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

Improving the iterative methods in TNO DIANA using physical properties of the underlying model.

Literature study report for the
Delft Institute of Applied Mathematics
as part of

the degree of

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

ALEX SANGERS

Delft, the Netherlands
December 2013

Literature study MSc report APPLIED MATHEMATICS

"Improving the iterative methods in TNO DIANA using physical properties of the underlying model."

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. H.X. Lin

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

December, 2013

Delft

Contents

1 Introduction

DIANA is an extensive multi-purpose finite element software package that is dedicated, but not exclusive, to a wide range of problems arising in Civil engineering including structural, geotechnical, tunneling, earthquake disciplines and oil & gas engineering. One of the computationally most intensive parts of a finite element analysis is the solution of one or more systems of linear equations, i.e., solving $Ku = f$. For this purpose a number of direct and iterative solution methods are available in DIANA.

Iterative solution methods are particularly attractive for large-scale three-dimensional (nonlinear) problems since they require less memory and, if properly working, are faster. A major drawback is that iterative methods 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 methods, such as preconditioning. Currently, the standard preconditioners used in DIANA are diagonal scaling, Incomplete LU decompositions and substructuring and in the context of domain decomposition are an additive Schwarz preconditioner and a coarse grid correction available.

The purpose of this research is to find out what problems are occuring with the iterative method of DIANA and how to solve these problems. One direction that will be considered is deflation, with the purpose to increase robustness and convergence speed of the iterative methods. Deflation is very suitable in combination with a preconditioner. The idea of deflation is to split the solution into two parts. The part that is deflated (projected out of the system) corresponds to the cause of slow convergence. The remaining part is converges relatively fast.

In Section 2 are some mathematical notation and definitions discussed. Section 3 introduces the Finite Element Method of DIANA. Section 4 describes the iterative solution methods, preconditioners and other techniques that are currently available at DIANA. The possible techniques to improve the iterative solver will be introduced in Section 5 and thereafter, the research question and test problems will be addressed in Section 6. Lastly, Section 7 illustrates how the deflation technique can be advantageous for the iterative solution methods.

2 Notation and definitions

Let us introduce some common mathematical notation and definitions.

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:*

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

Definition 2 *Let $x \in \mathbb{R}^n$ be a vector, $A \in \mathbb{R}^{n\times n}$ be a positive definite matrix and $p \in \mathbb{N}$.*

Then the following commonly used norms are defined by

$$||x||_1 = \sum_{i=1}^{n} |x_i|, \qquad ||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2},$$

$$||x||_\infty = \max_{i=1,\ldots,n} |x_i|, \qquad ||x||_A = \sqrt{x^T A x},$$

$$||A||_p = \max_{x \in \mathbb{R}^n \setminus \{0\}} \frac{||Ax||_p}{||x||_p}$$

The latter norm is called a matrix norm and for $p = 2$ (Euclidean norm) holds that $||A||_2 = \sqrt{\lambda_{\max}(A^T A)}$.

Definition 3 *Let $A \in \mathbb{R}^{n \times n}$ a matrix. The condition number κ of A is defined as*

$$\kappa(A) = ||A||_2 ||A^{-1}||_2.$$

If, furthermore, A is symmetric positive definite, this reduces to

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

Some norms are induced by an inner product as defined below.

Definition 4 *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*

$$\langle x, y \rangle_2 = \sum_{i=1}^{n} x_i y_i = x^T y,$$

$$\langle x, y \rangle_A = x^T A y.$$

Any norm induced by an inner product satisfies $||x||_* = \sqrt{\langle x, x \rangle_*}$. The 2-norm or Euclidean norm in Definition 2 is induced by the Euclidean inner product and for positive definite matrices A is the A-norm in Definition 2 induced by the A-inner product.

A nice property for matrices is to be self-adjoint.

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

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

Then the following function spaces are defined as

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

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.

3 The Finite Element Method

To illustrate how the linear system of equations $Ku = f$ is formed, consider the Poisson problem on V with $f \in \mathrm{L}^2(V)$:

$$\begin{cases} -\bigtriangledown^2 u = f, & \text{on } V, \\ \qquad u = 0, & \text{on } \Gamma = \partial V. \end{cases} \tag{3.1}$$

Often, partial differential equations (PDEs) arising from physics, such as (3.1), can be written as a minimization problem. Such problems typically minimize the underlying potential energy or seek the shortest path. The advantage of minimization problems is that they admit a larger solution class than a PDE formulation. Althought of historical relevance and importance and connection with physical meaning, the minimization problem formulation is only applicable in specific cases.

A more general approach is the weak formulation. Both the minimization problem and weak formulation lead to a formulation with 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 in the formulation of the minimization problem. As a rule of thumb, for second order PDEs all boundary conditions regarding to u are essential and all boundary conditions regarding to u' are natural. For more information on the minimization problem formulation, refer to [26].

3.1 The weak formulation

The weak formulation is another way (identical to the minimization problem if the minimization problem can be formulated) of admitting a larger solution class. The weak formulation uses the concept of weak derivatives. A weak derivative $g \in \mathrm{L}^2(V)$ of function $f \in \mathrm{L}^2(V)$ satisfies the following:

$$\int_V g(s)\lambda(s)\ ds = -\int_V f(s)\lambda'(s)\ ds, \quad \forall \lambda \in C_0^1(V). \tag{3.2}$$

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 on V, while the well-defined weak derivative 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 (almost-everywhere) and on $V\backslash\{0\}$ it is equal to the strong derivative of f. Furthermore, since the function space $C^1(V)$ is $||.||_{H^1}$-dense in the function space $H^1(V)$ [7], Equation (3.2) also holds for all $\lambda \in H_0^1(V)$.

Let us reconsider PDE (3.1), where the term $\bigtriangledown^2 u$ appears. This classical notation yields that u should be twice differentiable. The weak formulation admits a larger solution class, yielding $u \in H_0^2(V)$. Consider

$$-\bigtriangledown^2 u = f \tag{3.3}$$

$$\int_V -\bigtriangledown^2 u\ \lambda\ dV = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V) \tag{3.4}$$

$$\int_V \bigtriangledown u \cdot \bigtriangledown \lambda - \bigtriangledown \cdot (\bigtriangledown u\ \lambda)\ dV = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V) \tag{3.5}$$

$$\int_V \bigtriangledown u \cdot \bigtriangledown \lambda\ dV - \oint (\bigtriangledown u\ \lambda) \cdot \mathbf{n}\ d\Gamma = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V) \tag{3.6}$$

$$\int_V \bigtriangledown u \cdot \bigtriangledown \lambda\ dV = \int_V f\lambda\ dV, \quad \forall \lambda \in H_0^1(V). \tag{3.7}$$

Equality (3.4) 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 Equality (3.3) by the extension of DuBois-Reymond's lemma [26]. Equality (3.5) follows by Gauss divergence theorem and Equality (3.7) follows from the boundary conditions of λ.

This approach results in a lower order problem (in derivatives) with the same unique solution as the original differential equation. 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.

The Finite Element Method solves Equation (3.7) 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 choice of the test functions $\lambda_j \in H_0^1(0,1)$ strongly determines the sparsity of the resulting linear system of equations, which influences the required memory and CPU time. We would like to preserve the underlying model sufficiently accurate and usually the test functions λ_j satisfy $\lambda_j(x_i) = \delta_{ij}$, in order to ensure that $u(x_i) = u_i$ holds in the nodes. The number of nodes per element varies with the specific choice of test functions. Independent of the specific choice of λ_j, althought satisfying $\lambda_j(x_i) = \delta_{ij}$, leads the approximation u^n in our example in Equation (3.1) to

$$\begin{aligned} \int_V \triangledown u^n \cdot \triangledown \lambda_i \, dV &= \int_V f \lambda_i \, dV \\ \int_V \triangledown (\sum_{j=1}^{n} u_j \lambda_j) \cdot \triangledown \lambda_i \, dV &= \int_V f \lambda_i \, dV \\ \sum_{j=1}^{n} u_j \int_0^1 \triangledown \lambda_j \cdot \triangledown \lambda_i \, dV &= \int_V f \lambda_i \, dV. \end{aligned} \tag{3.8}$$

Using a total of n_{el} elements denoted by e_m, let us introduce

$$\begin{aligned} K_{ij} &= \int_V \triangledown \lambda_j \cdot \triangledown \lambda_i \, dV \\ &= \sum_{m=1}^{n_{el}} \int_{e_m} \triangledown \lambda_j \cdot \triangledown \lambda_i \, dV = \sum_{m=1}^{n_{el}} K_{ij}^{e_m}, \\ f_i &= \int_V f \lambda_i \, dV \\ &= \sum_{m=1}^{n_{el}} \int_{e_m} f \lambda_i \, dV = \sum_{m=1}^{n_{el}} f_i^{e_m}. \end{aligned} \tag{3.9}$$

This notation leads to

$$\sum_{j=1}^{n} K_{ij} u_j = f_i, \quad \forall i = \{1, \ldots, n\}, \tag{3.10}$$

$$K \underline{u} = f. \tag{3.11}$$

The solution $\underline{u}$ of the linear system (3.11) can be found by a direct or iterative solution method, as described in Section 4. The solution u of the original PDE in (3.1) is approximated by the solution $\underline{u} = \begin{pmatrix} u_1 & \ldots & u_n \end{pmatrix}^T$ found in (3.11) by $u \approx u^n = \sum_{j=1}^{n} u_j \lambda_j$.

3.2 Application of Finite Element Methods to structural problems

In a structural problem the displacements u, strains ε and stresses σ are often relevant parameters. The displacements are mostly directly calculated in the Finite Element Method, using the approach in previous section.

The strain is a measure of deformation, representing the displacements between particles relative to a reference length. Strain is therefore a dimensionless quantity. In general the strain is a matrix, but often the strain is expressed as a vector (engineering notation) for convenience [28]. The strain can be decomposed into normal and shear strain. The strain is expressed as a function of the displacements and often the strain is some function of the derivative of the displacements.

The stress is a measure of the internal forces per area (in case of compression comparable to pressure) that elements exert on each other. Stress is therefore of dimension force per area. Any strain generates a stress, as a reaction force on the deformation. Stress can also occur due to the external environment, for example when a solid vertical bar supports a hanging weight. Stress may exist even when strain is absent, or when no external forces occur (such as with so-called built-in stress). The stress is expressed as a function of the displacements and often the stress is some function of the second derivative of the displacements. In general the stress is a matrix, but often the stress is expressed as a vector (engineering notation) for convience [28]. The formulation of the displacement-based Finite Element Method is extensively described in Bathe [1].

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 stresses corresponding to ε are given by $\sigma = D_m(\varepsilon)$, with D_m the local stress-strain (elasticity) relation. Assuming linear elastic behaviour this can be written as $\sigma = D_m \varepsilon$ with D_m the rigidity matrix, often depending on Young's modulus E and Poisson's ration ν.

In element formulations the displacements u, strains ε and stresses 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 (3.4). The local matrices B_m and D_m depend on N_m and vary from element to element. The matrix T_m maps the local element numbering to the global numbering. The local stiffness matrices K_m and the global stiffness matrix K are formed by

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

where n_{el} is the number of elements. Note that in essence this is a specific (matrix) formulation of the general weak formulation approach.

3.3 Elements

Elements exists in a lot of variants. All elements consist of a number of nodes and corresponding degrees of freedom. Typically, the number of nodes per element is between one and forty. Each node consists of up to three translational degrees of freedom and up to three rotational degrees of freedom. Furthermore, also temperature or Lagrange multipliers can be degrees of freedom and in mixture elements pressure degrees of freedom also play a role. Geometrically speaking, DIANA offers nodal point elements, lines, triangles, quadrilaterals, pyramids, wedges and bricks.

3.3.1 Structural elements

Structural elements usually consist of up to three translational degrees of freedom per node. Rotational degrees of freedom are typical for special elements, such as shell elements. Standard type structural elements consists of a geometry and a material, the latter described by Young's modulus E and Poisson's ration ν [24]. Young's modules E indicates the material elasticity property. Poisson's ratio ν indicates the ratio of material deformation in the plane perpendicular to the direction of the exerting compression or stretching. The Young's modulus of a material can be used to calculate the force it exerts under specific strain. Assuming linear elasticity, the following three-dimensional relation can be given:

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{zx} \\ \sigma_{xy} \end{pmatrix} = C \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{yz} \\ \varepsilon_{zx} \\ \varepsilon_{xy} \end{pmatrix}, \tag{3.12}$$

where σ is the stress vector, ε is the strain vector and where the entries of the 'stiffness' matrix C depend on e.g. Poisson's ratio ν and Young's modulus E. Equation (3.12) is the three-dimensional generalization of Hooke's law for linear elastic material. In general, there are 36 stiffness matrix components. However, for all applications in DIANA the stiffness matrix is symmetric and depending on the application, more components may lose independence. This reduces the number of independent components to 21 (anisotropic), 9 (orthotropic), 5 (transverse isotropic) or 2 (isotropic) [30]. If we consider just the one-dimensional case, Hooke's law reduces to

$$\begin{aligned} \sigma &= E\epsilon, \\ F &= A\sigma = \frac{EA}{L}\Delta u, \end{aligned} \tag{3.13}$$

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. They 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 yield assumptions on the stress-strain relation, influencing the 'stiffness' matrix C.

A specific type of structural elements is shell elements [23]. The essence of shell elements is that the elements are planar (althought 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 shell-element node five (or six) degrees of freedom occur: three translational degrees of freedom and two (or, if drilling rotations are included, three) rotational degrees of freedom. A lot of principals of shells are described in Zienkiewicz [30].

3.3.2 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 or/and 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 [24]. Structural interface elements can have an initial stress (traction).

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 in general lead to slow convergence and should be avoided as much as possible by using structural interface elements if possible.

The fluid-structure interface elements are used in fluid-structure interaction analysis, coupling the fluids 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 the 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 stiffness D and depending on the application, the stiffness can be specified per direction and can depend on maturity, temperature, friction, etc. [24]. In any case the diagonal entries of D need to be specified. Assuming linear elasticity, the following three-dimensional relation is given:

$$\begin{pmatrix} \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} = \begin{pmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_3 \end{pmatrix} \begin{pmatrix} \Delta_x u \\ \Delta_y u \\ \Delta_z u \end{pmatrix}, \tag{3.14}$$

where τ is the traction vector (equivalent to the stress σ), D is the stiffness relation, Δu is the relative displacement. Equation (3.14) is called Hooke's law for linear elastic material. If we consider just the one-dimensional case, Hooke's law reduces to

$$\begin{aligned} \tau &= D\Delta u, \\ F = \tau A &= DA\Delta u, \end{aligned} \tag{3.15}$$

where F is the force and A is the cross-sectional area through which the force is applied. If we compare the one-dimensional stiffness of a classical structural element in (3.13) with an interface element in (3.15), then the one-dimensional relation is given by

$$\begin{aligned} F &= \left(\frac{EA}{L}\right)\Delta u = (DA)\Delta u, \\ \Rightarrow \frac{E}{L} &= D. \end{aligned} \tag{3.16}$$

3.3.3 Spring elements

Spring elements act as continuous damping in specific locations in the finite element model or model the interaction of the finite element model with its environment [24]. Spring element can consist of one or two nodes, can model translational or rotational springs and can be a spring and/or a dashpot. The spring constant k need to be specified. A spring often models one-dimensional elasticity and for linear static analysis the following relation holds (Hooke's law):

$$F = k\Delta u, \tag{3.17}$$

where F is the force, k is the spring stiffness and Δu is the relative displacement.

If we compare the spring stiffness relation with classical structural element stiffness relation, then

$$\begin{aligned} F = k\Delta u &= \left(\frac{EA}{L}\right)\Delta u, \\ \Rightarrow k &= \frac{EA}{L}. \end{aligned} \tag{3.18}$$

3.3.4 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 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. At fluid-structure interface elements the additional pore pressure potential degrees of freedom are only one-side added, extending only the first side to mixture.

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 [22], Section 60.2.

3.4 Element integration

The element integrals K_{ij} and f_i as in Equation (3.9) 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 [22] in the following way:

$$\int_{e_m} f \; dV = \sum_{i=1}^{n_\xi} w_{\xi_i} f(\xi_i), \tag{3.19}$$

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.

Sometimes it suffices to integrate only the low-order terms in the element integration. This is called the reduced integration scheme and is often sufficiently accurate for displacements. For stress and strain solutions it is better to use the original (full) integration scheme.

3.5 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. The solution of $F(u) = 0$ in the nonlinear case can be found be iteratively solving the linear(ized) systems as follows:

$$\begin{aligned} \tilde{K}(u^k)v^k &= \tilde{f}(u^k); \\ u^{k+1} &= u^k + v^k, \end{aligned} \tag{3.20}$$

where $v^k := u^{k+1} - u^k$. Forming and solving the linear system in (3.20) is the hard part. A number of iterative approximations are available in DIANA: Newton, Modified Newton, Quasi-Newton and linear and constant stiffness. Furthermore, continuation and line search are used to speed up these nonlinear iterative methods.

Newton's method solves $F(u) = 0$ using its Taylor expansion in the neighborhood of u^k by

$$\begin{aligned} F(u^{k+1}) &= F(u^k) + J(u^k)(u^{k+1} - u^k) + \mathcal{O}((u^{k+1} - u^k)^2), \\ &= F(u^k) + J(u^k)v^k, \end{aligned}$$

where $J = \frac{\partial F}{\partial u}$ and in the second step the second order terms are ignored. A better approximation $u^{k+1} = u^k + v^k$ can be constructed with the solution v^k of $J(u^k)v^k = -F(u^k)$. Note that u and F are vectors and J is the Jacobian matrix. Newton's method is effective, but the computation of the Jacobian J is very time consuming. Modified Newton, therefore, uses only the initial Jacobion matrix $J(u^0)$ so that each iteration is cheap. Of course, in general more iterations are needed with Modified Newton. Quasi-Newton methods, such as BFGS and Crisfield, 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 PDE satisfies the Maxwell equations of a magnetic field in an AC dynamo.

$$-\bigtriangledown \cdot (\nu_0 \nu_r \bigtriangledown u) = J, \tag{3.21}$$

where ν_0 is a constant and where

$$\nu_r = \alpha + (1-\alpha)\frac{||\bigtriangledown u||^8}{||\bigtriangledown u||^8 + \beta},$$

with $\alpha \in \mathbb{R}$, $\alpha \neq 1$ and $\beta \in \mathbb{R}$. If we are interested in the solution u that satisfies Equation (3.21), then we have to solve a nonlinear problem. In DIANA this means that we define one load step which is initialized at zero load and the zero solution and after the single step the load is increased with J, yielding that solution u of Equation (3.21) is computed by a nonlinear iteration method such as Newton's method. If the right-hand side vector is too large or if the model is strongly nonlinear, the nonlinear iterations could converge slow or not at all. This can be solved be applying several load steps to incrementally increase the right-hand side.

In many applications one is not only interested in solving one nonlinear PDE, but in several. The solutions can also affect subsequent solutions in next time/load steps, e.g., material elasticity can nonlinearly change after deformation. Suppose the solution vector u_m^k at time/load step k is converged after m nonlinear iterations. When apply 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 $k+1$.

4 Solution methods for linear systems available in DIANA

Consider the large linear system of equations $Ku = f$. Two classes of methods are available, namely direct and iterative methods. We will focus on the iterative methods due to its attractive properties for large three-dimensional problems. The iterative methods for solving $Ku = f$ are numerous. The first category of iterative methods are called Basic Iterative Methods (BIMs) and are based on a splitting $K = P - N$, followed by the iteration scheme

$$u_{m+1} = u_m + P^{-1} r_m, \tag{4.1}$$

with $r_m = f - Ku_m$ the residual. The matrix P should resemble K is 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 BIMs converge very slow in general ([27]).

One way to deal with this is to introduce a multigrid, which restricts the grid on a coarser grid, optionally multiple levels. This ensures faster convergence on the restricted matrix $\hat{K}$. Thereafter, the solution is interpolated back on the fine grid where another BIM iteration is performed.

The second category of methods are called Krylov subspace methods. These iterative solvers are often more effective than the BIMs and are currently used within DIANA.

4.1 Krylov subspace methods

A Krylov subspace is defined by $\mathcal{K}^m(K; r_0) = \text{span}\{r_0, Kr_0, \ldots, K^{m-1}r_0\}$, which is the m-order Krylov subspace generated by matrix K with starting vector r_0. Krylov subspace methods can be used to solve large systems of linear equations or to find eigenvalues, without performing matrix-matrix operations. A large amount of Krylov subspace based methods exist. In this report we mainly focus on (the derivation of) the well-known methods Conjugate Gradient (CG) and Generalized Minimal Residual (GMRES), since these methods are currently available in DIANA.

Consider $\mathcal{K}^m(K; r_0)$ with $r_0 = f - Ku_0$ the initial residual. We will use the notation $\mathcal{K}^m$ if there is no ambiguity. A Krylov subspace method uses the span of the vectors in subspace $\mathcal{K}^m$ to reduce the residual r_m.

The approximations u_m of u are based on a certain polynomial of degree $m-1$. In other words, $u_m = u_0 + q_{m-1}(K)f$, where q_{m-1} is a certain polynomial of degree $m-1$. The choice for the polynomial approximations strongly determines the success of the Krylov method.

Arnoldi's procedure is an algorithm for building an orthonormal basis of the Krylov subspace $\mathcal{K}^m(K; v_1)$ [19].

Algorithm 1 ***Arnoldi***

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

Every 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 $\bar{H}_m$ 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 a Hessenberg matrix. This is a matrix with only nonzero 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:

$$\begin{aligned} KV_m &= V_m H_m + w_m e_m^T && (4.2) \\ &= V_{m+1} \bar{H}_m && (4.3) \\ V_m^T K V_m &= H_m && (4.4) \end{aligned}$$

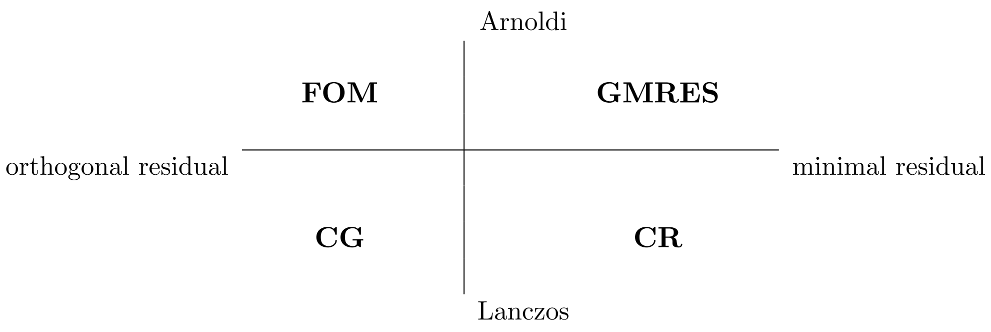
The equality of (4.3) can be seen by extracting from Algorithm 1

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

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

Rewriting in matrix formulations leads to (4.3). Equality (4.2) follows by step 4 in Algorithm 1, where w_j is orthogonal with respect to all previous vi. By premultiplying (4.2) with V_m^T and using orthonormality of its columns follows equality (4.4).

The following subsections describe different 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). FOM solves non-symmetric problems by orthogonalizing the residuals with respect to each other. The CG method applies the same strategy for symmetric problems. GMRES solves non-symmetric problems by minimizing the residual. 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. In addition to these four iterative methods also other methods are developed, but we will not discuss them in this report.



4.1.1 Full Orthogonalization Method

The Arnoldi procedure becomes particularly interesting if we take in Algorithm 1 the $v_1 = r_0/||r_0||_2 := r_0/\beta$. 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 is small. Remember we take $v_1 = r_0/\beta$ in Arnoldi's method. It follows from $KV_m = V_{m+1}\bar{H}_m$ that

$$\begin{aligned} r_m = f - Ku_m &= f - K(u_0 + V_m y_m) \\ &= r_0 - KV_m y_m \\ &= \beta v_1 - V_{m+1}\bar{H}_m y_m \\ &= V_{m+1}(\beta e_1 - \bar{H}_m y_m). \end{aligned} \tag{4.5}$$

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

$$\begin{aligned} y_m &= H_m^{-1}\beta e_1, \\ u_m &= u_0 + V_m y_m. \end{aligned} \tag{4.6}$$

To determine whether the solution u_m is sufficiently accurate, Equation (4.5) is reduced to

$$\begin{aligned} r_m = f - Ku_m &= \beta v_1 - V_{m+1}\bar{H}_m y_m \\ &= \beta v_1 - V_m H_m y_m - h_{m+1,m} e_m^T y_m v_{m+1} \\ &= -h_{m+1,m} e_m^T y_m v_{m+1}, \end{aligned} \tag{4.7}$$

by $H_m y_m = \beta e_1$. Taking the norm of Equation (4.7) 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,

$$\begin{aligned} & r_m = f - Ku_m = -(h_{m+1,m} e_m^T y_m) v_{m+1} \\ \Rightarrow\ & r_m \in \operatorname{span}\{v_{m+1}\} \\ \Rightarrow\ & r_m \perp \operatorname{span}\{v_1, \ldots, v_m\} \\ \Rightarrow\ & r_m \perp r_i, \quad \forall i \neq m. \end{aligned} \tag{4.8}$$

The FOM subsequently orthogonalizes all residuals and computes u_m by Equation (4.6).

4.1.2 Generalized Minimal Residual Method

DIANA uses restarted GMRES, which is based on the (full) GMRES algorithm. The full GMRES procedure is similar to the FOM and can be described by Algorithm 1. DIANA uses in the implementation of (restarted) GRMES a modified Gram-Schmidt procedure.

In the light of Equation (4.5), let us define the following operator

$$J(y_m) = ||f - Ku_m||_2 = ||f - K(u_0 + V_m y_m)||_2. \tag{4.9}$$

To solve the system $Ku = f$ it is clear that minimizing the Euclidean norm of the residual, $J(y)$, could be an advantageous strategy. Recall that at iteration m holds $f - Ku = V_{m+1}(\beta e_1 - \bar{H}_m y)$ from Equation (4.5). Taking the norm yields by orthonormality

$$J(y) = ||\beta e_1 - \bar{H}_m y||_2. \tag{4.10}$$

The GMRES method computes after sufficient convergence the solution u_m of the minimization problem (4.10) by

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

The restarted GMRES method computes this solution u_m after sufficient convergence or if the memory requirements of the Krylov vectors exceed a certain threshold. Restarted GMRES is bounded by e.g. m iterations and thereafter $u_0 := u_m$ is used to restart GMRES. At DIANA the GMRES is restarted if 50% of the memory used by the system matrix and the preconditioner is used for the Krylov vectors.

Full GMRES has optimal properties based on $\mathcal{K}^m(K; .)$, but has long recurrences. Each Krylov vector has to be stored and is used in each iteration, resulting in more CPU time and it can lead to memory issues.

The Hessenberg matrix H_m has eigenvalues and eigenvectors which approximate the eigenvalues respectively eigenvectors of matrix K. These eigenvalues and eigenvectors are called the Ritz values and Ritz vectors. Since H_m typically is much smaller than K, only the extreme eigenpairs are approximated. A QR algorithm could be used to determine the Ritz values and vectors. This information can be reused in the nonlinear loop, see Section 5.1.

4.1.3 The Conjugate Gradient Method

For symmetric positive definite (SPD) matrices the Conjugate Gradient (CG) method is a popular choice. SPD matrices K yield some nice properties, such as short-recurrence, optimality and orthogonal residual based on $\mathcal{K}^m(K;.)$. Looking at Arnoldi's procedure, note that by symmetry of K it follows from Equation (4.4) 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 nonzero 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. To adapt to common notation 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 [19].

Algorithm 2 ***Lanczos***

```
1  Choose a vector $v_1$, such that $||v_1||_2 = 1$
2  For $j = 1, 2, \dots, m$ 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 Modified Gram-Schmidt, 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 or minimizing the residual. 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 $T_m = L_m U_m$. The bandwidth of T_m is only two, resulting in

$$T_m = L_m U_m = \begin{pmatrix} 1 & & & \\ \lambda_2 & 1 & & \\ & \ddots & \ddots & \\ & & \lambda_m & 1 \end{pmatrix} \cdot \begin{pmatrix} \eta_1 & \beta_2 & & \\ & \eta_2 & \ddots & \\ & & \ddots & \beta_m \\ & & & \eta_m \end{pmatrix}. \tag{4.11}$$

Consider Equation (4.4), which can be reduced in the symmetric case to

$$\begin{aligned} V_m^T K V_m &= T_m = L_m U_m, \\ V_m^T K V_m U_m^{-1} &= L_m, \\ U_m^{-T} V_m^T V_m U_m^{-1} &= U_m^{-T} L_m. \end{aligned} \tag{4.12}$$

Define $P_m = V_m U_m^{-1}$, then Equation (4.12) reduces to

$$P_m^T V_m P_m = U_m^{-T} L_m. \tag{4.13}$$

This results in a symmetric and lower triangular matrix and therefore a diagonal matrix. The columns of P_m are called the search direction vectors or conjugate vectors p_j, $j = 1, \ldots, m$. From the resulting diagonal matrix in Equation (4.13) follows that the conjugate vectors p_j are K-orthogonal, i.e., $\langle p_i, Kp_j \rangle = 0$, $\forall i \neq j$.

The consequence of symmetry is that the residuals r_j and K-conjugate p_j can be constructed in a recurrence of two vectors, while for the nonsymmetric case the FOM requires m vectors to compute the next 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***

```
1  Compute $r_0 = f - Ku_0$, $p_0 = r_0$.
2  For $j = 0, 1, 2, \ldots,$until convergence, Do:
3    $\alpha_j = \langle r_j, r_j \rangle / \langle p_j, Kp_j \rangle$
4    $u_{j+1} = u_j + \alpha_j p_j$
5    $r_{j+1} = r_j - \alpha_j K p_j$
6    $\beta_j = \langle r_{j+1}, r_{j+1} \rangle / \langle r_j, r_j \rangle$
7    $p_{j+1} = r_{j+1} + \beta_j p_j$
8  EndDo
```

For a full derivation of the CG method please refer to [19]. In [19] is also shown that the coefficients in Algorithm 3 can be used to directly compute T_m as

$$T_m = \begin{pmatrix} \frac{1}{\alpha_0} & \frac{\sqrt{\beta_0}}{\alpha_0} & & & \\ \frac{\sqrt{\beta_0}}{\alpha_0} & \frac{1}{\alpha_1} + \frac{\beta_0}{\alpha_0} & \frac{\sqrt{\beta_1}}{\alpha_1} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} \\ & & & \frac{\sqrt{\beta_{m-2}}}{\alpha_{m-2}} & \frac{1}{\alpha_{m-1}} + \frac{\beta_{m-2}}{\alpha_{m-2}} \end{pmatrix}. \tag{4.14}$$

Corresponding eigenvalues and eigenvectors of T_m in Equation (4.14) are called Ritz values and Ritz vectors and they approximate the eigenvalues respectively eigenvectors of matrix K. A QR decomposition could be used to compute these eigenvalues and eigenvectors. In the nonlinear loop this information can be reused, see Section 5.1.

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 3 in Section 2. The following bound for the CG method is well-known.

Theorem 1 *Let K be a symmetric positive definite matrix. Then the error $u - u_m$ of the CG method at iteration m is bounded by*

$$||u - u_m||_K \leq 2 \left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right]^m ||u - u_0||_K. \tag{4.15}$$

This implies that small condition numbers $\kappa \geq 1$ result in fast convergence. The proof can be found in e.g. [19].

4.1.4 On other non-symmetric iterative methods

For general (non-symmetric) matrices K there exist several algorithms to solve $Ku = f$. Some popular choices are Bi-CGSTAB, IDR(s), GMRES and restarted GMRES, but a lot of other algorithms and variants exist. For non-symmetric matrices it is impossible to combine the advantageous properties optimality and short-recurrence of CG. The Bi-CGSTAB, IDR(s) and restarted GMRES algorithm are not optimal 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 Section 5.3 the IDR(s) method is described as an alternative for restarted GMRES. IDR(s) can be a valuable short-recurrence addition to the methods currently available in DIANA.

4.2 Preconditioning

For any Krylov subspace method it is important to have a good preconditioner to ensure fast and robust convergence. The idea is (in case of left preconditioning) that the preconditioned matrix $P^{-1}K$ has better convergence properties than the original K. A good preconditioner P should resemble K and it should be cheap to solve $Px = y$. Preconditioners are often inspired by the BIM matrix P (where $K = P - N$, see Equation (4.1)) or by direct methods. Preconditioning can be applied in different ways; from the left as in Equation (4.16), centrally as in Equation (4.17) and from the right as in Equation (4.18).

$$P^{-1}Ku = P^{-1}f, \tag{4.16}$$

$$P = LU; \qquad L^{-1}KU^{-1}x = L^{-1}f; \qquad u = U^{-1}x, \tag{4.17}$$

$$KP^{-1}x = f; \qquad u = P^{-1}x. \tag{4.18}$$

4.2.1 Preconditioned CG

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

Left and right preconditioning 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

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

This implies that using the P-inner product yields symmetry in case of left preconditioning.

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 using the P^{-1}-inner product yields symmetry in case of right preconditioning. 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 [19]. Moreover, the split preconditioning can also be written to the same algorithm, which implies that all preconditioning techniques yield the same solutions u_m.

4.2.2 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

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

Split 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 split preconditioning for GMRES, observe that the spectra of the three associated operators $P^{-1}K$, KP^{-1} and $L^{-1}KU^{-1}$ are identical. Still, in practice some difference in convergence behaviour can be seen. 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 in a finite space are equivalent, it still means that ill-conditioned systems can lead to different convergence behaviour due to numerical issues. DIANA applies right preconditioning for restarted GMRES.

4.2.3 Diagonal scaling

One of the simplest choice for P is diagonal scaling with diagonal elements $p_{ii} = k_{ii}$. The advantage of this preconditioner is that is very cheap to construct and very cheap to solve $Px = y$. The disadvantage is that this choice in general does not resemble K very accurately, resulting in only slightly less iterations.

4.2.4 Incomplete LU decomposition

Incomplete LU (ILU) decompositions exist in different variants, although all ILU decompositions are based on the same idea, namely that the LU decomposition is only partially done such that $K \approx P = LU$. This partial decomposition resembles K and can act as a preconditioner in a Krylov subspace method. In case of symmetric K, an incomplete Cholesky (IC) decomposition $K \approx P = LL^T$ is performed.

The standard preconditioner in DIANA is the ILU decomposition without fill-in, i.e., $l_{ij} = u_{ij} = 0$ if $k_{ij} = 0$. This makes sure that the sparsity pattern of K is unchanged, saving memory and CPU time.

If the ILU decomposition with no fill-in fails to convergence, then a threshold τ for fill-in is set up. This preconditioning is abbreviated by ILUT(τ). If needed, the threshold τ for fill-in can be decreased, resulting in a more accurate and expensive approximation of K. Note that we obtain the exact factorization if the drop tolerance is small enough, e.g. $\tau = 0$.

4.2.5 Other preconditioners

DIANA also offers a substructuring preconditioner. Furthermore, available in the context of domain decomposition, are an additive Schwarz and a coarsening preconditioner.

4.3 Domain decomposition

The purpose of domain decomposition is to divide the domain into a number of subdomains for parallel processing. The partitioning can be done in various ways, althought an efficient partitioner should have three objectives: minimize the number of so-called overlap degrees of freedom, minimize the variation in subdomain sizes and group together the degrees of freedom which have similar properties. The so-called overlap degrees of freedom can be loosely described as degrees of freedom that occur in multiple domains (and will be more precisely defined later this section). The first objective is to minimize communication between subdomains and improve parallelism. The second objective ensures optimal (balanced) parallel computation time. Lastly, the third objective is based on the observation that the preconditioner becomes more effective in such partitionings [13]. One can think of element types, degree of freedom types or material properties. This section we follow [13] closely, since the implementation in DIANA is very similar.

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. In DIANA, the partitioning is done by using Metis [12], a graph partitioning open software package. Metis partitions the degrees of freedom of the model. This implementation does make sure that the partitioning is balanced, based on the underlying connectivity of the elements and it also minimizes the overlap degrees of freedom. However, no other information, such as material properties, element types or stiffness is used to determine the partitioning.

Original degrees of freedom are called internal degrees of freedom, while the additional degrees of freedom are called overlap degrees of freedom. Each degree of freedom is an internal degree of freedom in exactly one subdomain, so an overlap degree of freedom in a subdomain is also an internal degree of freedom in exactly one other subdomain. Although the overlap degrees of freedom should be kept to a minimum, it might be advantageous to have some overlap.

By defining the restriction operators for each of the n_d subdomains, the matrix K can be expressed in terms of the subdomain matrices K_i (which might be overlapping) by

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

where L_i and R_i are the left and right restriction operators corresponding to the i-th subdomain. The left restriction operators L_i map the rows of the subdomain matrices to the global matrix and the right restriction operators R_i map the column of the subdomain matrices to the global matrix. The left restriction operator L_i correspond to the internal degrees of freedom of i-th subdomain, while the right restriction operator R_i corresponds to internal and overlapping degrees of freedom of the i-th subdomain. Note that non-zero columns of $(R_i - L_i)$ indicate the overlapping degrees of freedom of i-th subdomain.

The domain decomposition uses a two-level 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 number of subdomains.

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

$$P_1^{-1} = \sum_{i=1}^{n_d} R_i^T \hat{P}_i^{-1} R_i,$$

where $\hat{P}_i^{-1}$ is a subdomain preconditioner, such as ILU decomposition. If the additive Schwarz preconditioner fails, the more effective restricted additive Schwarz (RAS) preconditioner is being used, given by

$$P_1^{-1} = \sum_{i=1}^{n_d} L_i^T \hat{P}_i^{-1} R_i.$$

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

The second preconditioner is the coarse grid correction preconditioner P_2^{-1}. It is constructed in a similar way to classical multigrid, only the coarsening is extreme [20]. Applying the coarse grid correction preconditioner in an additive way yields $P^{-1} = I + P_2^{-1}$, or in the preconditioned case $P^{-1} = P_1^{-1} + P_2^{-1}$. Matrix P_2^{-1} is obtained by projecting the stiffness matrix K in the following way:

$$P_2^{-1} = Z(Z^T K Z)^{-1} Z^T = Z E^{-1} Z^T, \tag{4.19}$$

where Z is given by the rigid body modes of the n_d subdomains. In three dimensions each subdomain implies three translation and three rotation vectors, yielding the dimension of Z equal to $6n_d$. In Section 5.1.4 more information can be found about the rigid body modes.

The coarse restrictor operators $C_i \in \mathbb{R}^{6\times(6\cdot n_d)}$ restrict a global coarse vector x to a subdomain coarse vector x_i with the property

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

The computation of E^{-1} is most involved and is done 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 orthonomalization procedure to the columns of E. For details we refer to [13].

DIANA is able to combine the preconditioners in an additive or a multiplicative way. The additive way is simply the addition of the two preconditioners, i.e., the combined preconditioner $P^{-1} = P_1^{-1} + P_2^{-1}$, which would be implemented as follows:

$$\begin{aligned} y_1 &= P_1^{-1}x, \\ y_2 &= P_2^{-1}x, \\ y &= y_1 + y_2. \end{aligned}$$

A more effective strategy [13] is the multiplicative way, which computes

$$\begin{aligned} \tilde{y} &= P_1^{-1}x, \\ \tilde{r} &= x - K\tilde{y}, \\ y &= P_2^{-1}\tilde{r}. \end{aligned}$$

However, the multiplicative way needs additional communication between the threads and one extra matrix-vector multiplication.

The user of DIANA 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.

4.4 Substructuring

Substructuring can be seen as a specific type of domain decomposition method without overlapping subdomains. The idea is to treat a group of elements as a single substructure, for example, if many elements in a nonlinear model behave linearly, these elements can be put in a substructure. Substructuring in DIANA is implemented as a preconditioning technique.

In substructuring the elements are partitioned and thereafter, the degrees of freedom are divided in internal degrees of freedom and interface degrees of freedom. After reordering of n_s substructures the stiffness matrix can be written as

$$K \sim \begin{pmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_{n_s} & B_{n_s} \\ B_1^T & B_2^T & \dots & B_{n_s}^T & C \end{pmatrix}. \tag{4.20}$$

The A_i indices are the internal degrees of freedom, where the coupling between these substructures is done by the interface degrees of freedom B_i and B_i^T. The idea is that the reordered matrix K can easily be factorized as

$$\begin{pmatrix} A_1 & & & & \\ & A_2 & & & \\ & & \ddots & & \\ & & & A_{n_s} & \\ B_1^T & B_2^T & \dots & B_{n_s}^T & C^* \end{pmatrix} \begin{pmatrix} I & & & & A_1^{-1}B_1 \\ & I & & & A_2^{-1}B_2 \\ & & \ddots & & \vdots \\ & & & I & A_{n_s}^{-1}B_{n_s} \\ & & & & I \end{pmatrix}. \tag{4.21}$$

Here, the computation of the so-called Schur complement given by

$$C^* = C - \sum_{i=1}^{n_s} B_i^T A_i^{-1} B_i \tag{4.22}$$

is the most time consuming part. Assuming all A_i are SPD, we can compute $B_i^T A_i^{-1} B_i$ using a Cholesky factorization. Solving the system $C^* u = f$ can be the most time consuming part and the solution can be obtained by factorization of C^* or an iterative solution method. 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^*.

4.5 Domain decomposition vs. substructuring

The current implementation in DIANA provides domain decomposition and substructuring. These two preconditioning techniques are essentially different, but some differences at DIANA are an implementation choice. The table below shows the mayor differences of the implemented versions of domain decomposition and substructuring.

Substructuring	**Domain Decomposition**
Not parallel	Parallel
Partitioning of elements	Partitioning of degrees of freedom
Partitioning on low level	Partitioning on high level
Non-overlapping	Allowed both (non-) overlapping
Renumbering the degrees of freedom Solution for Schur complement	Additive Schwarz preconditioner Coarse grid correction
Low impact on implementation (implemented as preconditioner)	High impact on implementation (implemented as solution method)
Example: Linear vs. nonlinear splitting	Partitioning by Metis routine

5 Enhancing iterative methods

This section elaborates what possible solution techniques can be applied to improve the iterative methods.

5.1 Deflation

Deflation is a kind of preconditioning, which eliminates some small eigenvalues of K. These eigenvalues are projected out of the system of equations. Deflation has been developed by Nicolaides [17] and Dostál [2] and different deflation techniques have been improved and exploited by many authors [3,4,11]. Deflation has some analogies with multigrid methods, in the sense that deflation also uses two projections. To derive the deflation technique we shall seek these two projectors $\Pi^{\in}$ and $\Pi^{\perp}$. These projectors are based on the interpolation operator Z and restriction operator Y [4]. 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 $Y \in \mathbb{R}^{n\times m}$ be a basis for $\mathcal{Y}$ with $m \ll n$. The part of the solution u in $\mathcal{Z}$, $u^{\in}$, can be written as a linear combination of Z, implying $u^{\in} = Zy$. The residual $r^{\in} = f - Ku^{\in}$ is orthogonalized with respect to Y, i.e., $r^{\in} \perp Y$. This requirement [19] can be written as

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

By defining the Galerkin operator $E = Y^T KZ$ and rewriting above to $u^{\in}$ results in

$$\begin{aligned} u^{\in} = Zy &= ZE^{-1}Y^T f, \\ &= ZE^{-1}Y^T Ku. \end{aligned} \tag{5.2}$$

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

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

Note that for projector $\Pi^{\in}$ indeed holds $(\Pi^{\in})^2 = \Pi^{\in}$. Furthermore, the solution $u^{\in}$ can be calculated directly as in the first statement of Equation (5.2). 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

$$u = (I - \Pi^{\in})u + \Pi^{\in}u. \tag{5.3}$$

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

$$\begin{aligned} Ku^{\perp} &= K\Pi^{\in}u, \\ K\Pi^{\in}u &= K(I - ZE^{-1}Y^T K)u, \\ &= f - KZE^{-1}Y^T Ku, \\ &= (I - KZE^{-1}Y^T)f. \end{aligned} \tag{5.4}$$

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

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

Note that with indeed $(\Pi^{\perp})^2 = \Pi^{\perp}$. Using the identity $\Pi^{\perp}K = K\Pi^{\in}$, Equation (5.4) and introducing $\tilde{u}$ as the solution of the system (5.5) to avoid ambiguity, the following holds:

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

The solution $\tilde{u}$ of system (5.5) is the computational difficult part and can be solved iteratively. Deflation can in the light of Equation (5.5) be seen as a left preconditioner. The singularity of this system is not necessarily a problem, as long as the corresponding right-hand side is consistent (see [10] for CG). 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 $u^{\in}$ respectively $\tilde{u}$ of Equation (5.2) respectively (5.5) into (5.3) as

$$u = ZE^{-1}Y^Tf + \Pi^{\in}\tilde{u}.$$

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. Still, often the choice $Y = Z$ is made for certain properties. This results in a robust, non-singular Galerkin operator $E = Z^TKZ$. Furthermore, only one set of vectors need to be determined and stored. From this point on we take $Y = Z$ for the non-symmetric case as well and focus merely on the choice of 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 proper choice for Z is an important but not so obvious part of deflation. A choice for Z could be the span of the smallest eigenvectors of K. Other choices are subdomain deflation or rigid body modes deflation. These typical options are elaborated in Sections 5.1.2, 5.1.3 and 5.1.4.

5.1.1 Convergence of deflation

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

Deflation yields 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 results in a decrease in κ and thus increase in convergence speed. In [4] some theoretical bounds on the effective condition number of $\Pi^{\perp}K$ are given for SPD matrices K.

Theorem 2 *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 there exists a splitting $K = C + R$ such that C and R are symmetric positive semidefinite with $\mathcal{N}(C) = span\{Z\}$ the null space of C. Then the effective condition number of $\Pi^{\perp}K$ is bounded by*

$$\kappa_{\mathit{eff}}(\Pi^{\perp}K) \le \frac{\lambda_{\max}(K)}{\lambda_{m+1}(C)}. \tag{5.6}$$

The proof is given in [4]. Furthermore, in combination with a splitting Cholesky preconditioner $P = LL^T$ the following bound can be given:

Theorem 3 *Assume the conditions of Theorem 2 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*

$$\kappa_{\mathit{eff}}(L^{-1}\Pi^{\perp}KL^{-T}) \le \frac{\lambda_{\max}(L^{-1}KL^{-T})}{\lambda_{m+1}(L^{-1}CL^{-T})}. \tag{5.7}$$

The proof is given in [4]. These bounds yield that deflation can only improve the effective condition number of the (preconditioned) system of equations $Ku = f$.

5.1.2 Eigenvalue deflation

A typical choice for Z is the span of the eigenvectors corresponding to the smallest eigenvalues. Such an approach certainly is effective, but there are also some disadvantages.

Firstly, a priori, the smallest eigenvectors are unknown. In some iterative methods such as GMRES(s) and CG, the (approximate) eigenvectors can be computed relatively cheap. These vectors can be used as a deflation space in the restart of GMRES(s), see e.g. [3,14].

Secondly, a large system of equations yields a large amount of eigenvectors. Deflating a large amount of small eigenvectors means that the dimension of Z grows beyond its effectiveness.

Nevertheless, eigenvalue deflation has been shown to be effective. In CG and GMRES the Ritz vectors can be used in deflation. In [3] and [14] the restart of GMRES is augmented or deflated using approximated eigenvectors and in [5] and [6] is the CG method augmented in nonlinear structural analysis problems. In DIANA these two types of eigenvalue deflation can be an effective technique to speed up the convergence process.

5.1.3 Subdomain deflation

Let us divide the domain into d subdomains G_j, $j = \{1, \ldots, n_d\}$. Then we choose

$$Z_{ij} = \begin{cases} 1 & \text{if } i \in G_j, \\ 0 & \text{otherwise,} \end{cases} \tag{5.8}$$

resulting in $Z \in \mathbb{R}^{n\times n_d}$. If these domains have similar properties, then the convergence of an iteration method could speed up. This approach is very well suitable with domain decomposition, yielding parallel computations. DIANA uses a coarse preconditioner in domain decomposition, which is analogue to an extension of this deflation technique by the rigid body modes.

5.1.4 Rigid body modes deflation

The idea of rigid body modes deflation is to treat a collection of elements as a rigid body due to physical properties, such as material. The stiffness of a collection of elements can (relatively) be very large and therefore it acts as one rigid body. These elements 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.

Let us consider an arbitrary three-dimensional problem. Assume we split the stiffness matrix $K = C + R$ with C containing n_r independent singular submatrices corresponding to some very stiff parts and one positive definite submatrix corresponding to the other material. Using rigid body modes deflation, the subspace Z is equal to the span of the null space of C, i.e., $Z = \mathcal{N}(C) = \text{span}\{Z_a^{n_r}\}$ with $Z_a^{n_r} = \{z_{n_{ra}}^1, \dots, z_{n_{ra}}^6\}$ the n_r times six base vectors corresponding to the six rigid body modes of the aggregate. This results in $Z \in \mathbb{R}^{n\times(6\cdot n_r)}$.

The splitting $K = C + R$ provides us the possibility to decouple the matrix K into disjoint matrices C_i with $C = \bigcup_i C_i$ and mutual couplings R_i, with $R = \bigcup_i R_i$. If we choose matrices C_i on basis of material properties, then the matrices C_i do not have irregular jumps in its coefficients.

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

Recall Theorem 2 and 3 for the bounds on the effective condition number of the deflated system $\Pi^\perp K$ and the preconditioned deflated system $L^{-1}\Pi^\perp K L^{-T}$. The splitting of $K = C + R$ is here explicitly given.

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. Equation (5.9) shows the rigid body modes of a node at (x, y, z). Each column gives the translation in x-, y- and z-direction, respectively rotation in x-, y- and z-direction. Each row represents a degree of freedom, with the x-, y- and z-translation respectively the x-, y- and z-rotation degrees of freedom.

$$\begin{array}{l} \text{x-translation dof} \\ \text{y-translation dof} \\ \text{z-translation dof} \\ \text{x-rotation dof} \\ \text{y-rotation dof} \\ \text{z-rotation dof} \end{array} \begin{pmatrix} 1 & 0 & 0 & 0 & -z & y \\ 0 & 1 & 0 & z & 0 & -x \\ 0 & 0 & 1 & -y & x & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.9}$$

Note that often the rotational degrees of freedom (rows four till six) are absent, depending on the type of element. Furthermore, 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 $\{(1,0,0),(0,1,0),(0,0,1)\}$).

Let p be the number of nodes in a element, then the rigid body modes are spanned by a combination of the above vectors for nodes, increasing their length from 6 to $6p$ (actually, the vectors are padded with zeros to length n). Sets of elements make up the bodies of materials. The rigid body modes of a collection of elements is equal to the assembly of the rigid body modes of the individual elements. Therefore, each body (collection of elements) imply 6 deflation vectors in 3D, as each body has three translational and rotational degrees of freedom [8].

5.1.5 Extension of rigid body modes deflation

Interface elements can model various phenomena, such as elastic bedding, cracking, bond-slip along reinforcements, friction between interfaces, joints in rocks, contact and fluid-structure relations. Some of these phenomena can be seen as modeling *freedom* and *restrictions* of parts of the model. For example, interface elements used to model friction or contact allows translations and rotations, or joints in masonry allow translation in shear direction. Rigid body mode deflation may be extendable for the following situations where interface elements model some behavior:

- (Structural) Elastic or nonlinear-elastic bedding
- (Structural) Discrete cracking
- (Structural) Bond-slip between reinforcements
- (Structural) Friction between interfaces
- (Structural) Joints in rocks and masonry
- (Contact) Contact between two different bodies

Identifying rigid bodies in a model with interface and spring elements is not trivial at all. In Section 3 in Equation (3.16) and (3.18) is the stiffness relation given for the different element types. These comparisons involve element sizes and the cross-sectional area through which the force is applied. Furthermore, the parameters in interface element can depend on time, previous stresses, temperature, pressure, etc. A great challenge of this research will be to effectively identify the (approximate) rigid bodies in a model. Thereafter, rigid body mode deflation can speed up the iterative solution method.

Information can also be reused in the nonlinear loop or maybe even in a dynamic analysis. Certain properties in the model may be unchanged during some nonlinear iteration, implying that deflation vectors also may be unchanged. Therefore, the initialization time of deflation could easily pay off in a nonlinear loop.

5.1.6 Comparison of deflation and additive coarse grid correction

With the knowledge of the coarse grid correction of Section 4.3, we recognize that the Galerkin matrix $E = Z^TKZ$ is represented in the same way if $Y = Z$ is the span of the rigid body modes. If the coarse grid correction is used in an additive way, then it even holds that the deflation projector $\Pi^{\perp} = I - P_2^{-1}$, with P_2^{-1} the coarse grid corrector.

The coarse grid correction treats each subdomain as a rigid body, i.e., each subdomain is represented by a six-dimensional subspace. Clearly, this is in general an extreme coarsening. Intuitively, the coarse grid correction provides a direct solution for the simplification that each subdomain is a rigid body by solving $P_2 x = y$. This results in a convergence rate almost independent of the number of subdomains.

With rigid body deflation, one or more subdomains, which act as a rigid body due to e.g. material properties, are projected out of the system of equations. This deflated part is computed directly and the non-deflated part is solved by using a traditional iterative solver, such as CG or GMRES(s). The non-deflated system is $\Pi^{\perp} K u = \Pi^{\perp} f$.

In both techniques the Galerkin matrix $E = Z^T K Z$ acts as a representation of the subdomains using their rigid body modes. In [15] and [16] a comparison is made between coarse grid correction and deflated preconditioning. It can be proven that, with arbitrary full rank matrix Z, the effective conditioner number for deflation is always below the condition number of the system preconditioned by the coarse grid correction. On the other hand, deflation is slightly more expensive.

In [8] the rigid body modes deflation vectors and the subdomain deflation vectors are combined. If n_r rigid bodies are being distinguished and n_d subdomains are formed, then the total dimension of Z is $(n \times (6n_r + n_d))$. The Galerkin operator used in the coarse preconditioner in domain decomposition may be extended with the rigid body modes based on material properties. In other words, the rigid body modes could also be implemented as a coarse grid correction instead of deflation, and vice versa.

5.2 Scaling the degrees of freedom

In some elements different types of variables are present, such as in mixture elements (pressure and translation) or in shell elements (translation and rotation). The magnitude of these different types of variables can differ greatly due to the corresponding unit. This large difference in magnitude can lead to an ill-conditioned stiffness matrix with large jump in its coefficients. This yields slow convergence in general.

The type of degree of freedom for every degree of freedom is known before the solver starts. This knowledge can be used to scale the degrees of freedom corresponding to their type. This can be done by (right) preconditioning as in Equation (4.18). If P_s^{-1} only consists of (diagonal) scaling, then it is symmetric and applicable for CG and GMRES.

Applying multiple preconditioning techniques implies that, after scaling with P_s^{-1}, the second preconditioner P^{-1} preconditions the system $K P_s^{-1} x = f$, instead of being applying to $Ku = f$.

5.3 Induced dimension reduction

Much research has been done on solving the nonsymmetric system $Ku = f$. The recently proposed method IDR(s) [21] has proven to be highly efficient for some classes of applications. 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 [29]. Any IDR(s) method is based on this idea of IDR and its generalization is given in [21].

Theorem 4 *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 $j \le N$.

For the proof please refer to [21]. The IDR(s) method assumes the space $\mathcal{S}$ to be the left null space of some full rank $N \times s$ matrix $P = \begin{pmatrix} p_1 & p_2 & \cdots & p_s \end{pmatrix}$, 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 [21]:

$$\begin{aligned} r_{n+1} &= r_n - \alpha K v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \\ u_{n+1} &= u_n + \alpha v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta u_{n-l}, \end{aligned} \tag{5.10}$$

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

$$r_{n+1} = (I - \omega_{j+1}K)v_n, \quad \text{with } v_n \in \mathcal{G}_j \cap \mathcal{S}. \tag{5.11}$$

If we choose

$$v_n = r_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{5.12}$$

then we obtain the recursion of r_{n+1} in Equation (5.10) 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 (5.12). If we choose γ_l such that $v_n \in \mathcal{S}$ by Equation (5.11), then by Theorem 4 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 4 ***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 [25]. On some important problem classes it outperforms other short-recurrence iterative methods, such as Bi-CGSTAB.

6 Research question and test problems

As the demand for larger and more accurate finite element analysis grows every year, so do the corresponding models. These large three-dimensional problems can lead to millions of degrees of freedom and thus, to an equally large system of equations. Iterative methods have proved to be able to solve these systems in a reasonable time and require less memory than the direct methods.

6.1 Research question

In Table 1 the current shortcoming of the iterative solvers are listed. There is a division in requirements and wishes, based on priorities. The research question is as follows:

How can the iterative methods in DIANA be improved using physical properties of the underlying model?

6.2 Test problems

The current problems listed in Table 1 are demonstrated by some test problems. The performance of the iterative solvers is in some cases unsatisfactory. These test problems serve as a basis of eliminating the limitations of the currently implemented iterative solvers. An overview of the test problems is given in Table 2.

1. (Model1*: Multiple materials, linear static analysis.*)
   Model1 is a linear geotechnical model from the field consisting of eight materials which are more or less layers in the model. The model is shown in Figure 1. *Model1* has 74.646 degrees of freedom of which 70.465 degrees of freedom are really free (no boundary condition). The boundary conditions are positioned at the edges and at two

Requirements
The iterative solver must be improved on models with jumps in material properties
The iterative solver must be improved on models with interface elements
The iterative solver must be improved on models with mixture elements
The iterative solver must be improved by reusing information in the nonlinear loop
Wishes
The iterative solver could be accelerated
The iterative solver could be improved on models with shell elements
The iterative solver could be extended by IDR(s)
The iterative solver could be using GPU or a similar computation strategy
(The iterative solver could be improved on models with jumps in element sizes)

Table 1: Requirements and wishes.

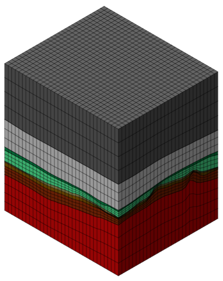


Figure 1: A graphical representation of test problems *Model1* and *Model2*.

sides of the model, supporting all three translation directions. The load is focused at a point in the middle layer of the model. The model contains some 'ill-shaped elements', meaning that the volume of the element is relatively low compared with the nodal distances. The solution converges in 199 CG iterations using the Incomplete Cholesky (IC) preconditioner.

2. (Model2*: Multiple materials, nonlinear static analysis.*)
   Model2 is similar to *Model1*, but with a nonlinear analysis. A number of load steps can be applied. The nonlinear relation is solved by the constant stiffness method (see Section 3.5). The solution is found in one nonlinear iteration using the CG method, which converged in 198 iteration using the IC preconditioner. The nonlinearity of this model can (should) be further increased to increase the amount of required nonlinear iterations.

3. (Model3*: Mixture elements, nonlinear static analysis.*)
   Model3 is a nonlinear geotechnical model of a block consisting of solely mixture elements of one material. *Model3* has 38.324 degrees of freedom of which 34.998 degrees of freedom are really free (no boundary condition). The boundary conditions are positioned at the edges and at two sides of the model, supporting all three translation

directions. At one side the pressure is kept constant. The load is positioned uniformly at the top plane. A number of load steps can be applied. The nonlinear relation is solved by the constant stiffness method (see Section 3.5). The solution is found in one nonlinear iteration using the GMRES method, which converged in 359 iteration using the ILUT($\tau = 10^{-6}$) preconditioner. The nonlinearity of this model can (should) be further increased to increase the amount of required nonlinear iterations.

4. (Model4*: Mixture elements and multiple materials, nonlinear static analysis.*)
   Model4 is a nonlinear geotechnical model of a block consisting of mixture elements and two materials. The stiffer material is layered between two layers of the elastic material. *Model4* has 36.817 degrees of freedom of which 33.832 degrees of freedom are really free (no boundary condition). The boundary conditions are positioned at the edges and at two sides of the model, supporting all three translation directions. At one side the pressure is kept constant. The load is positioned uniformly at the top plane. A number of load steps can be applied. The nonlinear relation is solved by the constant stiffness method (see Section 3.5). The solution is found in one nonlinear iteration using the GMRES method, which converged in 396 iteration using the ILUT($\tau = 10^{-6}$) preconditioner. The nonlinearity of this model can (should) be further increased to increase the amount of required nonlinear iterations.

5. (Model5*: Linear elastic bedding modeled by interface elements, linear static analysis.*)
   Model5 is a linear model of a block standing on fixed interface elements, which function as a linear elastic bedding. *Model5* has 211.806 degrees of freedom of which 206.681 degrees of freedom are really free (no boundary condition). The boundary conditions are positioned at the edges and at two sides of the model, supporting all three translation directions. The load is focused at the top plane corner of where the support is located, which physically will result in tilting the block. The CG method converges in 130 iterations using the IC preconditioner.

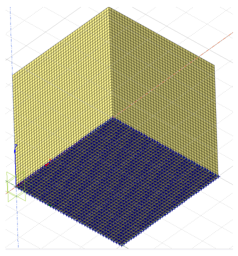


Figure 2: A graphical representation of test problems *Model5* and *Model6*. The blue bottom represents the linear elastic bedding.

6. (Model6*: Linear elastic bedding modeled by spring elements, linear static analysis.*)
   Model6 is similar to *Model5*, but instead of interface elements, the linear elastic bedding is modeled with spring elements. *Model6* has 206.763 degrees of freedom of which 203, 401 degrees of freedom are really free (no boundary condition). The boundary conditions are positioned at the edges and at two sides of the model, supporting all three translation directions. The load is also focused at the top plane corner of where the support is located. The CG method converges in 143 iterations using the IC preconditioner.

7. (Model7*: Interface elements, linear static analysis.*)
   Model7 is a linear model consisting of one material and a linear static analysis. The

	# free dof	Element type	# materials	Analysis	Symmetry
Model1	70.465	Brick	8	Linear	Y
Model2	70.465	Brick	8	Nonlinear	Y
Model3	34.998	Mixture brick	1	Nonlinear	N
Model4	33.832	Mixture brick	2	Nonlinear	N
Model5	206.681	Interface plane & Brick	1	Linear	Y
Model6	203.401	Springs & Brick	1	Linear	Y
Model7	21.764	Interface plane & Pyramid	1	Linear	Y
Model8	21.764	Interface plane & Pyramid	2	Linear	Y

Table 2: Overview of test problems.

interface elements 'split' the block into two parts as illustrated in Figure 3. The interface elements are located at the edge of the blue and yellow part. *Model7* has 24.333 degrees of freedom of which 21.764 degrees of freedom are really free (no boundary condition). The boundary conditions are positioned at the edges, at all outer surfaces of the lower part of the block and at two sides of the block. The boundary conditions support several directions. The load is positioned uniformly at the top plane. The stiffness of the interface elements is relatively low compared to

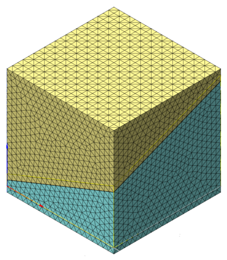


Figure 3: A graphical representation of test problems *Model7* and *Model8*.

the stiffness of the material. The CG method requires 118 iterations to converge.

8. (Model8*: Interface elements and multiple materials, linear static analysis.*)
   Model8 is similar to *Model7*, but with two materials. The CG method converges in 94 iterations using the IC preconditioner.

6.3 Methodology

Section 5 described what various methods could improve the iterative solver. An important technique is *deflation*, described in Section 5.1. In particular deflation based on rigid body modes is interesting, which is for example applicable on material properties [8]. In this research rigid body mode deflation will be implemented at DIANA. Moreover, other possible applications of rigid body mode deflation will be explored involving interface elements, spring elements and maybe shell elements. Furthermore, in the nonlinear iteration loop information we could use deflation to *reuse information* from previous iterations by using Ritz vectors or other previous deflation vectors. Section 5.2 explained how the different types of degrees of freedom that are present in mixture and shell elements, can be *scaled* by preconditioning. Section 5.3 described the *IDR(s) algorithm* for nonsymmetric matrices. In the light of the research question, this algorithm does not use the underlying model. Nevertheless, IDR(s) could be a valuable additional method in the DIANA software. IDR(s) allows reusing information from previous iterations, potentially exploitable by deflation, and furthermore, a nonsymmetric short-recurrence solution method is currently absent at DIANA.

6.4 Work plan

Firstly, rigid body mode deflation will be implemented in DIANA. Rigid body mode deflation can be extended by recursive deflation, i.e., repeated deflation. This can additionally be done in the nonlinear loop, so that the initialization time of deflation is less dominant in the nonlinear analysis. Thereafter, deflation could be combined with domain decomposition (with more than one domain).

Secondly, currently the mixture elements cause a poorly scaled system of linear equations. Using a (right) diagonal preconditioner with entries based on the type of degree of freedom, much improvement could be possible. This preconditioner should also allow multiple preconditioners, such as ILU decomposition.

Thirdly, more ideas for using deflation will be investigated. One idea is to use information of the previous iterations at a restart of GMRES, or to gradually improve the deflation vectors based on eigenvalues as soon as this information becomes available by the CG or GMRES(s) method. Another idea is to use rigid body mode deflation in models with interface elements, or possibly with shell elements. A possibility could be to deflate large (dummy) stiffness imposed by interface elements and shell elements. Another possible improvement it is to deflate some user-imposed 'nearly rigid bodies', e.g., by linear bedding or friction modeled by interface elements. All these ideas need to be further investigated.

Thereafter, if time allows, the short-recurrence iterative method IDR(s) could be implemented in DIANA. The IDR(s) method for nonsymmetric matrices has recently been developed [25]. It requires less memory, but does not have the optimality property of (full) GMRES. Restarting GMRES yields slower convergence and in such cases, especially when the system of equations is large, the IDR(s) method can outperform restarted GMRES in memory requirements, amount of iterations and required time.

The work plan is illustrated in a Gantt chart in Figure 7 in Appendix B on page 45.

7 Illustrative results

This section illustrates the potential advantage of the deflation technique based on the rigid body modes as described in Section 5.1.4. Test problem *Block5* is shown in Figure 4 and consists of 33.425 degrees of freedom and two materials. The inner material is very stiff (factor 10^6) compared to the surrounding material. The block is supported at the ground and at the top edges of the block and a load is put uniformly at the top plane of the block. The inner sphere with high stiffness acts, by approximation, as a rigid body in the low stiffness cube. Therefore, applying rigid body mode deflation based on material properties on the sphere could lead to faster convergence, as the corresponding eigenvalues are close to zero. The approximate computation time for PCG and DPCG reduces 13 seconds using 410 iterations to 6 seconds using 115 iterations.

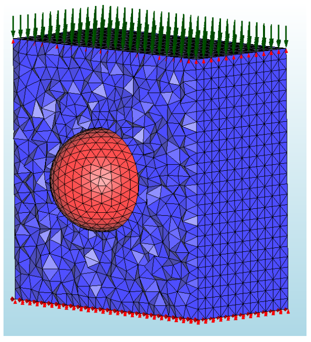


Figure 4: Test problem *Block5* with high stiffness material (red sphere) in low stiffness material (blue cube).

When applying current implementation on test problem *Block5*, we get a promising result shown in Figure 5.

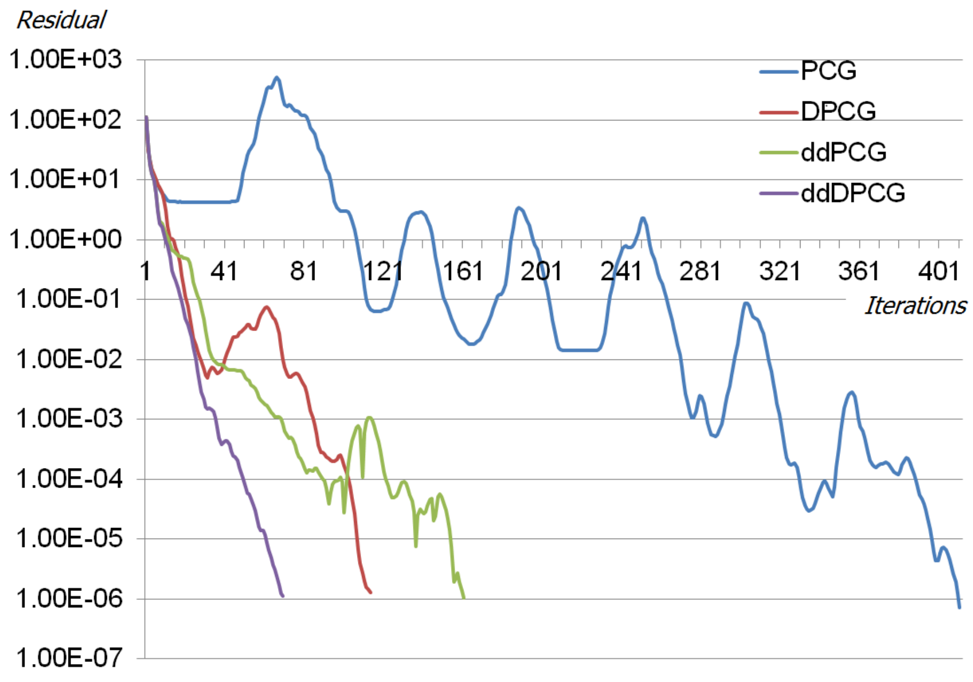


Figure 5: Illustrative result of rigid body mode deflation in comparison with other different solution methods for test problem *Block5*.

In Figure 5 is abbreviated ddDPCG stands for domain decompositioned Deflated Preconditioned Conjugate Gradient and all other abbreviations can be derived. All methods use IC without fill-in. The domain decompositions consists of one domain, so essentially the only difference is the added coarse grid correction. Figure 5 shows that the ddDPCG method has nice behaviour; the residual decreases very regularly. The DPCG method shows one jump in the decrease in residual, the PCG shows six jumps and the ddPCG, althought not uneffective, is somewhat irregular and seems to show two or three jumps.

The jumps could be caused by 'difficult' eigenvalues of the system. Typically, PCG has difficulty with reducing the residual in the span of the smallest eigenvectors (that is, the eigenvectors corresponding to the smallest eigenvalues). We know that real rigid body modes are eigenvectors corresponding to eigenvalue zero, so the approximate rigid body modes caused by the sphere are approximately zero. Deflation of those eigenvectors could lead to the smooth decrease in residual, which is shown by the ddDPCG method in Figure 5. The reason for irregular behavior of the ddPCG method is probably due to the coarse grid correction. This corrector uses basically the same information as the rigid body mode deflation, but than on the whole domain instead of only on the sphere. The reason for the single jump in DPCG is unknown. There could be a rigid body mode hidden in the geometry of the block, which would also explain the relatively fast convergence of the ddPCG compared to PCG.

Test problem *Block9* consists of four materials. Three disjoint small cubes of high stiffness materials are contained within a large block of low stiffness material. The difference in stiffness between the small cubes and the block is of order $10^5 \sim 10^6$. The idea of test problem *Block9* is analogue to test problem *Block5*, although with three rigid bodies. In this case, three rigid bodies imply 18 deflation vectors. Due to the sharp edges imposed by the cubes, the iterative solver PCG performs poorly using 813 iterations and approximately 10.5 seconds. Using deflation (DCPG) this reduces to 135 iterations and less then 3 seconds. The convergence can be seen in Figure 6.

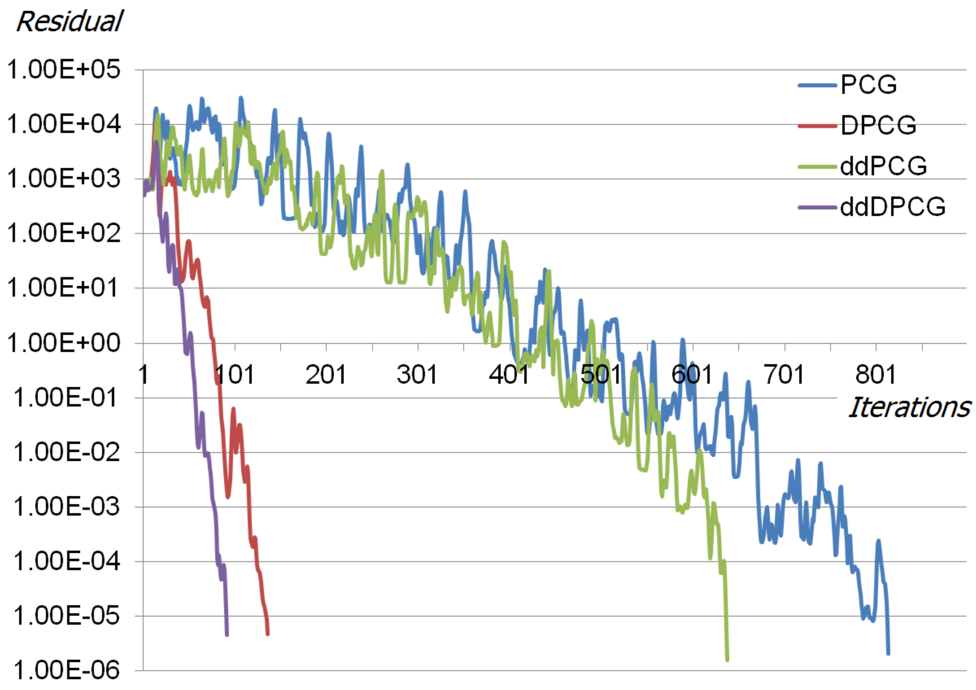


Figure 6: Illustrative result of rigid body mode deflation in comparison with other different solution methods for test problem *Block9*.

The current implementation has severe short-comings, which need be to resolved:

- Only deflation based on material properties
- Model may not contain interface elements or spring elements
- Assumption that deflation of stiffest material is allowed; the stiffest materials are

deflated without checking the boundary conditions

- Assumption that each material is *one* rigid body, i.e., the corresponding elements are all interconnected
- Only applicable for one domain
- Only applicable for symmetric stiffness matrices (Conjugate Gradient)
- Reusing information in a nonlinear loop is not possible

A Additional solution strategies

A.1 Algebraic multigrid

Multigrid methods are efficient iterative methods for the solution of linear systems. The methods use two complementary processes, relaxation and coarse grid correction. In the relaxation phase a BIM iteration is used 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. This decomposition is analogue to deflation.

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. The algebraic multigrid is more adaptive and only need information from the stiffness matrix itself, although the costs per iteration are slightly higher than with geometric multigrid.

It seems that algebraic multigrid based on aggregates has a strong analogy with deflation. For the coarse grid projection of the algebraic multigrid we form

$$E = Z^T K Z, \tag{A.1}$$

with the projector $\Pi^{\in} = (I - ZE^{-1}Z^T K)$. With aggregation-based multigrid we define

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

similar as with subdomain deflation. The currently implemented coarse grid correction uses a (rigid body) extension of this form of algebraic multigrid. This technique can also be implemented as a stand-alone solution algorithm, where the coarsening process is repeatedly applied, see e.g. [18]. For a comparison of algebraic multigrid with deflation see e.g. [9].

A.2 Physics-based domain decomposition

In Section 4 is discussed that an effective domain decomposition should have three objectives: minimize the number of overlap degrees of freedom, minimize the variation in subdomain sizes and separate degrees of freedom associated with different material properties [13]. A partitioning algorithm, aiming at all three objectives and starting with the third objective, is given in [13]. Such an approach could be advantageous compared to the current implementation, where the third objective is neglected.

B Planning

Figure 7 on page 45 illustrates the planning.

References

[1] Klaus-Jürgen Bathe. *Finite element procedures.* Prentice Hall, 1996.

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

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

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

[5] Pierre Gosselet and Christian Rey. On a selective reuse of krylov subspaces in newton-krylov approaches for nonlinear elasticity. *Proceedings of the fourteenth international conference on domain decomposition methods*, pages 419–426, 2003.

[6] Pierre Gosselet, Christian Rey, and Julien Pebrel. Total and selective reuse of krylov subspaces for the resolution of sequences of nonlinear structural problems. *CoRR*, abs/1301.7530, 2013.

[7] Markus Haase. Lectures on Functional Analysis. Delft Institute of Applied Mathematics, 2012.

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

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

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

[11] K. Kahl and H. Rittich. Analysis of the deflated conjugate gradient method based on symmetric multigrid theory. 2012.

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

[13] F.J. Lingen, P.G. Bonnier, R.B.J. Brinkgreve, M.B. van Gijzen, and C. Vuik. 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.

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

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

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

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

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

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

[20] B.F. Smith. Domain decomposition methods for partial differential equations. Technical report, 1990.

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

[22] TNO DIANA. *DIANA Finite Element Analysis, User's Manual, Analysis Procedures*, draft edition, 2013.

[23] TNO DIANA. *DIANA Finite Element Analysis, User's Manual, Element Library*, draft edition, 2013.

[24] TNO DIANA. *DIANA Finite Element Analysis, User's Manual, Material Library*, draft edition, 2013.

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

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

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

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

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

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

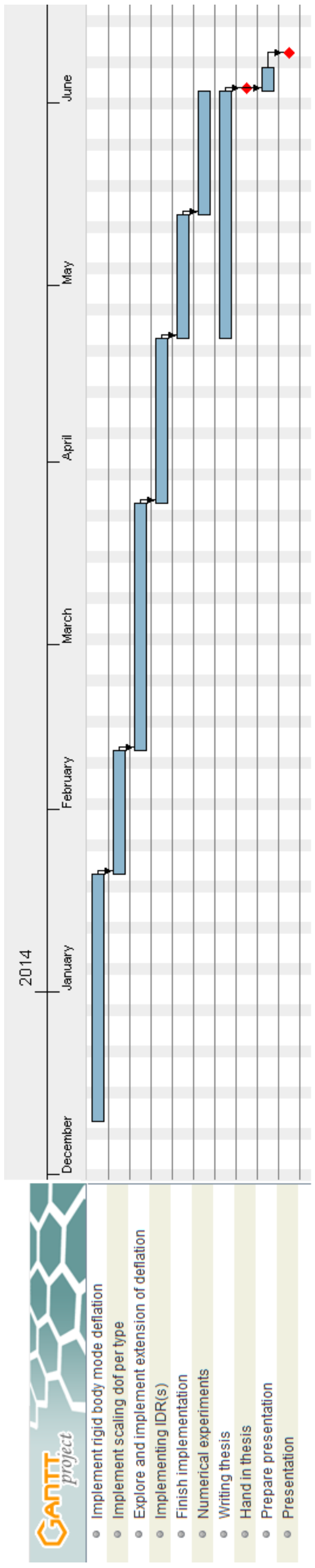


Figure 7: Gantt chart