIANA *be improved by incorporating the physical properties of the model?*

Eight cases are described in terms of dimensions, boundary conditions, external forces and performance. The first five cases illustrate the strengths and weaknesses of the iterative solvers and the last three cases are models of real-life applications. An analysis of the spectrum of the one-dimensional Poisson problem is a motivation to eliminate the rigid body modes corresponding to the stiff parts of the model.

The stiffness jumps are induced by approximate rigid bodies present in the underlying model, which result in slow convergence of the iterative solvers. The iterative solvers in DIANA are Conjugate Gradient (CG) and restarted GMRES (GMRES(s)). Identifying the rigid bodies in a finite element model only takes a fraction of the time of the complete analysis, often less than one second. The corresponding rigid body modes can be used by the iterative solvers for deflation or coarse grid correction.

In terms of iterations it is often beneficial to identify and use the rigid bodies in the model. In terms of CPU time it is beneficial for cases with stiffness jumps of 10^3 or larger. For these cases, the rigid body modes significantly improve the convergence of the iterative solvers, thus decreasing the computation time.

In the *3Cubes*, *MixtureCube*, *PitExcavation* and *Road* cases, the standard PARDISO solver clearly outperformed the iterative solvers. This is the result of small models, whereas the iterative solvers are more suitable for large models. In the *PitExcavation* case, the performance gain of PARDISO is enforced by reusing the matrix decomposition in the nonlinear iteration loop. Furthermore, the *MixtureCube* model with mixture elements is hard to solve by the iterative solver. The rigid body modes approach does not significantly improve convergence for models with mixture elements or other elements with scalar degrees of freedom

of another order of magnitude.

Rigid body modes deflation and rigid body modes coarse grid correction are compared in sequential and parallel computations. In general, deflation is more powerful than coarse grid correction in number of iterations. Deflation is also often more efficient in CPU time for a small number of threads. On the other hand, coarse grid correction is slightly cheaper per iteration, it scales better with the number of threads and it is more robust than deflation. In general, deflation outperforms coarse grid correction until four threads, provided that deflation can be applied sufficiently accurate. Deflation failed to converge for some cases with an ill-conditioned coarse matrix. The condition of the coarse matrix can be improved by removing some small overlapping rigid bodies, which can occur due to overlapping subdomains. In general, the number of removed bodies increases with the number of subdomains. If the condition number is still too large, the strategy is to switch from deflation to coarse grid correction. This takes about one-tenth of a second of computing time.

In nonlinear analysis, the identified rigid bodies of the previous linear system can sometimes be reused. In the *PitExcavation* case the rigid bodies were reused in the constant stiffness method in each phase of the analysis. In the *Road* case the rigid bodies were reused in 56% of the linear systems in Newton's method. Rigid bodies can be reused if the element stiffness matrices do not change significantly. A relative stiffness change of factor two seems an acceptable choice.

Rigid body modes deflation and rigid body modes coarse grid correction are implemented in CG and GMRES(s) in DIANA and may be available in DIANA 9.6. The techniques have been extensively tested by performing the standard DIANA test suites. An amount of 828 linear, 139 geomechanical and 1605 nonlinear test problems were analyzed on the HENDRIKS machine. The test were performed to validate correct behavior on a large scale of problems.

10.3 Future research

Rigid body modes deflation and rigid body modes coarse grid correction have been extensively tested. The recommendations for future research aim in particular at new concepts in the context of rigid bodies and at other problems with the iterative solvers.

- Approximate rigid bodies can be used in the partitioning. The corresponding rigid body modes can be specialized for certain problems.
  - The Schwarz domain decomposition method in DIANA can be improved by a partitioning that takes the rigid bodies into account. The idea is to identify the rigid bodies in the model and subsequently to decompose the domain into subdomains with the identified rigid bodies as a starting point [28].
  - Another point of interest is the approximate null space matrix Z corresponding to a rigid body. DIANA provides some elements with high stiffness in only a part of the element (e.g. in specified directions). The approximate null space of a collection of these elements can be improved by restricting the rigid body modes to the high stiffness parts. A question that arises is whether this approach is worthwhile.
- Nonlinear iteration schemes can require considerable computation time. Reusing information from previous solutions is a natural approach to speed up the solve pro-

cedure. The nonlinear iteration methods linear and constant stiffness can reuse the matrix decomposition of a direct solver. For slightly changing systems of equations, Newton-Krylov methods are efficient methods for solving sequences of linear systems.

  - Several improvements can be made in DIANA, such as automatically using the exact decomposition of the previous nonlinear iteration as a preconditioner in the current nonlinear iteration. Also, eigenvector deflation can effectively reuse information in nonlinear iterations loops [9, 14, 15, 43].
  - The identified rigid bodies are reused in a nonlinear iteration scheme if the element stiffness matrices do not change significantly. The criterion to reuse the rigid bodies can be further optimized.

- Several elements in DIANA can result in slow converging iterative solvers. Mixture and shell elements introduce different units within the element stiffness matrices, which induce jumps in the global stiffness matrix. Other scalar degree of freedoms can also have bad influence on the convergence.

  - The convergence of the iterative solvers is poor with e.g. mixture elements and shell elements. This could be resolved by defining or identifying each non-translational degree of freedom as an approximate rigid body, but in practice this is too cumbersome. An obvious approach would be to seek a proper (physics-based) preconditioner.
  - Mixture elements always yield non-symmetric systems, which may be difficult to solve and may require many iterations. An alternative for GMRES(s) is the short-recurrent IDR(s) algorithm. IDR(s) can outperform GMRES(s) in case of a large number of required iterations [38, 45]. One of the main challenges is to select the best performing non-symmetric method beforehand.

A DIANA

Table 17 converses some important parameters in this thesis to DIANA-implementation (where we like six-character capitalized parameters). These parameters are useful for DIANA developers.

	Thesis	DIANA
Maximum number of allowed bodies per subdomain	n_b	`MAXBOD`
Distinguish parameter for combining bodies	δ	`DISBOD`
Nonlinear tolerance paramater for reusing bodies	γ	`REUSEB`

Table 17: Conversion table for parameters.

A.1 The DIANA user

An average user of DIANA is likely to be unfamiliar with solution methods. Therefore, the solver is implemented as a 'black box' and requires but little knowledge of the user.

In the light of the implemented rigid body approach, the user can specify two input parameters, namely `TYPE` and `MAXBOD`. The input `TYPE` determines how to use the rigid bodies, thus by deflation or coarse grid correction (preconditioning). The default is `TYPE = AUTOMA`. The input `MAXBOD` determines the maximum number of allowed bodies per subdomain, which was previously denoted by n_b in Section 6. The default is `MAXBOD = 4`. The possible input values are described in Table 18.

Input	Value	Result
`TYPE`	`DEFLAT`	Rigid body modes deflation
	`PRECON`	Rigid body modes preconditioning
	`AUTOMA`	Automatic
	[Unspecified]	Automatic
`MAXBOD`	$n_b \in \mathbb{N}_{\geq 1}$	Maximum of n_b bodies
	[Unspecified]	Automatic

Table 18: Possible user-supplied input.

The following two input commands give the same results.

```
SOLVE ITERAT
```

```
SOLVE ITERAT RIGBOD TYPE AUTOMA MAXBOD = 4
```

Automatic rigid body type determines how to apply the rigid bodies by the number of specified threads. Rigid body modes deflation is applied if the number of threads is less than four; otherwise rigid body modes preconditioning is applied. The choice can still change during the solve procedure (from deflation to coarse grid correction) due to the condition number of the coarse matrix.

The command `RIGBOD OFF` explicitly switches off rigid body identification.

A.2 HENDRIKS machine

The details of DIANA's HENDRIKS machine are showed in Table 19.

Dell PowerEdge 2900
2 Quad-Core Intel Xeon X5355 2.66GHz/2x4MB 1333FSB
24GB 667MHz FBD (12x2GB dual rank DIMMs)
2 * 146GB SAS (15,000rpm) 3.5 inch Hard Drive (RAID1) System
2 * 300GB SAS (10,000rpm) 3.5 inch Hard Drive (RAID1) Data
2 * 450GB SAS (15,000rpm) 3.5 inch Hard Drive (RAID0) Scratch
PERC 5/i integrated RAID Controller Card, 256MB cache, battery backup
Red Hat Enterprise Linux Server release 5.10 (Tikanga)

Table 19: Detailed hardware specifications of the HENDRIKS machine.

B Performance and memory considerations

B.1 Approximate null space matrix

The approximate null space matrix Z is in general a sparse matrix, depending on the number of rigid bodies. Computations concerning Z can therefore be performed sparse. An advantage of storing Z sparsely is the memory requirements.

Let us illustrate this by a model Ω with four rigid bodies Ω^a, Ω^b, Ω^c and Ω^d. Suppose that the rank of each body is equal to six (three translations and three rotations). Equation (B.1) shows how the four rigid bodies can be described by six vectors with the same sparse structure; it contains only non-zeros in for those degrees of freedom that lay in that particular body. For this reason, it can be advantageous to exploit this sparse block structure. The corresponding sparse computations use one local-to-global mapping for six vectors at a time. This local-to-global mapping is therefore cheap.

$$Z = \begin{pmatrix} Z^a & & & \\ & Z^b & & \\ & & Z^c & \\ & & & Z^d \end{pmatrix}. \tag{B.1}$$

Full computations are merely faster for the special case with just one rigid body and only slightly faster in that case. The current implementation only supports sparse computations.

B.2 Pre-computing KZ versus not pre-computing KZ

Deflation requires to compute $y = KZx$ every iteration. The matrix KZ can be pre-computed (one-time-computation in advance of the iteration process) or be computed as $y = KZx = K\tilde{y}$, i.e. not pre-computed. A disadvantage of pre-computing KZ is some additional memory requirements, but this can be minor by converting K_Z to a sparse matrix.

The test problems in Table 20 compare pre-computing KZ and not pre-computing KZ within deflation with varying number of available threads. Only test problems with sufficiently accurate coarse matrix E (see Section 8.3.3) are included; other test problems automatically switched to coarse grid correction. The computations were performed using an executable that was created without optimization during compiling.

Case	# threads	CPU(s)	
		Pre-computed	Not pre-computed
Block	1	**0.83**	1.01
	2	**0.57**	0.65
	4	**0.47**	0.49
	6	0.35	**0.34**
	8	0.49	**0.46**
SphereInCube	1	**3.16**	3.83
3Cubes	1	**1.54**	1.87
	2	**0.98**	1.15
	4	**0.86**	0.89
Geo	1	**19.45**	24.04
	2	**12.64**	14.83
	4	**9.64**	10.88
	6	**10.47**	11.49
	8	**9.96**	10.58

Table 20: Computation time in seconds for preparing and solving the system of equations.

The best performing tests in Table 20 are colored green. Note that not pre-computing KZ can only be advantageous if the number of threads is large, but in these cases it is advantageous to use coarse grid correction. Therefore, deflation with pre-computing sparse KZ is applied.

C Stiffness range as generalization of the coloring algorithm

This approach defines the stiffness ranges in the model. A stiffness range in the model can be seen as a newly defined *material*. The elements are divided in a number of stiffness ranges and each element is contained in exactly one stiffness range. Thereafter, the classical coloring algorithm can be applied as in Algorithm 6. The steps are illustrated below.

1. Compute the trace of the element stiffness matrices and determine the maximum and minimum of all elements.

2. Determine the stiffness ranges. These stiffness ranges are based on relative differences: the upper bound of the current stiffness range is a constant factor times the upper bound of the previous stiffness range.

3. Put all elements into a stiffness range.

4. Check which stiffness ranges are filled with one or more elements and label these as active stiffness ranges.

5. Remove inactive stiffness ranges and combine stiffness ranges until a reasonable amount ($> n_b$) of stiffness ranges remain.

6. Apply the coloring algorithm to identify rigid bodies.

7. Combine identified bodies until n_b bodies (per subdomain) remain.

A disadvantage of this approach is its cumbersomeness. The identification of the stiffness ranges can be ineffective, since some neighboring elements with comparable stiffness can be split up into two stiffness ranges. The identification of stiffness ranges is merely based on stiffness and not on location; this can easily lead to a diffused distribution of the stiffness ranges, resulting in a large number of identified bodies.

D Induced dimension reduction

The recently proposed method IDR(s) [38] has proven to be highly efficient for some classes of non-symmetric systems. It is a short-recurrence Krylov subspace method, but, different from Bi-CG-type algorithms, it is not typically based on the bi-Lanzcos method. These Bi-CG-type methods (such as CGS and Bi-CGSTAB) are essentially based on biorthogonal bases $\mathcal{K}^m(K; r_0)$ and $\mathcal{K}^m(K^H; r_0) := \mathcal{K}^m(\bar{K}^T; r_0)$. The IDR method is based on forcing the residuals r_n in subspace $\mathcal{G}_j$ which is of decreasing dimension.

The original IDR method was published in Wesseling et al. [50]. Any IDR(s) method is based on this idea of IDR and its generalization is given in Sonneveld et al. [38].

Theorem 10. *Let K be any matrix in $\mathbb{C}^{N\times N}$, let v_0 be any nonzero vector in $\mathbb{C}^N$, and let $\mathcal{G}_0$ be the full Krylov space $\mathcal{K}^N(K, v_0)$. Let $\mathcal{S}$ denote any proper subspace of $\mathbb{C}^N$ such that $\mathcal{S}$ and $\mathcal{G}_0$ do not share a nontrivial invariant subspace of K, and define the sequence $\mathcal{G}_j$, $j = 1, 2, \ldots$ as*

$$\mathcal{G}_j = (I - \omega_j K)(\mathcal{G}_{j-1} \cap \mathcal{S}),$$

where the ω_j's are nonzero scalars. Then
(i) $\mathcal{G}_j \subset \mathcal{G}_{j-1} \quad \forall j > 0$,
(ii) $\mathcal{G}_j = \{0\}$ for some $j \leq N$.

For the proof please refer to Sonneveld et al. [38]. The IDR(s) method assumes the space $\mathcal{S}$ to be the left null space of some full rank $N \times s$ matrix $P = (\ p_1 \ \ p_2 \ \ \cdots \ \ p_s\ )$, shortly noted by $\mathcal{S} = \mathcal{N}(P^H)$.

The residuals r_n are in the Krylov subspaces $\mathcal{K}^n(K; r_0)$ and therefore, r_n can be written as $q_{n-1}(K)r_0$, where q_{n-1} is a certain polynomial of degree $n-1$. If we are able to find a recursion for r_n, then it should also be possible to find a recursion for u_n, since

$$K\Delta u_n = -\Delta r_n = (q_n(K) - q_{n+1}(K))r_0,$$

where the operator Δ is defined by $\Delta x_j := x_{j+1} - x_j$. Therefore, the general Krylov method can be described in the following form [38]:

$$
\begin{aligned}
r_{n+1} &= r_n - \alpha K v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \\
u_{n+1} &= u_n + \alpha v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta u_{n-l},
\end{aligned}
\tag{D.1}
$$

with $v_n \in \mathcal{K}^n(K; r_0) \setminus \mathcal{K}^{n-1}(K; r_0)$. The integer $\hat{l}$ is the depth of the recursion, e.g., using $\hat{l} = n$ is a long recurrence. If we force the residual r_{n+1} into $\mathcal{G}_{j+1}$ then

$$
r_{n+1} = (I - \omega_{j+1} K) v_n, \quad \text{with } v_n \in \mathcal{G}_j \cap \mathcal{S}. \tag{D.2}
$$

If we choose

$$
v_n = r_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{D.3}
$$

then we obtain the recursion of r_{n+1} in Equation (D.1) with $\alpha = \omega_{j+1}$.

Now suppose $r_n, \Delta r_{n-l} \in \mathcal{G}_j$, $l = 1, \ldots, \hat{l}$. This implies that $v_n \in \mathcal{G}_j$ by Equation (D.3). If we choose γ_l such that $v_n \in \mathcal{S}$ by Equation (D.2), then by Theorem 10 we have $r_{n+1} \in \mathcal{G}_{j+1}$. To satisfy this we need to find the correct γ_l. Taking $\hat{l} = s$ yields a unique solution for γ_l in solving the s-by-s linear system.

Defining the matrices

$$
\begin{aligned}
\Delta R_n &= (\ \Delta r_{n-1} \ \ \Delta r_{n-2} \ \ \cdots \ \ \Delta r_{n-s}\ ), \\
\Delta X_n &= (\ \Delta x_{n-1} \ \ \Delta x_{j-2} \ \ \cdots \ \ \Delta x_{n-s}\ ),
\end{aligned}
$$

then we can calculate $r_{n+1} \in \mathcal{G}_{j+1}$ as follows:

Algorithm 9. ***IDR update residual***

1 *Solve:* $c \in \mathbb{C}^s$ *from* $(P^H \Delta R_n) c = P^H r_n$
2 $v = r_n - \Delta R_n c$
3 $r_{n+1} = v - \omega_{j+1} K v$

The choice for ω_{j+1} is unspecified and is typically chosen to minimize the residual norm, provided that ω_{j+1} does not become very small (a threshold can be used).

A suitable IDR(s) algorithm for DIANA could be IDR(s)-biortho, described in Van Gijzen et al. [45].

References

[1] Auzinger, W., and Melenk, J. Iterative Solution of Large Linear Systems. *Vienna University of Technology* (2011).

[2] Bathe, K.-J. *Finite element procedures.* Prentice Hall, 1996.

[3] Brown, P., and Walker, H. GMRES On (Nearly) Singular Systems. *SIAM J. Matrix Anal. Appl. 18*, 1 (Jan. 1997), 37–51.

[4] Broyden, C. G. The convergence of a class of double-rank minimization algorithms: 2. the new algorithm. *IMA Journal of Applied Mathematics 6*, 3 (1970), 222–231.

[5] Butenhof, D. R. *Programming with POSIX threads*, third ed. Addison-Wesley Longman, Inc., Jan 1998.

[6] Crisfield, M. A. *Non-linear Finite Element Analysis of Solids and Structures*, vol. 1. John Wiley & Sons, 1991.

[7] Dagum, L., and Menon, R. OpenMP: an industry-standard API for shared-memory programming. *IEEE Computational Science & Engineering 5*, 1 (Jan 1998), 46–55.

[8] Dostál, Z. Conjugate gradient method with preconditioning by projector. *International Journal of Computer Mathematics 23*, 3 (1988), 315–323.

[9] Erhel, J., Burrage, K., and Pohl, B. Restarted GMRES preconditioned by deflation. *Journal of Computational and Applied Mathematics 69* (1995), 303–318.

[10] Foster, I. *Designing and building parallel programs.* Addison-Wesley, Inc., 1995.

[11] Francis, J. G. F. The QR Transformation A Unitary Analogue to the LR TransformationPart 1. *The Computer Journal 4*, 3 (1961), 265–271.

[12] Frank, J., and Vuik, C. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing 23* (2001), 442–462.

[13] Golub, G. H., and Loan, C. F. V. *Matrix Computations (3rd Ed.).* Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[14] Gosselet, P., and Rey, C. On a selective reuse of krylov subspaces in newton-krylov approaches for nonlinear elasticity. *Proceedings of the fourteenth international conference on domain decomposition methods* (2003), 419–426.

[15] Gosselet, P., Rey, C., and Pebrel, J. Total and selective reuse of krylov subspaces for the resolution of sequences of nonlinear structural problems. *CoRR abs/1301.7530* (2013).

[16] Haase, M. Lectures on Functional Analysis. Delft Institute of Applied Mathematics, 2012.

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

[18] Horn, R., and Johnson, C. *Matrix Analysis.* Cambridge University Press, Cambridge, UK, 1985.

[19] Hunter, J. K. Notes on Partial Differential Equations. Department of Mathematics, 2010.

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

[21] Jönsthövel, T., van Gijzen, M., Kasbergen, C., and Scarpas, A. Preconditioned conjugate gradient method enhanced by deflation of rigid body modes applied to composite materials. *Computer Modeling in Engineering and Sciences 47* (2009), 97–118.

[22] Jönsthövel, T., van Gijzen, M., MacLachlan, S., C.Vuik, and Scarpas, A. Comparison of the deflated preconditioned conjugate gradient method and algebraic multigrid for composite materials. *Computational Mechanics 50* (2012), 321–333.

[23] Jönsthövel, T., van Gijzen, M., Vuik, C., and Scarpas, A. On the use of rigid body modes in the deflated preconditioned conjugate gradient method. Report 11-04, Delft University of Technology, Delft Institute of Applied Mathematics, 2011.

[24] Kaasschieter, E. Preconditioned conjugate gradients for solving singular systems. *Journal of Computational and Applied Mathematics 24*, 12 (1988), 265 – 275.

[25] Kahl, K., and Rittich, H. Analysis of the deflated conjugate gradient method based on symmetric multigrid theory.

[26] Karypis, G., and Kumar, V. Metis. a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Tech. rep., University of Minnesota, September 1998.

[27] Lingen, F. *Design of an object oriented finite element package for parallel computers*. PhD thesis, Delft University of Technology, 2000.

[28] Lingen, F., Bonnier, P., Brinkgreve, R., van Gijzen, M., and Vuik, C. A parallel linear solver exploiting the physical properties of the underlying mechanical problem. Report 12-12, Delft University of Technology, Delft Institute of Applied Mathematics, 2012.

[29] Morgan, R. GMRES with Deflated Restarting. *SIAM J. Sci. Comput. 24*, 1 (January 2002), 20–37.

[30] Nabben, R., and Vuik, C. A comparison of Deflation and Coarse Grid Correction applied to porous media flow. *SIAM J. Numer. Anal. 42* (2004), 1631–1647.

[31] Nabben, R., and Vuik, C. Domain decomposition methods and deflated Krylov subspace iterations. In *European Conference on Computational Fluid Dynamics ECCOMAS CFD 2006* (2006), TU Delft.

[32] Nicolaides, R. A. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis 24*, 2 (Apr. 1987), 355–365.

[33] Notay, Y. An aggregation-based algebraic multigrid method. *ETNA 37* (2010), 123–146.

[34] Saad, Y. *Iterative Methods for Sparse Linear Systems*, 2nd ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[35] Saad, Y., and Schultz, M. GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM J. Sci. Stat. Comput. 7*, 3 (1986), 856–869.

[36] Schenk, O., Gärtner, K., and Fichtner, W. Efficient Sparse LU Factorization with Left-Right Looking Strategy on Shared Memory Multiprocessors. *BIT Numerical Mathematics 40*, 1 (2000), 158–176.

[37] Smith, B. Domain decomposition methods for partial differential equations. Tech. rep., 1990.

[38] Sonneveld, P., and van Gijzen, M. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM Journal on Scientific Computing 31*, 2 (2008), 1035–1062.

[39] Tang, J., Nabben, R., Vuik, C., and Erlangga, Y. Comparison of Two-Level Preconditioners Derived from Deflation, Domain Decomposition and Multigrid Methods . *Journal of Scientific Computing 39* (2009), 340–370.

[40] TNO DIANA. Diana *9.5 Finite Element Analysis, User's Manual, Analysis Procedures*, first ed., 2014.

[41] TNO DIANA. Diana *9.5 Finite Element Analysis, User's Manual, Element Library*, first ed., 2014.

[42] TNO DIANA. Diana *9.5 Finite Element Analysis, User's Manual, Material Library*, first ed., 2014.

[43] van der Linden, J. *Development of a deflation-based linear solver in reservoir simulation.* PhD thesis, Delft University of Technology, 2013.

[44] van der Vorst, H. Iterative Krylov methods for large linear systems. *Cambridge University Press* (2003).

[45] van Gijzen, M., and Sonneveld, P. Algorithm 913: An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties. *ACM Transactions on Mathematical Software 38*, 1 (November 2011), 5:1–5:19.

[46] van Kan, J., Segal, A., and Vermolen, F. *Numerical Methods in Scientific Computing*, 1st ed. VSSD, Delft, The Netherlands, 2005.

[47] Vermolen, F., Vuik, C., and Segal, A. Deflation in preconditioned conjugate gradient methods for finite element problems. In *Conjugate Gradient and Finite Element Methods*. Springer, 2004, pp. 103–129.

[48] Vuik, C., and Lahaye, D. Scientific computing (wi4201). Lecture notes for wi4201, 2012.

[49] Wells, G. N. The finite element method: An introduction. Lecture notes for CT5142, January 2011.

[50] Wesseling, P., and Sonneveld, P. Numerical experiments with a multiple grid and a preconditioned Lanczos type method. In *Approximation Methods for Navier-Stokes Problems*, R. Rautmann, Ed., vol. 771 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1980, pp. 543–562.

[51] Yeung, M., Tang, J., and Vuik, C. On the convergence of GMRES with invariant-subspace deflation. Reports of the Department of Applied Mathematical Analysis, 2010.

[52] Zienkiewicz, O. *The Finite Element Method*, 3rd ed. McGRAW-HILL Book Company (UK) Limited, Maidenhead, England, 1977.