Methods for improving sequentially linear analysis

A literature study

by

W. Swart

as partial fulfilment to obtain the degree of Master of Science

at the Delft University of Technology,

to be presented on Friday February 23, 2017 at 10:00 AM.

Student number:	4234898	
Project duration:	December 1, 2017 – September 1, 2018	
Thesis committee:	Dr. ir. M. van Gijzen,	TU Delft, supervisor
	Dr. ir. G. Schreppers,	DIANA FEA
	Prof. dr. ir. J. G. Rots	TU Delft
	Dr. ir. W. T. van Horssen	TU Delft

TU Delft DIANA FEA

Definitions

Definition 1. *Let Ω be bounded by $\partial\Omega$. Then the following function spaces are defined:*

$$
\begin{aligned}
C(\Omega) &:= \left\{ f : \Omega \to \mathbb{R} \;\middle|\; f \text{ continuous over } \Omega \right\} \\
C^p(\Omega) &:= \left\{ f : \Omega \to \mathbb{R} \;\middle|\; f \text{ is up to } p \text{ times continously differentiable over } \Omega \right\} \\
C_0(\Omega) &:= \left\{ f \in C(\Omega) \;\middle|\; f|_{\partial\Omega} = 0 \right\} \\
L^p(\Omega) &:= \left\{ f : \Omega \to \mathbb{R} \;\middle|\; \int_\Omega |f|^p \mathrm{d}\Omega < \infty \right\} \\
H^1(\Omega) &:= \left\{ f \in L^2(\Omega) \;\middle|\; \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \in L^2(\Omega) \right\}
\end{aligned}
$$

Definition 2. *Let $\mathbf{u} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$. Then A is:*

$$
\begin{aligned}
\textit{Symmetric if and only if:} \quad & A^T = A \\
\textit{Positive definite if and only if:} \quad & \mathbf{u}^T A\mathbf{u} > 0\,, \quad \forall \mathbf{u} \neq \mathbf{0}
\end{aligned}
$$

A matrix A that is both symmetric and postive definite is called symmetric positive definite (SPD).

Definition 3. *Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and let $A \in \mathbb{R}^{n\times n}$ be SPD. The A-inner product of $\mathbf{x}, \mathbf{y}$ is defined as $(\mathbf{x}, \mathbf{y})_A = \mathbf{x}^T A\mathbf{y}$.*

Definition 4. *Given $1 \leq p < \infty$, the p-norm (Hölder norm) of a vector $\mathbf{u} \in \mathbb{R}^n$ denotes as $\|\mathbf{u}\|_p$ is defined by*

$$
\|\mathbf{u}\|_p = \left(\sum_{i=1}^{n} |u_i|^p \right)^{\frac{1}{p}}.
$$

In particular it holds that

$$
\|\mathbf{u}\|_\infty = \max_{i=1,\ldots,n} |u_i|\,.
$$

Definition 5. *Given $1 \leq p < \infty$, the p-norm (Hölder norm) of a matrix $A \in \mathbb{R}^{m\times n}$ denoted as $\|A\|_p$ is defined by*

$$
\|A\|_p = \sup_{\mathbf{u}\in\mathbb{R}^n\setminus\{\mathbf{0}\}} \frac{\|A\mathbf{u}\|_p}{\|\mathbf{u}\|_p}\,.
$$

For $p = \infty$ the following expression for the matrix norm exists

$$
\|A\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^{n} |A_{i,j}|\,.
$$

Definition 6. *Let $A \in \mathbb{R}^{n\times n}$. The spectral condition number measured in p-norm $\kappa_p(A)$ of A is defined as*

$$\kappa_p(A) = \|A\|_p \left\|A^{-1}\right\|_p .$$

In particular if A is SPD it holds that

$$\kappa_2(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} .$$

Definition 7. *The A-norm of a vector $\mathbf{u}$ is defined as $\|\mathbf{u}\|_A = \sqrt{(\mathbf{u},\mathbf{u})_A}$.*

Definition 8. *Let $A \in \mathbb{R}^{n\times n}$ and $p, q \in \mathbb{R}_{\geq 0}$. Then A is said to have lower bandwidth p if and only if p is the smallest number such that $a_{ij} = 0$ whenever $i > j + p$. Analogously, A is said to have upper bandwidth q if and only if q is the smallest number such that $a_{ij} = 0$ whenever $j > i + q$.*

If $p = 0$ ($q = 0$) the matrix A is upper (lower) triangular.

Contents

1 Introduction

1.1. Background and motivation

In the recent years, the structural failure of buildings has lead to increased attention to the assessment of structural risks. Examples are the cracking of masonry buildings in Groningen due to gas extraction or the colllaapse of a concrete floor in Eindhoven due to technical errors. Not only residents but also insurance companies are interested in having a clear view of the risks involved and what damage can be expected.

To assess these risks, a nonlinear finite element analysis (NLFEA) can be carried out. In order for these assessments to produce reliable results, it is important that the adopted numerical method used is robust. However, when analyzing for example masonry or concrete structures robustness issues can arise. These issues are inherent to the iterative nature of NLFEA. Most of these analyses apply loads incremental after which an equilibrium is re-established between internal and external forces. Once the imbalance is sufficiently small the load is incremented again and the process is repeated. In most cases this iterative process works relatively well, however problems arise when large deformations occur in-between load increments. The masonry and concrete structures which are often subject to the NLFEA are characterized by brittle behaviour meaning that cracks can occur suddenly and propagate rapidly. As a result, the iterative procedure may not be able to find a converged solution.

To address these issues, Rots [15] proposed an alternative robust finite element method named sequentially linear analysis (SLA). The main assumption of the method is that it assumes incremental material degradation, that is the stiffness and strength properties of the material degrade stepwise. To locate at which point in the model the damage is applied a selection procedure is used which compares the current strength to the stresses as obtained with a linear analysis. This way a critical load factor can be determined with which the linear analysis load can be scaled such that the stress reaches the current strength only in one element, resulting in progressive damage. After the damage is incremented, the

stiffness matrix is recomputed and the process is repeated. With this method, the nonlinear behaviour of structures can be approximated with a sequence of linear analyses. In this incremental method, no iterative process is required making SLA inherently stable.

SLA has been successfully implemented into *DIANA* (Displacement Analyzer), a software package developed by *DIANA FEA BV*. Where standard finite element packages can only solve a limited range of conventional engineering problems, *DIANA* is able to handle non-conventional problems. Examples are the analysis of big structures such as dams; stresses induced from extreme loading conditions such as fire; earthquakes; explosions or complex non-linear behaviour. Ongoing research is performed on *DIANA* to maintain its reputation as being the best software package of its kind.

While SLA has been proven to be effective in robustly simulating brittle failure [18], it can be computationally intensive. One of the main reason is the possibility of having to solve a significant number of linear systems. This can occur in structures where the damage is (almost) fully incremented for many elements. Another key factor is the absence of efficient re-usage of solutions from previous damage increments. Since the damage increments are only applied to single elements, the resulting stiffness matrix remains mostly unchanged between damage increments. Only a small number of entries corresponding to the particular critical element are adjusted meaning that the system of linear equations is similar to that of the previous increment. Therefore, calculating the solution without exploitation of this property possibly comes at the cost of significant additional computing time, especially for large problems.

Due to the possibility of SLA requiring significant computer time, a desire developed at *DIANA FEA BV* to aim some of the research on these issues. As part of this desire, a master thesis position was made available with the aim of addressing the mentioned issues. While the first reason mentioned above is an inherent problem of the structural behaviour, the other reason is a result of the implementation of the SLA procedure. Therefore, the aim of this thesis, part of which is this literature study, is to exploit the incremental approach of SLA to achieve improved computing times. This literature study will provide the necessary background information on NLFEA, SLA, solution methods and some proposed techniques for improving these methods.

1.2. Research question

The main research question that will be addressed in the thesis is the following:

How can sequentially linear analysis be improved such that it requires reduced computing time?

Chapter 8 provides a detailed description of research proposal including sub-questions and methodology. The next section will provide an overview of the structure of this literature study.

1.3. Outline

This literature study will provide the necessary background information for the thesis. First an introduction is given on the finite element method including a brief description on its application to structural problems. Then NLFEA and the proposed method of SLA are introduced. Subsequently, two different classes of solution methods are discussed for the linear analyses in SLA followed by techniques for improving these methods. A detailed overview of the structure of this literature study is provided below.

Section 2: Introduction to the displacement based finite element method and its application to structural problems.

Section 3: Explanation of nonlinear analyses problems and incremental-iterative solution techniques to solve such problems.

Section 4: Explanation of the general idea of sequentially linear analysis, an alternative approach to solving structural problems.

Section 5: Direct solution techniques for solving systems of linear equations, including implementations into DIANA.

Section 6: Iterative solution techniques for solving systems of linear equations, including preconditioning and deflation.

Section 7: Solution technique for reusing solutions of linear systems after a small rank update to the system matrix.

Section 8: Research proposal for the remainder of the thesis.

2

Finite element method

The finite element method (FEM) is a numerical method for solving partial differential equations (PDE). The method is employed extensively in various areas such as for example the analysis of solids and structures, heat transfer and fluid flow. The flexibility of FEM in dealing with complex geometries is one of the reasons why the method is preferred over other numerical methods. Generally speaking, two different approaches exist to derive a linear system of equations from a given PDE. The first is known as Ritz' method and determines the solution by converting the PDE to a minimization problem. The second method determines the solution to the PDE by first translating it to a weak formulation, which is known as Galerkin's method. An introduction will be given for both methods. For a complete overview of FEM the interested reader is referred to standard textbooks such as Zienkiewicz [23] or Bathe [13].

Given a PDE it is possible under certain conditions, mainly the symmetry of the differential operator, to derive a minimization problem that is in some sense equivalent to the PDE. [20] These minimisation problems often aim at minimizing an underlying energy potential or shortest path to solve the equivalent PDE. The advantage of the minimization problem is that fewer boundary conditions are necessary meaning that a larger class of solutions is allowed. The boundary conditions that are still present in the minimization problem are called *essential*. On the other hand are boundary conditions that are present in the PDE formulation of the problem but not explicitely anymore in the minimization problem. These boundary conditions follow implicitly from the minimization formulation and are referred to as *natural* boundary conditions. To obtain a system of linear equations from the minimization problem, the solution is approximated by a weighted combination of linearly independent basis functions. Substitution of this approximation into the minimization problem and subsequent partial differentiation with respect to the unknown weights then yields a system of equations in terms of weights and integrals over the basis functions. Using a preferred numerical integration scheme allows these equations to be rewritten as a system of equations linear in the unknown weights.

As mentioned, the equivalent minimization problem does not necessarily exist for arbi-

trary PDEs. In these cases, Galerkin's method can be used to derive a weak formulation from the PDE. In situations where the minimization does exist it is equivalent to the weak formulation, making the weak formulation a more generally applicable method. In the next sections, the definition of the weak formulation will be given along with its application to structural problems.

2.1. Weak formulation

The aim of the weak formulation is to allow a larger solution class than the PDE admits. To illustrate how to obtain the weak formulation of a given PDE, consider a classical Poisson problem:

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = g(x,y) & \text{on } \partial\Omega \end{cases} \tag{2.1}$$

Multiplication with a so-called test function $\varphi \in C_0^1(\Omega)$ and integration over the domain Ω the boundary value problem from Equation (2.1) can be written as:

$$-\int_\Omega \varphi \Delta u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega\,, \quad \forall \varphi \in C_0^1(\Omega)\,. \tag{2.2}$$

Applying integration by parts and Gauss's divergence theorem to Equation (2.2) allows it to be rewritten as:

$$-\int_\Omega \nabla(\varphi \nabla u) - \nabla\varphi\nabla u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega\,, \quad \forall \varphi \in C_0^1(\Omega) \tag{2.3}$$

$$\Longleftrightarrow -\int_{\partial\Omega} (\varphi \nabla u)\cdot \mathbf{n} \mathrm{d}\Gamma + \int_\Omega \nabla\varphi\nabla u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega\,, \quad \forall \varphi \in C_0^1(\Omega) \tag{2.4}$$

Since it holds that $\varphi \in C_0^1(\Omega)$, the integral over the boundary $\partial\Omega$ vanishes and Equation (2.4) simplifies to:

$$\int_\Omega \nabla\varphi\nabla u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega\,, \quad \forall \varphi \in C_0^1(\Omega)\,. \tag{2.5}$$

The weak formulation corresponding to the boundary value problem of Equation (2.1) is thus:

$$\begin{cases} \text{Find } u \in C_g(\Omega) \text{ such that} \\ \displaystyle\int_\Omega \nabla\varphi\nabla u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega\,, \quad \forall \varphi \in C_0^1(\Omega) \end{cases} \tag{2.6}$$

In the original PDE the solution u had to be twice differentiable due to the Laplace operator. However, in the weak formulation (2.6) it can be seen that u only needs to be once differentiable. This illustrates the fact that the weak formulation allows a larger class of solutions than the corresponding PDE.

The next section will discuss Galerkin's method for deriving the system of linear equations from the weak formulation.

2.2. Galerkin's method

Galerkin's method solves the weak formulation (2.6) by dividing the area into a finite number of non-overlapping elements and approximating the solution by a linear combination of basis functions defined on these elements.

Dividing the area of interest Ω into n_{el} non-overlapping elements then allows the domain to be decomposed as:

$$\overline{\Omega} \simeq \bigcup_{k=1}^{n_{el}} \overline{e_k}. \tag{2.7}$$

Each element e_k consists of a number of nodes, some of which can be shared with neighbouring elements.

Choosing a set of basis functions $\{\varphi_j\}$ and introducing the notation $\mathbf{x} = (x, y)$[1], the solution u can be approximated as follows:

$$u(\mathbf{x}) \approx u^n(\mathbf{x}) = \sum_{j=1}^{n} u_j \varphi_j(\mathbf{x}). \tag{2.8}$$

To be able to ensure that $u(\mathbf{x}_i) = u_i$ the basis functions have to be chosen such that $\varphi_j = 1$ only at $\mathbf{x} = \mathbf{x}_j$ and 0 elsewhere. Therefore, the basis functions have to equal the Kroneckerdelta function:

$$\varphi_i(\mathbf{x}_j) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{2.9}$$

Substitution of the approximation of u in weak formulation (2.6) then yields:

$$\begin{aligned} \int_\Omega \nabla\varphi_i \nabla \left(\sum_{j=1}^{n} u_j \varphi_j \right) \mathrm{d}\Omega &= \int_\Omega \nabla\varphi_i \sum_{j=1}^{n} u_j \nabla\varphi_j \mathrm{d}\Omega \\ &= \sum_{j=1}^{n} u_j \int_\Omega \nabla\varphi_i \nabla\varphi_j \mathrm{d}\Omega \\ &= \int_\Omega \varphi_i f \mathrm{d}\Omega \end{aligned} \tag{2.10}$$

[1] $\mathbf{x} = (x, y, z)$ in $\mathbb{R}^3$.

Introducing two new variables for the integrals:

$$K_{ij} = \int_\Omega \nabla\varphi_i \nabla\varphi_j \mathrm{d}\Omega = \sum_{m=1}^{n_{el}} \int_{e_m} \nabla\varphi_i \nabla\varphi_j \mathrm{d}\Omega$$
$$= \sum_{m=1}^{n_{el}} K_{ij}^{e_m} \tag{2.11}$$
$$f_i = \int_\Omega \varphi_i f \mathrm{d}\Omega = \sum_{m=1}^{n_{el}} \int_{e_m} \varphi_i f \mathrm{d}\Omega$$
$$= \sum_{m=1}^{n_{el}} f_i^{e_m} \tag{2.12}$$

In structural settings, the terms $K_{ij}^{e_m}$ are referred to as element stiffness matrices and $f_i^{e_m}$ element force vectors which add elementary contributions to the system stiffness matrix and force vector respectively. By substitution of Equations (2.11) and (2.12) into Equation (2.10) the expression simplifies to:

$$\sum_{j=1}^{n} K_{ij} u_j = f_i\,, \quad \forall i \in \{1, \dots n\}\,, \tag{2.13}$$

where n is the total number of nodal degrees of freedom. These n equations can be assembled to obtain the matrix-vector notation:

$$K\mathbf{u} = \mathbf{f}. \tag{2.14}$$

The next section will provide an introduction on how the finite element approach described in this section is applied to structural problems.

2.3. Application to structural problems

In static structural problems, the user is interested in finding the structural response as a result of a given load. This response is given by displacements, stresses and strains. While it is clear what a displacement is, this does not necessarily hold for stress and strain.

The two quantities (mechanical) stress and strain are closely related. Where stress, denoted by $\boldsymbol{\sigma}$, is a quantity that expresses the internal forces per area that neighbouring particles exert on each other, strain, denoted by $\boldsymbol{\varepsilon}$, is a measure for the deformation of a material. Stresses can occur in the absence of strains, for example in a beam supporting a weight. The presence of the weight induces stresses in the beam, but it does not necessarily have to strain. It is even possible for stresses to occur it the absence of external forces due to, for example, self-weight. The relation between stresses and strains is typically expressed using a material dependant stress-strain curve.

A detailed description of how the finite element method can be applied to structural problems can be found in Zienkiewicz [23]. For static structural problems, the local strain $\boldsymbol{\varepsilon}$ can

be calculated by taking the product of the *strain-displacement* relation B_i and the displacements $\mathbf{u}$:

$$\boldsymbol{\varepsilon} = B_i \mathbf{u} \,. \tag{2.15}$$

The strain-displacement relation B_i is only defined for a particular point and thus needs to be evaluated for every point. Assuming linear elastic behaviour in the structure, the local stresses can be calculated as:

$$\boldsymbol{\sigma} = D_i \left(\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_0\right) + \boldsymbol{\sigma}_0 \,. \tag{2.16}$$

In this equation D_i is the elasticity matrix which is a function of the related material properties such as Young's modulus (measure of stiffness of solids) and Poisson's ratio (measure of contraction of a material under tension). Furthermore, it is possible that a structure is under stresses and strains prior to the analysis. This is taken into account with the terms $\boldsymbol{\sigma}_0$, $\boldsymbol{\varepsilon}_0$ respectively.

The system of linear equations can be obtained by imposing the principle of virtual work by equating the external and internal work done by the various forces and stresses during the virtual displacement. The textbooks by Zienkiewicz [23] and Bathe [13] provide extensive elaborations and examples on the principle of virtual work. Imposing the principle of virtual work it follows that the element stiffness matrices $K_{ij}^{e_m}$ from Equation (2.11) can be written as:

$$K_{ij}^{e_m} = \int_\Omega B_m^T D_m B_m \mathrm{d}\Omega \,. \tag{2.17}$$

In order to be able to map the local element numbering to the global numbering, an element-dependant mapping T_i has to be introduced. Applying this mapping to all element stiffness matrices in Equation (2.11) yields:

$$K_{ij} = \sum_{m=1}^{n_{el}} T_m^T K_{ij}^{e_m} T_m \,. \tag{2.18}$$

Similarly, the principle of virtual work allows the right-hand side of Equation (2.14) to be written as a sum of integrals over the geometry (boundary).

In order to be able to obtain a solution, the integrals over the geometry have to be calculated. In practical problems, this is not possible analytically and hence numerical integration has be applied. The next section will elaborate on how the element integrations are performed which are necessary to obtain a set of linear equations that can be solved.

2.4. Numerical integration

In the previous sections it was shown how Galerkin's method can be used to derive a system of linear equations from the weak formulation of a PDE. In this derivation, Equations (2.11) and (2.12) still require the integration over the elements. In most practical problems, exact integration is not possible hence a numerical integration scheme has to be used. Some

well-known numerical integration schemes are Newton-Cotes of various degrees or Gaussian quadrature.

All of the numerical integration schemes have in common that they evaluate the function to be integrated in specific points, called integration points. The number of these integration points is dependent of the chosen numerical scheme. In DIANA, Gaussian quadrature is chosen as integration scheme as this minimizes the required number of integration points. To illustrate Gaussian quadrature consider again the element force vector from Equation (2.12). Applying Gaussian quadrature to this equation yields:

$$\begin{aligned} f_i^{e_m} &= \int_{e_m} \varphi_i(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\Omega \\ &:= \int_{e_m} g_i(\mathbf{x}) \mathrm{d}\Omega \\ &\approx \sum_{i=1}^{n_\xi} w_\xi g_i(\boldsymbol{\xi}_i) \end{aligned}$$

Here are $\boldsymbol{\xi}_i$ the integration points, n_ξ the number of integration points and w_ξ describe the weight function of the applied method for the specific integration interval.

3

Nonlinear analysis

In nonlinear finite element analysis the relation between the exerted force and the displacement is no longer linear which can be the result of material-, geometric- or contact nonlinearities, or a combination. Similar to a linear analysis, the problem is to find the displacement of the structure which equilibrates the external and internal forces such that a stationary state is found. To solve this, the problem is not only discretized spatially but also temporal (increments). To obtain a feasible solution within one increment, an iterative procedure is used to equate the forces. Due to the combination of temporal discretization and iterative nature, this solution procedure is referred to as incremental-iterative.

Discretizing the displacement vector in time (increment) as

$$\mathbf{u}^{t+\Delta t} = \mathbf{u}^{t} + \Delta\mathbf{u}\,, \tag{3.1}$$

the problem can then be defined as finding $\Delta\mathbf{u}$ such that the resulting displacement equates the internal and external foces:

$$\mathbf{f}_{\text{int}}(\mathbf{u}^{t+\Delta t}, \mathbf{u}^{t}, \ldots, \mathbf{u}^{1}) = \mathbf{f}_{\text{ext}}(\mathbf{u}^{t+\Delta t})\,. \tag{3.2}$$

Due to the possibility of the internal forces depending on the history of the displacements these previous displacement vectors are included in Equation (3.2) in the argument of the internal force. To be able to solve the above problem with a iterative method such as Newton-Raphson, Equation (3.2) is rewritten as follows. Find a displacement increment $\Delta\mathbf{u}$ such that

$$\mathbf{g}(\Delta\mathbf{u}) := \mathbf{f}_{\text{ext}}(\mathbf{u}^{t+\Delta t}) - \mathbf{f}_{\text{int}}(\mathbf{u}^{t+\Delta t}, \mathbf{u}^{t}, \ldots, \mathbf{u}^{1}) = \mathbf{0}\,. \tag{3.3}$$

All methods available in *DIANA* adapt the displacement vector using iterative increments $\delta\mathbf{u}$

$$\Delta\mathbf{u}_{i+1} = \Delta\mathbf{u}_{i} + \delta\mathbf{u}_{i+1}\,, \tag{3.4}$$

where i is the iteration number.

This process is repeated until a displacement increment is found for which the residual force vector $\mathbf{g}$ reaches $\mathbf{0}$, up to a prescribed tolerance. At that point, a new increment is determined and the process repeats.

The difference between the available methods is the way in which the iterative increments are determined. The following sections will discuss the different iterative procedures that are available within *DIANA*. Note that it is not necessary to use the same method for all increments, in theory the force equilibrium can be reached with a different method for every increment.

3.1. Newton-Raphson methods

In the class of Newton-Raphson methods, generally two variants can be distinguished; *regular*- and *modified* Newton-Raphson. Both methods determine the iterative increment using the Jacobian, which in structural problem is the 'stiffness matrix'. Contrary to Chapter 2 in which stiffness matrix represented the linear relation between force and displacement, such linear a relation does not exist in nonlinear problems. Hence some kind of linearized form of the relation between the force and displacement vector is calculated. The difference between regular- and modified Newton-Raphson is the point at which this stiffness matrix is evaluated.

Regular Newton-Raphson recomputes the stiffness matrix every iteration. As a result, every prediction is based on the most recent information. This property is illustrated in Figure 3.1 where it can be seen that the tangent changes with each iterations within an increment.

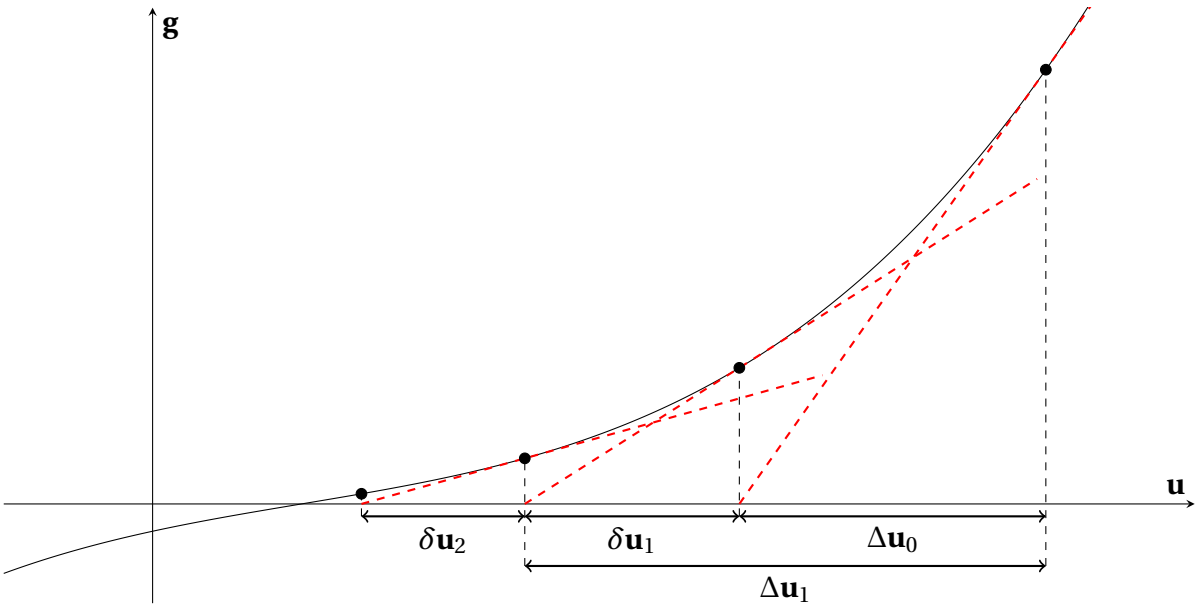


Figure 3.1: Example of regular Newton-Raphson.

The main advantage of using regular Newton-Raphson is its quadratic convergence. This implies that usually few iterations are required to attain the root of the residual force vector. However, due to the required evaluation of the stiffness matrix in every iteration it is relatively expensive. Furthermore, convergence of the approximations can not be guaranteed for sufficiently bad initial guesses.

The modified Newton-Raphson method only computes the stiffness matrix at the start of every increment. This implies that all predictions within an increment are based only on the information available at the start of the increment. An illustration of the modified Newton-Raphson can be seen in Figure 3.2 in which the constant tangent can be observed within an increment.

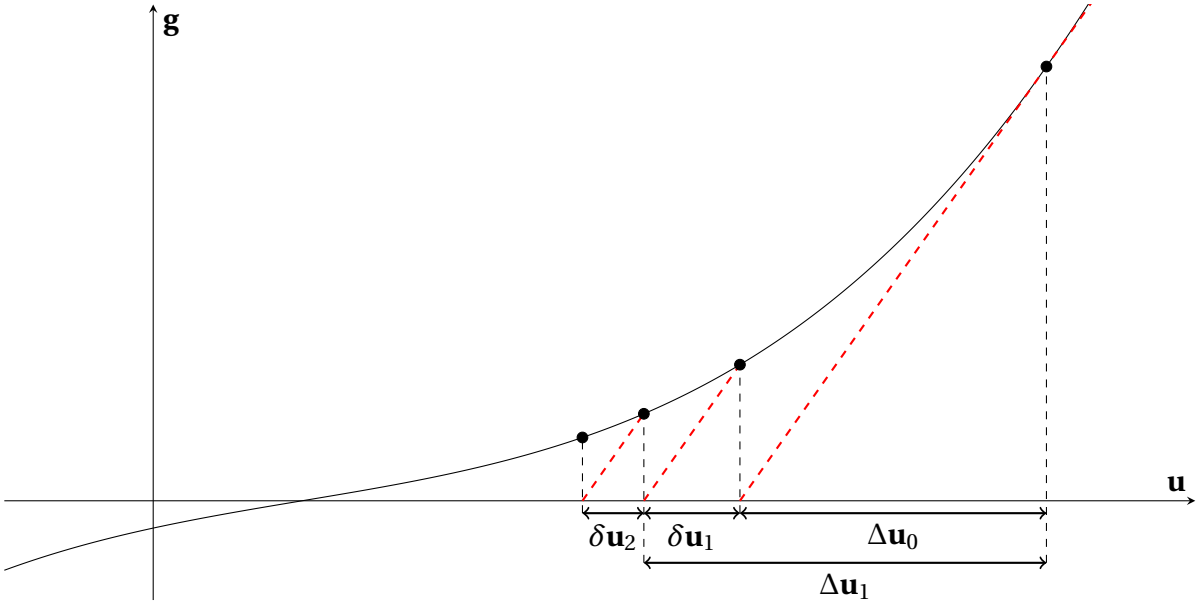


Figure 3.2: Example of modified Newton-Raphson.

Usually modified Newton-Raphson requires more iterations than regular Newton-Raphson to obtain the force equilibrium due to the fact that only information from the start of an increment is used. However, since the stiffness matrix is only set-up once at the start of an increment, the iterations of modified Newton-Raphson are faster. Furthermore, if a direct solution method is used to obtain the iterative increments the costly decomposition of the stiffness matrix need only be computed once after which the relatively cheap substitutions will suffice. In figure 3.2 it can be seen that for the same function, starting point and number of iterations the modified Newton-Raphson method is considerably further from the root than regular Newton-Raphson as was seen in Figure 3.1.

3.2. Quasi-Newton methods

In situations where the computation of the stiffness matrix in every iteration is too expensive, an approximate of the stiffness matrix can be used instead. Any method that replaces

the exact stiffness matrix with such approximation is referred to as a quasi-Newton method. Note the misleading use of *exact* stiffness matrix, which in fact is already some linearized form between the force and displacement vector. Three different quasi-Newton methods have been implemented each of which approximate the stiffness matrix differently. Two of these are:

$$\text{Broyden:} \quad K_{i+1}^{-1} = K_i^{-1} + \frac{\left(\delta \mathbf{u}_i - K_i^{-1}\delta \mathbf{g}_i\right)\delta \mathbf{u}_i^T K_i^{-1}}{\delta \mathbf{u}_i^T K_i^{-1}\delta \mathbf{g}_i} \tag{3.5}$$

$$\text{BFGS:} \quad K_{i+1}^{-1} = \left(I + \frac{\delta \mathbf{u}_i \delta \mathbf{g}_i^T}{\delta \mathbf{u}_i^T \delta \mathbf{g}_i}\right) K_i^{-1} \left(I - \frac{\delta \mathbf{g}_i \delta \mathbf{u}_i^T}{\delta \mathbf{u}_i^T \delta \mathbf{g}_i}\right) + \frac{\delta \mathbf{u}_i \delta \mathbf{u}_i^T}{\delta \mathbf{u}_i^T \delta \mathbf{g}_i} \tag{3.6}$$

The inverses of the stiffness matrices K_{i+1}^{-1} are not calculated explicitly, but are calculated by successive application of the above equations with the Jacobian K_0 from the start of the increment and the iterative increments. This approach implies that for every iteration an additional iterative increment vector has to be stored and that additional vector calculations are required.

The third implemented quasi-Newton method was proposed by Crisfield [4] and suggests only to use the most recent correction vector. This prevents the increasing storage needed to store all intermediate iterative increments. Especially in problems with a large number of degrees of freedom this can result in significant savings in required memory.

3.3. Additional methods

Besides the Newton-Raphson and quasi-Newton methods some other iterative methods are available to obtain a force equilibrium.

The first two methods are the linear- and constant stiffness methods which both use the same stiffness matrix during all iterations within an increment. The linear stiffness method uses the stiffness matrix from the first increment. This means that it costs the least per iteration than all other methods, while potentially requiring significantly more iterations. On the other hand, the constant stiffness method uses the stiffness matrix from the previous increment. If one uses the constant stiffness method for all increments, it is equivalent to the linear stiffness method.

A third method is the continuation method. If a displacement is relatively continuous, the displacement of the previous increment can be used as an initial guess for the subsequent increment. This initial guess can then serve as a starting point for one of the other mentioned methods.

The last method that is available in *DIANA* is line search. The problem with most former mentioned methods is that divergence can occur for a poorly chosen initial guess. If these methods fail, line search can still be useful. The method uses the iterative increments $\delta \mathbf{u}$ from one of the formerly mentioned methods and then scales Equation (3.4) to obtain

$$\Delta \mathbf{u}_{i+1} = \Delta \mathbf{u}_i + \eta \delta \mathbf{u}_{i+1}\,. \tag{3.7}$$

The scaling parameter η is determined by minimizing the energy potential from which the scaled iterative increment can be considered as the best solution in the predicted direction. For an in-depth elaboration on this topic, Crisfield [4] provides an excellent starting point.

3.4. Convergence criteria

The iterative process of the methods mentioned in the previous sections have to be terminated once force equilibrium has been reached with sufficient accuracy. Not only is the iterative process stopped once it reaches the prescribed accuracy, but also once a predefined maximum number of iterations is attained which prevents excessive iterations due to poorly chosen stopping criteria. Two of the most frequently used stopping criteria in *DIANA* are the following.

The first stopping criterion is based on the force norm, which is the Euclidean norm of the residual force vector $\mathbf{g}_i$: $\sqrt{\mathbf{g}_i^T \mathbf{g}_i}$. Checking this norm against the force norm at the start of the increment allows to check for convergence and yields the force norm ratio:

$$\text{Force norm ratio:} \quad r = \frac{\sqrt{\mathbf{g}_i^T \mathbf{g}_i}}{\sqrt{\mathbf{g}_0^T \mathbf{g}_0}}.$$

The second stopping criterion is similar to the force norm ratio but instead it is based on the displacement norm, which is the Euclidean norm of the iterative increment $\delta\mathbf{u}_i$: $\sqrt{\delta\mathbf{u}_i^T \delta\mathbf{u}_i}$. Checking for convergence can then be achieved by checking this norm against the displacement norm at the start of the increment. This yields the displacement norm ratio:

$$\text{Displacement norm ratio:} \quad r = \frac{\sqrt{\delta\mathbf{u}_i^T \delta\mathbf{u}_i}}{\sqrt{\Delta\mathbf{u}_0^T \Delta\mathbf{u}_0}}.$$

Other possibilities for stopping criteria are based on the norm of the build up energy in the structure due to the iterative increments, or on a comparison between the residual force norm from the current and previous iteration.

3.5. Incremental procedures

The incremental-iterative procedure consists of two parts; an *incremental* and *iterative* part. The preceding sections of this chapter have all focused on the iterative part. For the increment part several solution methods are available. The most simple of these methods are *load control* and *displacement control*. A more advanced method is given by the *arc-length control* method. For a detailed description on these and related methods, the reader is referred to the *DIANA* Theory manual [3].

4

Sequentially linear analysis

The previous chapter described iterative procedures that can be used to obtain a solution in NLFEA. Generally, these iterative processes work relatively well in finding a sufficiently converged solution. However, in structures where brittle behaviour occurs suddenly and propagates rapidly, the iterative process may not be able to find a converged solution anymore. To address these issues, Rots [15] proposed sequentially linear analysis (SLA) as an alternative. This section will provide the necessary background information on SLA. A good starting point to this is a visualization of the step-by-step nature of SLA, which can be seen in Figure 4.1.

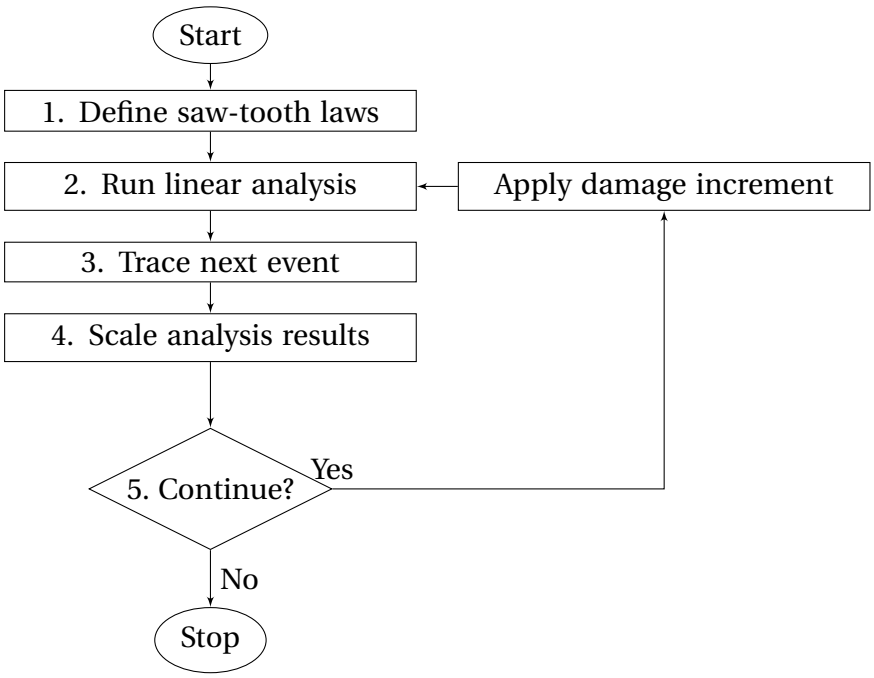


Figure 4.1: Flowchart of SLA step-by-step strategy.

With this strategy, the structural response of the structure is captured as a sequence of subsequent events. In this context, an event refers to a damage increment to a point in the structure. The general procedure of SLA is then given by the following steps.

1. Initialisation: definition of material deformation under tensile- or compressive stresses (saw-tooth laws)
2. Run the linear analysis with a (unit) load
3. Detect which element is closest to its stress limit
4. Scale the linear analysis according to the critical element
5. Check stopping criteria; if not reached apply damage increment and repeat

In the following sections, more in-depth elaboration will be given on the main steps of SLA. Section 4.1 will describe how material properties are discretized using saw-tooth laws. Since this chapter focuses on SLA, Section 4.2 only briefly presents what solvers are available to solve the resulting linear system of equations. An in-depth description on these solution procedures can be found in Chapters 5 and 6. Once the linear system is solved, the location of the next event is traced. This is described in Section 4.3. Lastly, Section 4.4 elaborates on the stopping criteria available within SLA.

4.1. Saw-tooth laws

A fundamental assumption within SLA is to approximate the nonlinear material behaviour with a series of linear relations. Therefore, as initialisation SLA requires a discretisation of the stress-strain curve which defines the material dependent relation between stress and strain. Figure 4.2 shows some of the predefined tensile stress-strain curves that are present in *DIANA*.

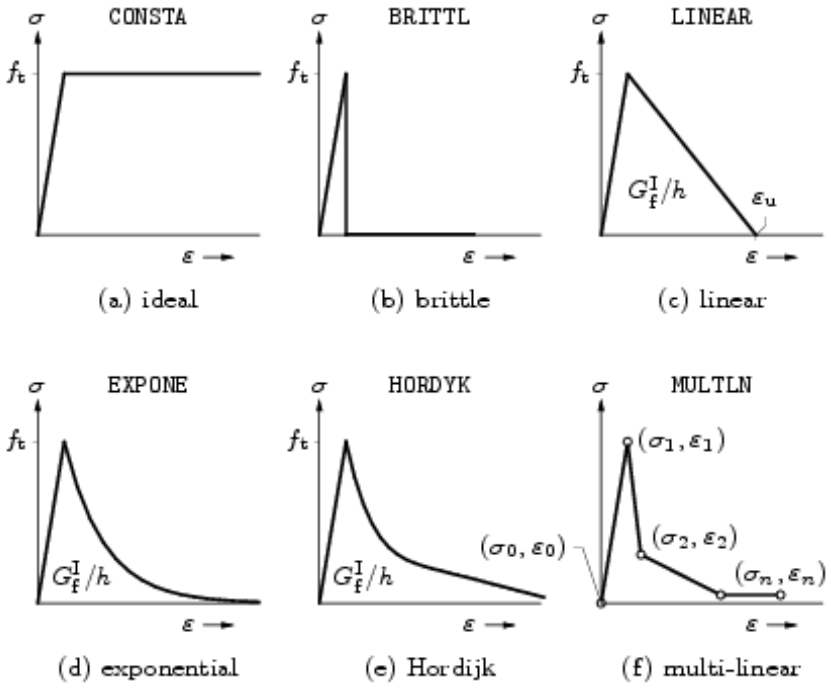


Figure 4.2: Predefined tensile softening curves in *DIANA*. Source: *DIANA* theory manual [3].

The last 5 curves display a decrease in required stress for increased strain. This type of nonlinear behaviour is referred to as material softening and characterises brittle materials such as concrete and masonry. In order to approximate this nonlinear behaviour with a sequence of linear relations, the stress-strain is discretised using either constant or variable strain increments. An example of the resulting saw-tooth law with variable strain increments is shown in the left-hand side of Figure 4.3.

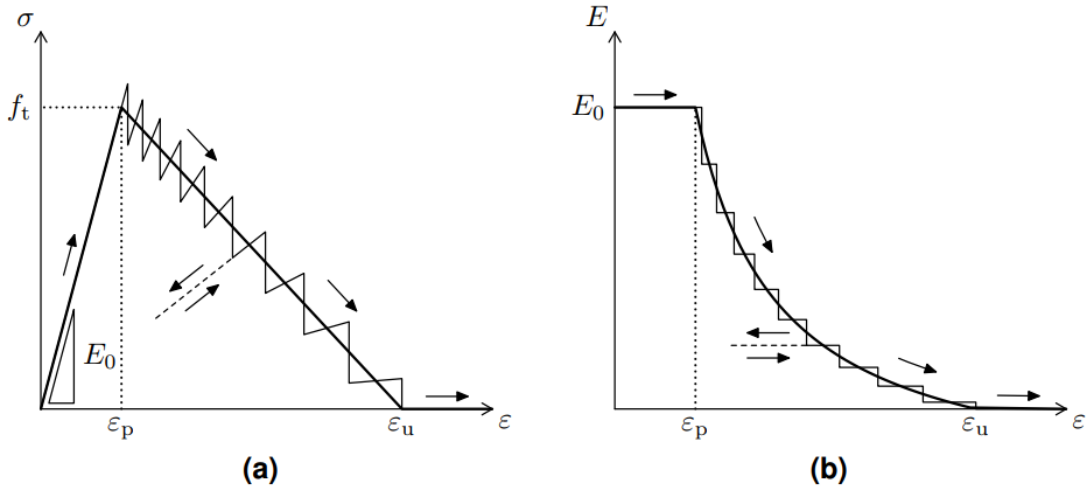


Figure 4.3: Example of a saw-tooth law and corresponding decrease in Young's modulus. Source: v.d. Graaf [18].

The resulting saw-tooth law transforms the continuous reduction in stiffness into discrete jumps. Once an integration point attains its stress limit, the next step with reduced stiffness and strength properties is assumed. This process is repeated until the stiffness has reached a predefined stopping criteria or equals 0. The situation in which the stiffness equals 0 corresponds to complete material failure. Each step can thus be seen as a damage increment reducing the material's strength properties.

4.2. Linear analysis

The previous section described how the nonlinear material behaviour is discretised such that, once the material reaches its limit stress, its strength properties can be decreased. Once this has been defined, all structural properties are available to assemble the system of linear equations. Depending on the type of loading (proportional/non-proportional), a certain load is assumed[1]. To solve the resulting system of equations several solution techniques and implementations of those are available in *DIANA*. All of these solution techniques can be categorized as a either direct- or iterative. Chapters 5 and 6 will provide an in-depth analysis of these two classes of solution methods and mention some implementations of these techniques.

4.3. Trace next event

In the linear analysis, a unit load on the structure was assumed. To determine in which element the next damage increment occurs, a critical load factor has to be determined for every element. This factor indicates how much the load in that point can increase until material failure occurs. The critical load factor is defined as:

$$\lambda^{(j)}_{\text{crit};i} = \frac{f^{(j)}_i}{\sigma^{(j)}_{\text{gov};i}}, \tag{4.1}$$

where $\sigma^{(j)}_{\text{gov};i}$ is the governing stress component for integration point i and $f^{(j)}_i$ the current material strength. Taking the minimum of all these load factors then yields a critical load factor:

$$\lambda^{(j)}_{\text{crit}} = \min_i \left(\lambda^{(j)}_{\text{crit};i}\right) \quad \forall i : \lambda^{(j)}_{\text{crit};i} > 0\,. \tag{4.2}$$

Scaling of the linear analysis with this critical load factor then results in the smallest load that will lead to progressive damage. It is assumed that in the scaled analysis, the governing stress reaches the material strength only in one of the integration points, the critical integration point.

[1] Since the focus of this thesis is on the mathematical side of SLA and not on the mechanical, detailed elaboration on non-proportional loading will be omitted. For such elaboration the interested reader is referred to Rots [15] or for a more detailed description to v.d. Graaf [18].

4.4. Stopping criteria

Before applying the damage to the critical integration point, it has to be decided whether or not to continue the analysis. The simplest stopping criteria is checking whether the number of linear analyses has reached a predefined maximal number [18]. However, other less trivial stopping criteria are also possible. One possibility is to terminate the analysis as soon as a certain displacement at a given degree of freedom is attained. Another possibility is to stop the analysis once the damage in an integration point reaches a certain percentage of complete material failure (90%-95%).

If the stopping criteria is not yet attained, the damage increment is applied to the critical integration point according to the saw-tooth law. A new linear system is then set-up and the analysis is repeated.

5

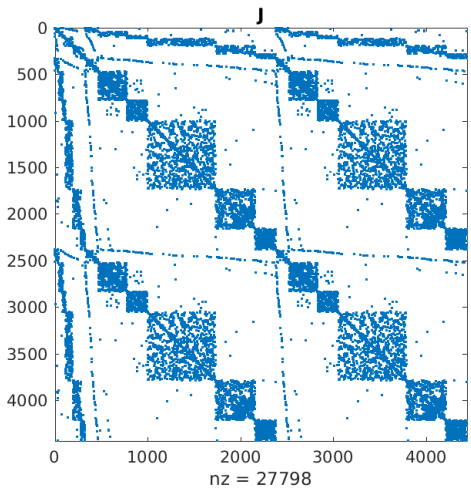
Direct methods for linear systems

The analysis of sequential linear analysis requires solving a linear system $K\mathbf{u} = \mathbf{f}$ after each damage increment. For any practical problem it is not feasible to invert the resulting stiffness matrix K directly. Therefore, two classes of solution methods exist to calculate the displacements $\mathbf{u}$: direct and iterative methods. This chapter will focus on the direct solution methods whereas the next chapter will focus on the latter.

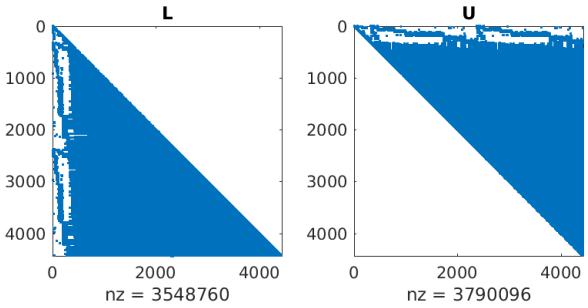
Typically, direct solution methods exist of three stages; a reordering of the system of equations, a decomposition of the system matrix, and a forward- and backward substitution to obtain the solution. The following three sections will discuss each of these stages separately.

5.1. Matrix reordering

The discretization of the structural equations using the finite element methods generally results in sparse stiffness matrices; matrices of which most entries are 0. When solving large sparse systems of equations it is desirable to exploit this property, as it reduces the number of calculations to be performed. However, in the decomposition of a sparse matrix the sparsity pattern is often destroyed and *fill-in* occurs meaning that the factors of the decomposed matrix contain significantly more non-zero entries than the original matrix. To illustrate the occurance of fill-in, Figure 5.1 shows a matrix decomposition without applying a reordering scheme to the matrix.



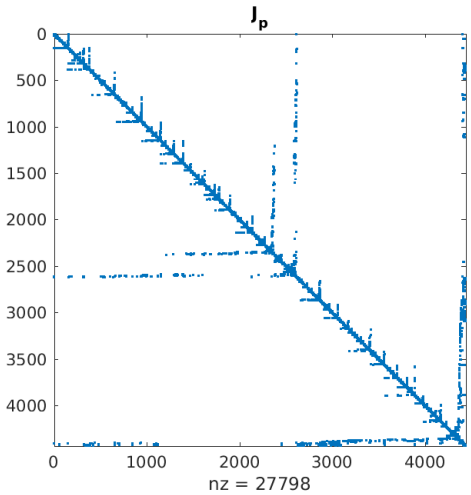
(a) Unreordered matrix.



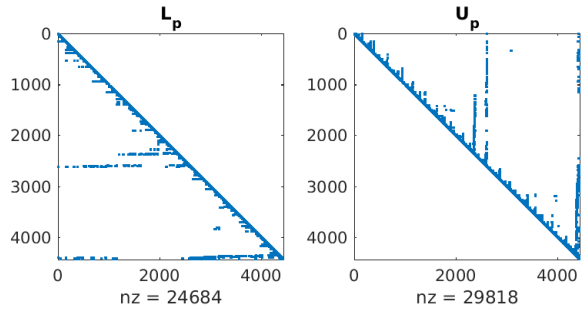
(b) LU decomposition of unreordered matrix.

Figure 5.1: Example of non-zero pattern of a matrix decomposition with fill-in.

Clearly, performing a matrix decomposition without applying a reordering scheme can result in significant fill-in. In the rest of the direct method this results in a severe performance penalty as every non-zero element has to be stored in memory, but also unnecessary computations have to be performed on these non-zero entries. To illustrate the significance of reordering, Figure 5.2 shows the same matrix and decomposition, only now after applying a reordering scheme (AMD) to the matrix.



(a) Reordered matrix.



(b) LU decomposition of reordered matrix.

Figure 5.2: Example of non-zero pattern of a matrix decomposition with reduced fill-in due to reordering (AMD).

From the number of non-zero elements it is clear that the reordering scheme has largely maintained the sparsity pattern. This example motivates the use of reordering schemes especially when solving large sparse systems of linear equations.

In general, a matrix reordering can be written as multiplications of permutation matrices P, Q with the original matrix K as

$$K\mathbf{u} = \mathbf{f} \quad \Longrightarrow \quad \left(PKQ^T\right)(Q\mathbf{u}) = P\mathbf{f} \qquad \text{(Complete pivoting)} \tag{5.1}$$

From Equation (5.1) it can be seen that *complete pivoting* permutes both the rows and columns. Note that for symmetric matrices it must hold that $P = Q$ in order to maintain symmetry of the permuted matrix. The aim is then to find reorderings P, Q such that the bandwidth of the resulting decomposition is minimal. It has been shown by Yannakakis [22] that finding such minimal fill-in reordering is NP-hard, meaning that no known efficient way of finding such reordering exists. Despite this, several well-known reordering heuristics exist of which some of the most used will be discussed.

5.1.1. Cuthill-McKee reordering

The Cuthill-McKee (CM) algorithm is a well-known reordering heuristic for reducing the bandwidth of sparse symmetric matrices [5]. The general idea of the algorithm is as follows. Assuming a square matrix K, consider K as the adjacency matrix of a graph. Every non-zero element of K represent a connection between the corresponding nodes in the graph. The algorithm then first finds the node with minimal vertex degree (number of adjacent nodes). From this node, a breadth-first search is performed. For every depth level, the nodes are relabelled according to the ascending vertex degree order. Once all nodes in the depth level are relabelled, another depth level is considered. This process repeats until all nodes are relabelled. The above algorithm can in short be written using the following pseudocode.

Algorithm 1 CutHill-Mckee pseudo-algorithm

1: Represent given matrix $K \in \mathbb{R}^{n\times n}$ as the adjacency matrix of a graph
2: Find vertex x with minimal degree, set $R_1 := (\{x\})$
3: **while** $|R_i| < n$ **do**
4: Find the adjacency set K_i of R_i
5: Sort the nodes in K_i with ascending vertex order
6: Relabel the nodes in K_i according the sorted vertex orders
7: Set $R_{i+1} = R_i \cup K_i$
8: **end while**

Instead of the CM algorithm, often the *reversed* CM algorithm is used as proposed by George [8]. This method inverts the labelling as provided by the CM algorithm and has been shown to often produce a superior ordering in terms of fill-in while the bandwidth remains unchanged.

5.1.2. Minimum degree reordering

The Minimum Degree (MD) algorithm is another well-known reordering scheme that can be used to reorder symmetric matrices. The general idea of MD is that it simulates n steps of Gaussian elimination. In each of these steps, one row and one corresponding column interchange is applied to the part of the matrix that remains to be factored such that the number of non-zero entries in the pivot row and column are minimized.

Many different implementations of the MD algorithm exist, such as *Approximate Minimum Degree* (AMD) or *Symmetric Approximate Minimum Degree* (SYMAMD).

5.2. Matrix decomposition

Most direct solution techniques for solving systems of large equations rely on some form of *Gaussian elimination*. The most basic form of Gaussian elimination reduces the original matrix $K \in \mathbb{R}^{n\times n}$ to a factorisation LU, in which L, U are lower and upper triangular respectively. To write the matrix K in the factored form, Gaussian elimination step-by-step adds multiples of one equation to another. The general idea of Gaussian is then as follows [2].

Step 1
Denote the original system $K\mathbf{u} = \mathbf{f}$ as $K^{(1)}\mathbf{u} = \mathbf{f}^{(1)}$ with $k_{i,j}^{(1)}, f_i^{(1)}$ the entries of $K^{(1)}, \mathbf{f}^{(1)}$.

Assume row 1 has a non-zero pivot; $k_{1,1}^{(1)} \neq 0$. This allows the definition of the row multipliers:

$$m_{i,1} = \frac{k_{i,1}^{(1)}}{k_{1,1}^{(1)}}, \quad i = 2, \ldots, n. \tag{5.2}$$

These multipliers quantify how many times the first row has to be added to the other $n-1$ rows in order to obtain 0 entries below the diagonal in the first column. Writing the row multipliers in the matrix notation as

$$M_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -m_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -m_{n,1} & 0 & \cdots & 1 \end{pmatrix}, \tag{5.3}$$

the unknown u_1 can be removed from equations $2, \ldots, n$ by multiplying the system of equations with the so-called Gauss transform M_1:

$$K^{(2)}\mathbf{u} =: M_1 K^{(1)}\mathbf{u} = M_1\mathbf{f}^{(1)} := \mathbf{f}^{(2)}. \tag{5.4}$$

To motivate the structure of $K^{(2)}$, write the matrix multiplication of Equation (5.4) as:

$$k_{i,j}^{(2)} = k_{i,j}^{(1)} - m_{i,1}k_{1,j}^{(1)}, \quad i, j = 2, \ldots, n$$
$$f_i^{(2)} = f_i^{(1)} - m_{i,1}f_1^{(1)}, \quad i = 2, \ldots, n$$

Equation (5.4) can then be written as:

$$\begin{pmatrix} k_{1,1}^{(1)} & k_{1,2}^{(1)} & \cdots & k_{1,n}^{(1)} \\ 0 & k_{2,2}^{(2)} & \cdots & k_{2,n}^{(1)} \\ \vdots & \vdots & & \vdots \\ 0 & k_{n,2}^{(2)} & \cdots & k_{n,n}^{(2)} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} f_1^{(1)} \\ f_2^{(2)} \\ \vdots \\ f_n^{(2)} \end{pmatrix}. \tag{5.5}$$

Step k

Let $1 \leq k \leq n-1$ and assume again a non-zero pivot in row k; $k_{k,k}^{(k)} \neq 0$. Furthermore, assume that $K^{(k)}\mathbf{u} = \mathbf{f}^{(k)}$ has been constructed such that the first $k-1$ columns of $K^{(k)}$ have 0 entries below the diagonal:

$$K^{(k)} = \begin{pmatrix} k_{1,1}^{(1)} & k_{1,2}^{(1)} & & \cdots & & k_{1,n}^{(1)} \\ 0 & k_{2,2}^{(2)} & & & & k_{2,n}^{(2)} \\ \vdots & & \ddots & & & \vdots \\ 0 & \cdots & 0 & k_{k,k}^{(k)} & \vdots & k_{k,n}^{(k)} \\ \vdots & & & \vdots & & \vdots \\ 0 & \cdots & 0 & k_{n,k}^{(1)} & \cdots & k_{n,n}^{(k)} \end{pmatrix}. \tag{5.6}$$

Defining the row multipliers

$$m_{i,k} = \frac{k_{i,k}^{(k)}}{k_{k,k}^{(k)}}, \quad i = k+1, \ldots, n, \tag{5.7}$$

allows the elimination of the non-zero entries below the diagonal in column k. Writing the row multipliers in matrix notation as

$$M_k = \begin{pmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots & & \vdots \\ \vdots & & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1,k} & 1 & \ddots & \vdots \\ \vdots & & \vdots & & \ddots & 0 \\ 0 & \cdots & -m_{n,k} & & & 1 \end{pmatrix}, \tag{5.8}$$

allows thee unknown u_k to be removed from equations $k+1, \ldots, n$ by multiplying the system of equations from the previous step with M_k:

$$K^{(k+1)}\mathbf{u} =: M_k K^{(k)}\mathbf{u} = M_k \mathbf{f}^{(k)} := \mathbf{f}^{(k+1)}. \tag{5.9}$$

By continuing in this manner, after $n-1$ steps the original matrix K has been reduced to upper triangular form such that the resulting system of equations can be written as

$$\begin{pmatrix} k_{1,1}^{(1)} & \cdots & \cdots & k_{1,n}^{(1)} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & k_{n,n}^{(n)} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} f_1^{(1)} \\ f_2^{(2)} \\ \vdots \\ f_n^{(n)} \end{pmatrix}, \tag{5.10}$$

which by construction can be written as:

$$M_{n-1}M_{n-2}\dots M_1 K\mathbf{u} = M_{n-1}M_{n-2}\dots M_1\mathbf{f}, \tag{5.11}$$

where each of the matrices M_i have ones on the diagonal and are lower triangular. Multiplying each side of Equation (5.11) with the product of the Gauss transforms yields the following system of equations which is equal to the original system $K\mathbf{u} = \mathbf{f}$:

$$(M_{n-1}M_{n-2}\dots M_1)^{-1}(M_{n-1}M_{n-2}\dots M_1 K)\,\mathbf{u} = \mathbf{f}. \tag{5.12}$$

By construction it holds that $M_{n-1}M_{n-2}\dots M_1 K$ is upper triangular hence it is denoted with U. On the other hand, the inverse of the product of Gauss transforms can be written as

$$(M_{n-1}M_{n-2}\dots M_1)^{-1} = M_1^{-1}\dots M_{n-2}^{-1}M_{n-1}^{-1}. \tag{5.13}$$

It can be shown by direct multiplication that the inverses M_i^{-1} are equal to M_i with all off-diagonals changed in sign. Therefore, the inverse of the product of Gauss transformations can be written as a product of lower triangular matrices which again is lower triangular, hence the factorisation from Equation (5.12) can be written as

$$K\mathbf{u} = (M_{n-1}M_{n-2}\dots M_1)^{-1}(M_{n-1}M_{n-2}\dots M_1 K)\,\mathbf{u} := LU\mathbf{u} = \mathbf{f}, \tag{5.14}$$

known as the *LU decomposition* where U is upper triangular and L lower triangular. In the next section it will be shown that decomposing the matrix in such form will prove extremely convenient in solving linear systems of equations. The LU decomposition exists whenever all submatrices of K are non-singular [2]. The LU decomposition can be written slightly different by decomposing the matrix U further into factors D and M^T, where $D = \mathrm{diag}(U)$ and $M^T = D^{-1}U$. Since the matrix D is diagonal, its inverse is easy to compute. The inverse D^{-1} exists since K is non-singular and thus has non-zero determinant. Furthermore, since left multiplication with a diagonal matrix only scales the rows, the matrix M^T is still upper triangular. The LU decomposition then becomes

$$K = LDM^T, \tag{5.15}$$

where L is lower triangular, D diagonal and M^T upper triangular. The decomposition from Equation (5.15) is referred to as the LDM decomposition.

Now consider a matrix K that is symmetric positive definite (SPD). Since all submatrices of a SPD matrix are non-singular, an LU decomposition exists. Writing this LU as a LDM decomposition it immediately follows from the symmetry of K that $L = M$. Hence the LDM decomposition for SPD matrices reduces to the *LDL decomposition*:

$$K = LDL^T. \tag{5.16}$$

Due to the positive definiteness of K, the (diagonal) entries of D are positive. Hence, it can be written as $D = \sqrt{D}\sqrt{D}$ with which the decomposition of K can be written as the *Cholesky decomposition*:

$$K = CC^T, \tag{5.17}$$

where C is lower triangular and $C = L\sqrt{D}$. The advantage of the Cholesky decomposition over the LU decomposition is twofold. Not only is the memory usage reduced by half with respect to the LU decomposition, but also the required number of computations to determine the decomposition is halved.

In the derivation of the LU decomposition, every step assumed that the corresponding pivot element was non-zero. This assumption can be omitted by starting each step with switching two rows as to put a non-zero element on the pivot location. However, some elements might be 0 in exact arithmetic but due to rounding errors become non-zero. Using such element as a pivot element can result in significant errors building up in the solution. Furthermore, if the entries of the matrix vary several orders of magnitude it is likely that large rounding errors will occur. Therefore, to prevent these problems from happening the next subsection introduces the concepts of *scaling* and *partial-*, *complete* pivoting.

5.2.1. Scaling & Pivoting

If the matrix entries of K vary several orders of magnitude it is possible that rounding errors occur due to the machine precision. The conditioning of the matrix can be improved by scaling the rows of the matrix. However, there is no a priori strategy to determine scaling factors such that the effects of rounding errors are always decreased. Empirical results have provided a possible approach which is to construct a diagonal scaling matrix D_{scal} such that the rows of $D_{scal}K$ all have approximately the same ∞-norm [2]. Not only is it possible to scale the rows of K, but also techniques exist to scale the columns of K in order to improve the conditioning of K [6, 9].

In Gauss elimination, every step assumed a non-zero pivot. In order to ommit this assumption and to avoid small pivot elements *partial pivoting* (row interchanges) will be applied prior to calculating the row multipliers as in Equation (5.7). To this extend let $1 \leq k \leq n-1$. In the k-th step of Gaussian elimination define

$$c_k = \max_{k \leq i \leq n} |k_{i,k}^{(k)}| . \tag{5.18}$$

Let i be the smallest row index such that the maximum for c_k is attained. If $i > k$ switch rows i and k in both A and $\mathbf{f}$. By construction it then holds that all row multipliers satisfy

$$|m_{i,k}| \leq 1 , \quad i = k+1, \ldots, n . \tag{5.19}$$

This property prevents the excessive growth of elements in $K^{(k)}$ and thus decreases the probability of rounding errors. Partial pivoting in step k of Gaussian elimination is typically denoted with a permutation matrix P_k such that after $n-1$ steps the system is given by

$$M_{n-1}P_{n-1}M_{n-2}P_{n-2}\ldots M_1P_1K\mathbf{u} = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\ldots M_1P_1\mathbf{f} . \tag{5.20}$$

Instead of only interchanging rows, it is also possible to switch both rows and columns which is referred to as *complete pivoting*. Let again $1 \leq k \leq n-1$. Similarly to partial pivot-

ing, in the k-th step of Gaussian elimination define

$$c_k = \max_{k \leq i,j \leq n} |k_{i,k}^{(k)}|. \tag{5.21}$$

Let i, j be the smallest row- and column indices such that the maximum for c_k is attained. If $i, j > k$ switch the rows of $A, \mathbf{f}$ and the columns of A are switched according to i and j. This ensures that the resulting row multipliers are minimal due c_k being the maximum of the remaining untransformed block matrix. As a result, complete pivoting results in a small growth factor of matrix elements decreasing the probability of rounding errors. It is important to note that the column switch implies a switch in variables. Once the resulting system of equations has been solved this has to be reversed in order to obtain the solution to the original problem. Since the computation cost of complete pivoting is higher than for partial pivoting, and the error behaviour is often the same for both methods, in many practical problems it is chosen to use partial pivoting.

When K is SPD (and hence its Cholesky decomposition exists) the use of scaling or pivoting is not necessary [2].

5.3. Forward and back substitution

In the previous section it was shown how Gaussian elimination can be applied to write a matrix K as a product of a lower- and upper triangular matrix. These favourable properties can be used to efficiently solve the system of equations. By introducing an auxiliary vector $\mathbf{y}$ of same size as $\mathbf{f}$, the system of equations $LU\mathbf{u} = \mathbf{f}$ can be written as:

$$L\mathbf{y} = \mathbf{f}, \tag{5.22}$$

$$U\mathbf{u} = \mathbf{y}. \tag{5.23}$$

Due to the triangular properties of L and U, Equations (5.22), (5.23) are efficiently solvable by *forward* and *backward substitution* respectively. Introducing the notation $A_{i,1:i} := [A_{i,1}, \ldots, A_{i,i}]$,the two general algorithms for determining $\mathbf{y}$ and $\mathbf{u}$ are given in Algorithms 2 and 3.

Algorithm 2 Forward substitution

for $i = 1, \ldots, n$ **do**
  $y_i = \left[f_i - L_{i,1:i-1} \cdot y_{1:i-1}\right] / L_{i,i}$
end for

Algorithm 3 Backward substitution

for $i = n, \ldots, 1$ **do**
  $u_i = \left[y_i - U_{i,i+1:n} \cdot u_{i+1:n}\right] / U_{i,i}$
end for

Note that by construction it holds that $L_{i,i} = 1$ for every $i = 1, \ldots, n$. The same algorithms can be used for a Cholesky factorisation since in that case $U = L^T$.

It can be shown that in the presence of round-off errors solving the linear equations $L\mathbf{y} = \mathbf{f}$ obtains the solution vector $\hat{\mathbf{y}}$ such that

$$(L + F)\,\hat{\mathbf{y}} = \mathbf{f}, \tag{5.24}$$

where F is a perturbation matrix to L as a result of round-off errors. On most modern computers with double precision it can be shown that the entries of F are relatively small compared to those of L. This implies that the forward substitution from Equation (5.22) is numerically stable. For a proof of the numerical stability the reader is referred to Higham [11]. Analogously it can be proven that the back substitution from Equation (5.23) is also numerically stable.

5.4. Implementation: Parallel Sparse Direct Solver

Many different implementations of the previously mentioned decompositions and solution methods exists. Arguably the most well-known, and also available within *DIANA*, is Intel's *Parallel Sparse Direct Solver* (PARDISO), which is a "high-performance, robust, memory-efficient and easy to use solver for large and sparse (non)symmetric linear systems" [1]. PARDISO calculates the solution to a system of sparse linear equations $K\mathbf{u} = \mathbf{f}$ using a highly optimized parallel LU, LDL^T or CC^T decomposition. PARDISO is hand-optimized to run on Intel hardware. For sufficiently large problems, numerical experiments have demonstrated an almost linear relation between computational time and number of parallel cores with a speed-up of factor 7 using eight processors being observed.

6

Iterative methods for linear systems

The previous chapter described direct solution methods for solving linear systems of equations. The most important downside of direct methods is the storage requirement especially for large three-dimensional problems. For this reason, iterative methods provide an alternative since these methods do not require the computation and storage of the decomposition of a matrix. Two different categories of iterative methods for solving linear systems exist: basic iterative methods and Krylov subspace methods. The next two sections will elaborate on both these categories.

6.1. Basic iterative methods

The fundamental idea of iterative methods to solve a linear system $K\mathbf{u} = \mathbf{f}$ is to construct a sequence of successive approximations $\{\mathbf{u}_k\}$, where

$$\lim_{k\to\infty} \mathbf{u}_k = \mathbf{u}\,, \tag{6.1}$$

with $\mathbf{u}$ the exact solution. Clearly it is impossible to create an infinite sequence of approximations, hence the iterative scheme is terminated once a sufficiently accurate solution is found. To construct a basic iterative scheme a matrix splitting of K is constructed

$$K = P - R\,, \tag{6.2}$$

in which $P \in \mathbb{R}^{n\times n}$ is non-singular. Using the splitting of Equation (6.2) then allows to rewrite the linear system as $P\mathbf{u} = R\mathbf{u} + \mathbf{f}$. By multiplying both sides of the equation with

P^{-1} an iterative scheme can be derived as follows

$$\begin{aligned}\mathbf{u}_{k+1} &= P^{-1}R\mathbf{u}_k + P^{-1}\mathbf{f} \\ &= P^{-1}(P-K)\mathbf{u}_k + P^{-1}\mathbf{f} \\ &= \left(I - P^{-1}K\right)\mathbf{u}_k + P^{-1}\mathbf{f} \qquad (6.3) \\ &= \mathbf{u}_k + P^{-1}(\mathbf{f} - K\mathbf{u}_k) \\ &= \mathbf{u}_k + P^{-1}\mathbf{r}_k \qquad (6.4)\end{aligned}$$

where $\mathbf{r}_k$ is the residual vector of iteration k and $\mathbf{u}_0$ an initial guess. Since each iteration requires a linear system solve with the matrix P, the matrix P should be chosen such that this operation is as cheap as possible. This is the case when P is either a diagonal-, upper- or lower triangular matrix.

The iteration from Equation (6.3) can be seen as a technique for solving

$$\left[I - \left(I - P^{-1}K\right)\right]\mathbf{u} = P^{-1}\mathbf{f},$$

which can be rewritten as

$$P^{-1}K\mathbf{u} = P^{-1}\mathbf{f}. \qquad (6.5)$$

From this formulation it is clear that the matrix P serves as a preconditioner to the original problem.

Some of the most well-known choices for preconditioners P are given below. Here D denotes a matrix of equal size to K with K's diagonal elements and zeros elsewhere, L and U respectively contain all strictly lower- and upper-diagonal elements of K and zeros elsewhere.

$$\begin{aligned}\text{Jacobi:} \quad & P = D \\ \text{forward Gauss-Seidel:} \quad & P = D + L \\ \text{backward Gauss-Seidel:} \quad & P = D + U \\ \omega\text{-damped Jacobi} \quad & P = \frac{1}{\omega}D \\ \text{SOR}(\omega): \quad & P = \frac{1}{\omega}D + L \\ \text{SSOR}(\omega): \quad & P = \frac{\omega}{\omega(2-\omega)}\left(\frac{1}{\omega}D + L\right)D^{-1}\left(\frac{1}{\omega}D + U\right)\end{aligned}$$

As mentioned earlier, successive approximations are calculated until the approximation is sufficiently accurate. To determine when the algorithm stops, a stopping criterion has to be defined. In a good choice of stopping criterion one aims at balancing the imposed accuracy of the approximation with the required computational cost. A commonly used stopping criterion is to iterate until

$$\frac{\|\mathbf{r_k}\|}{\|\mathbf{f}\|} \leq \varepsilon \qquad (6.6)$$

for some ε. This stopping criterion is scaling invariant and the required accuracy does not depend on the initial guess. Higham [11] provides a detailed elaboration on this and more preferable stopping criterion.

6.2. Krylov subspace methods

In basic iterative methods from the previous section the solutions are calculated with the recursion

$$\mathbf{u}_{k+1} = \mathbf{u}_k + P^{-1}\mathbf{r}_k\,.$$

Writing out the first few iterations of this recursion gives

$$\begin{aligned}
&\mathbf{u}_0\,,\\
&\mathbf{u}_1 = \mathbf{u}_0 + P^{-1}\mathbf{r}_0\,,\\
&\mathbf{u}_2 = \mathbf{u}_1 + P^{-1}\mathbf{r}_1 = \mathbf{u}_0 + P^{-1}\mathbf{r}_0 + P^{-1}\left(\mathbf{f} - K\mathbf{u}_1\right)\\
&\quad = \mathbf{u}_0 + P^{-1}\mathbf{r}_0 + P^{-1}\left(\mathbf{f} - K\left[\mathbf{u}_0 + P^{-1}\mathbf{r}_0\right]\right)\\
&\quad = \mathbf{u}_0 + 2P^{-1}\mathbf{r}_0 - P^{-1}KP^{-1}\mathbf{r}_0\,,\\
&\quad \vdots
\end{aligned}$$

From this it follows that the solutions of basic iterative methods belong to the following subspace

$$\mathbf{u}_i \in \mathbf{u}_0 + \operatorname{span}\left\{P^{-1}\mathbf{r}_0, \left(P^{-1}K\right)\left(P^{-1}\mathbf{r}_0\right), \ldots, \left(P^{-1}K\right)^{i-1}\left(P^{-1}\mathbf{r}_0\right)\right\}.$$

These types of subspaces of the form $\mathcal{K}^i\left(K;\mathbf{r}_0\right) := \operatorname{span}\left\{\mathbf{r}_0, K\mathbf{r}_0, \ldots, K^{i-1}\mathbf{r}_0\right\}$ are called *Krylov subspaces* of dimension i corresponding to the matrix K and initial residual $\mathbf{r}_0$. The basic iterative methods thus find an i-th approximate solution in the Krylov subspace $\mathbf{u}_0 + \mathcal{K}^i\left(P^{-1}K; P^{-1}\mathbf{r}_0\right)$.

Krylov subspace methods are popular for solving large systems of linear equations. The idea of these methods (and projection methods in general) is to extract an approximate solution to the problem from a subspace $\mathcal{K}^n$ of $\mathbb{R}^n$. For Krylov subspace methods the subspace $\mathcal{K}^n$ is a Krylov subspace. If $\mathcal{K}^m$ has dimension m, then m constraints must be imposed to be able to find such an approximation. Typically, these constraints are imposed as independent orthogonality conditions meaning that the residual $\mathbf{f} - K\mathbf{u}$ is constrained to be orthogonal to m linear independent vectors. These constraints define another subspace $\mathcal{L}$ of dimension m [16].

The above procedure can be formulated as

$$\text{Find} \quad \mathbf{u}_m \in \mathbf{u}_0 + \mathcal{K}^m \quad \text{such that} \quad \mathbf{f} - K\,\mathbf{u}_m \perp \mathcal{L}^m\,, \tag{6.7}$$

with $\mathbf{u}_0$ an arbitrary initial guess.

For Krylov subspace methods the subspace $\mathcal{K}^m$ is the Krylov subspace

$$\mathcal{K}^m\left(K, \mathbf{r}_0\right) = \operatorname{span}\left\{\mathbf{r}_0, K\mathbf{r}_0, \ldots, K^{m-1}\mathbf{r}_0\right\}, \tag{6.8}$$

where a preconditioner P has been left out for simplicity.

Assuming symmetry of the matrix K, *Lanczos algorithm* can be used to construct an orthogonal basis $\{\mathbf{v}_1,\dots,\mathbf{v}_m\}$ for $\mathcal{K}^m$ as follows.

Algorithm 4 Lanczos Algorithm

1: Choose a vector $\mathbf{v}_1$, such that $\|\mathbf{v}_1\|_2 = 1$
2: Set $\beta_1 = 0$, $\mathbf{v}_0 = 0$
3: **for** $j = 1,\dots,m$ **do**
4: $\quad \mathbf{w}_j = K\mathbf{v}_j - \beta_j \mathbf{v}_{j-1}$
5: $\quad \alpha_j = (\mathbf{w}_j, \mathbf{v}_j)$
6: $\quad \mathbf{w}_j = \mathbf{w}_j - \alpha_j \mathbf{v}_j$
7: $\quad \beta_{j+1} = \|\mathbf{w}_j\|_2$
8: $\quad$ **if** $\beta_{j+1} = 0$ **then**
9: $\quad\quad$ Stop
10: $\quad$ **else**
11: $\quad\quad \mathbf{v}_{j+1} = \mathbf{w}_j / \beta_{j+1}$
12: $\quad$ **end if**
13: **end for**

In exact arithmetic, Lanczos algorithm ensures that the vectors $\mathbf{v}_i$ are orthogonal. In reality, however, orthogonality is lost rapidly after a few iterations due to rounding errors. To address this issue, many improvements have been developed to reduce the occurrence of orthogonality. An overview of some of these improved orthogonalization schemes can be found in Saad [16].

The Lanczos is a short recurrence algorithm meaning that only a few vectors have to be stored. To illustrate this steps $11, 6, 4$ can be combined to obtain

$$\beta_{j+1}\mathbf{v}_{j+1} = K\mathbf{v}_j - \beta_j \mathbf{v}_{j-1} - \alpha_j \mathbf{v}_j \,. \tag{6.9}$$

Rearranging terms yields

$$K\mathbf{v}_j = \beta_j \mathbf{v}_{j-1} + \alpha_j \mathbf{v}_j + \beta_{j+1}\mathbf{v}_{j+1}\,, \quad j = 1,\dots,m\,. \tag{6.10}$$

Writing $V_m = [\mathbf{v}_1,\dots,\mathbf{v}_m]$ and

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & 0 \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_m \\ 0 & & & \beta_{m+1} \end{pmatrix} \tag{6.11}$$

allows Equation (6.10) to be rewritten as

$$KV_m = V_m T_m \,. \tag{6.12}$$

Multiplication with V_m^T and using the orthogonality of the vectors $\mathbf{v}_i$ then yields the expression

$$V_m^T K V_m = T_m \,. \tag{6.13}$$

Since the matrix T_m is tridiagonal, only two additional vectors have to be stored in order to be able to compute the next vector. This is a significant advantage over the analogous Arnoldi's algorithm for the unsymmetrical case which is characterized by long recurrences. For a description on the unsymmetrical case and Arnoldi's algorithm the reader is referred to Saad [16].

One of the most popular Krylov subspace methods for solving large sparse SPD linear systems is the *Conjugate Gradient* (*CG*) method. The next section will provide an intuitive introduction to *CG*.

6.2.1. CG

The Conjugate Gradient method is an iterative technique to solve large sparse SPD linear systems by orthogonalizing the residuals with respect to each other. An intuitive way to derive the *CG* method is to first consider the *steepest descent* method. To this extend consider the problem of minimizing the function

$$\phi(\mathbf{u}) = \frac{1}{2}\mathbf{u}^T K\mathbf{u} - \mathbf{u}^T\mathbf{f}.$$

The minimum of $\phi(\mathbf{u})$ is attained by $\mathbf{u} = K^{-1}\mathbf{f}$ hence the above minimization is equivalent to solving the system $K\mathbf{u} = \mathbf{f}$.

At a current point $\mathbf{u}_k$ the function $\phi(\mathbf{u}_k)$ decreases fastest in the direction of the negative gradient $-\nabla\phi(\mathbf{u}_k)$ which can be shown to be equal to the residual [9]:

$$-\nabla\phi(\mathbf{u}_k) = \mathbf{f} - K\mathbf{u}_k = \mathbf{r}_k \,. \tag{6.14}$$

To choose a new approximation along the line of steepest descent, $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha\mathbf{r}_k$, the value α has to be determined such that the function ϕ is minimized along that line. Hence the value of α can be found by solving:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\phi(\mathbf{u}_{k+1}) = 0$$

Expanding the left-hand side we obtain:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\alpha}\phi(\mathbf{u}_{k+1}) &= \frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\frac{1}{2}\left(\mathbf{u}_k + \alpha\mathbf{r}_k\right)^T K\left(\mathbf{u}_k + \alpha\mathbf{r}_k\right) - \left(\mathbf{u}_k + \alpha\mathbf{r}_k\right)^T\mathbf{f}\right) \\
&= \frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\frac{1}{2}\mathbf{u}_k^T K\mathbf{u}_k + \alpha\mathbf{r}_k^T K\mathbf{u}_k + \frac{1}{2}\alpha^2\mathbf{r}_k^T K\mathbf{r}_k - \mathbf{u}_k^T\mathbf{f} - \alpha\mathbf{r}_k^T\mathbf{f}\right) \\
&= \mathbf{r}_k^T K\mathbf{u}_k + \alpha\mathbf{r}_k^T K\mathbf{r}_k - \mathbf{r}_k^T\mathbf{f} \\
&= 0
\end{aligned}$$

Solving for α then yields:

$$\alpha = \frac{\mathbf{r}_k^T\mathbf{f} - \mathbf{r}_k^T K\mathbf{u}_k}{\mathbf{r}_k^T K\mathbf{r}_k} = \frac{\mathbf{r}_k^T(\mathbf{f} - K\mathbf{u}_k)}{\mathbf{r}_k^T K\mathbf{r}_k} = \frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{r}_k^T K\mathbf{r}_k}.$$

The above framework describes the method of steepest descents. In this method it is possible that the gradient directions are not different enough during iterations and the approximation jumps back-and-forth between points. To avoid this, the *CG* method considers minimization of ϕ along a set of different directions that do not necessarily correspond to the residuals as in Equation (6.14). Instead, the new approximation is constructed with a new set of search directions $\{\mathbf{p}_1, \mathbf{p}_2, \ldots\}$:

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha\mathbf{p}_k. \tag{6.15}$$

For this choice it can be shown, similarly to the derivation above, that the value α for which the function along the new search direction is minimized is given by

$$\alpha = \alpha_k = \frac{\mathbf{p}_k^T\mathbf{r}_k}{\mathbf{p}_k^T K\mathbf{p}_k}. \tag{6.16}$$

The goal then is to find the search directions $\mathbf{p}_i$ in a way that guarantees convergence. Substitution of the choice of α into the function ϕ then allows to write

$$\phi(\mathbf{u}_{k+1}) = \phi(\mathbf{u}_k + \alpha\mathbf{p}_k) = \phi(\mathbf{u}_k) - \frac{1}{2}\frac{\left(\mathbf{p}_k^T\mathbf{r}_k\right)^2}{\mathbf{p}_k^T K\mathbf{p}_k}, \tag{6.17}$$

from which it follows that we want to choose the search direction $\mathbf{p}_k$ such that it is not orthogonal to the residual $\mathbf{r}_k$. Because $\mathbf{u}_{k+1} \in \mathbf{u}_0 + \text{span}\{\mathbf{p}_1, \ldots, \mathbf{p}_k\}$ and that ϕ is being minimized over $\mathbb{R}^k$ it follows that convergence is guaranteed in at most n steps. However, for this to be viable it must hold that $\mathbf{u}_k$ is 'easy' to compute from $\mathbf{u}_{k-1}$.

Writing out Equation (6.15) obtains

$$\begin{aligned}
\mathbf{u}_{k+1} &= \mathbf{u}_k + \alpha\mathbf{p}_k \\
&\vdots \\
&= \mathbf{u}_0 + \sum_{i=1}^{k-1} y_i\mathbf{p}_i + \alpha\mathbf{p}_k \\
&:= \mathbf{u}_0 + P_{k-1}\mathbf{y} + \alpha\mathbf{p}_k
\end{aligned}$$

where the y_i correspond to the α's from previous iterations and $P_{k-1} = [\mathbf{p}_1, \ldots, \mathbf{p}_{k-1}]$. Substitution of this expression into ϕ and rearranging terms then yields

$$\phi(\mathbf{u}_k) = \alpha\mathbf{y}^T P_{k-1}^T K\mathbf{p}_k + \frac{1}{2}\alpha^2\mathbf{p}_k^T K\mathbf{p}_k - \alpha\mathbf{p}_k^T\mathbf{r}_0. \tag{6.18}$$

Choosing the search direction $\mathbf{p}_k$ K-conjugate to all other previous search directions, i.e. $\mathbf{p}_k \in \text{span}\{K\mathbf{p}_1, \ldots, K\mathbf{p}_{k-1}\}^\perp$, it can be shown that the original minimization reduces to two

uncoupled minimizations for α and $\mathbf{y}$ where the minimization for α is minimized by the expression found in Equation (6.16). To see that this choice for α is well-defined observe the following. The matrix $P_k^T K P_k$ is a diagonal matrix, and because K is positive definite and the search directions are non-zero it is non-singular.

Combining both steepest descent and the above K-conjugate searching, the search vector $\mathbf{p}_k$ can be chosen such that it is closest (in the 2-norm sense) to $\mathbf{r}_k$ while still being k-conjugate to all previous search directions:

$$\mathbf{p}_k = \min_{\mathbf{p} \in \text{span}\left\{K\mathbf{p}_1,\ldots,K\mathbf{p}_{k-1}\right\}^{\perp}} ||\mathbf{p} - \mathbf{r}_k||_2 \,. \tag{6.19}$$

For search directions constructed according to the above minimization, the following theorem and corollary hold.

Theorem 1. *After k iterations we have*

$$\begin{aligned} \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k K \mathbf{p}_k \\ P_k^T \mathbf{r}_k &= 0 \\ \text{span}\left\{\mathbf{p}_1,\ldots,\mathbf{p}_k\right\} &= \text{span}\left\{\mathbf{r}_1,\ldots,\mathbf{r}_k\right\} = \mathcal{K}_k(K;\mathbf{r}_0) \end{aligned}$$

and the residuals $\mathbf{r}_0,\ldots,\mathbf{r}_k$ *are mutually orthogonal.*

Corollary 1.1. *The residuals and search directions have the property for* $k \geq 2$*:*

$$\mathbf{p}_k \in \text{span}\left\{\mathbf{p}_{k-1}, \mathbf{r}_{k-1}\right\}$$

For the proofs the reader is referred to [9]. The above theorem and corollary can be used to find a more efficient expression for α and a more simple recursion for the search direction:

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_{k1} \,, \tag{6.20}$$

where

$$\beta_k = \mathbf{r}_{k+1}^T \mathbf{r}_{k+1} / \mathbf{r}_k^T \mathbf{r}_k \,. \tag{6.21}$$

The general form of the conjugate gradient method then follows as given in Algorithm 5.

Algorithm 5 Conjugate Gradient algorithm

1: Set $\mathbf{r}_0 = \mathbf{f} - K\mathbf{u}_0$, $\mathbf{p}_0 = \mathbf{r}_0$.
2: $\mathbf{r}_k \neq \mathbf{0}$
3: **for** $k = 0, 1, \ldots$ **do**
4: $\quad \alpha_k = \mathbf{r}_k^T \mathbf{r}_k / \mathbf{p}_k^T K \mathbf{p}_k$
5: $\quad \mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{p}_k$
6: $\quad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k K \mathbf{p}_k$
7: $\quad \beta_k = \mathbf{r}_{k+1}^T \mathbf{r}_{k+1} / \mathbf{r}_k^T \mathbf{r}_k$
8: $\quad \mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
9: **end for**

A well-known error-bound is known for approximations obtained with the *CG* method [16]. To this extend, we define the condition number of a matrix K, which for a SPD matrix is defined the ratio of the extreme eigenvalues $\kappa := \kappa_2(K) = \frac{\lambda_{\max}}{\lambda_{\min}}$. The error of approximation in iteration k is then bounded by :

$$\|\mathbf{u} - \mathbf{u}_k\|_K \leq 2\|\mathbf{u} - \mathbf{u}_0\|_K \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k. \tag{6.22}$$

6.2.2. Preconditioning

As can be seen from Equation (6.22) the convergence of *CG* is highly dependent on the extreme eigenvalues of the linear operator K. Therefore, the linear system can be preconditioned to obtain more favourable extreme eigenvalues. The preconditioned stiffness matrix leads to the linear system of Equation (6.5). Ideally, the preconditioning matrix P is chosen such that the eigenvalues of the preconditioned system are clustered around 1 which implies very fast convergence of *CG*. However, since the preconditioned system is evaluated every iteration the preconditioning must be cheap to compute and inexpensive to apply to the linear system [12]. Therefore, in general some approximate form of the stiffness matrix is used, as described in Section 6.1.

In [12] an algorithm is presented for preconditioned *CG*, which adds only one line to the original algorithm as can be seen in Algorithm 6.

Algorithm 6 Preconditioned Conjugate Gradient algorithm

Set $\mathbf{r}_0 = \mathbf{f} - K\mathbf{u}_0$, $\mathbf{z}_0 = P^{-1}\mathbf{r}_0$, $\mathbf{p}_0 = \mathbf{z}_0$.
$\mathbf{r}_k \neq \mathbf{0}$
for $k = 0, 1, \ldots$ **do**
  $\alpha_k = \mathbf{r}_k^T \mathbf{z}_k / \mathbf{p}_k^T K \mathbf{p}_k$
  $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k K \mathbf{p}_k$
  $\mathbf{z}_{k+1} = P^{-1}\mathbf{r}_{k+1}$
  $\beta_k = \mathbf{r}_{k+1}^T \mathbf{z}_{k+1} / \mathbf{r}_k^T \mathbf{z}_k$
  $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$
end for

In SLA, the stiffness matrix is updated every iteration with a low-rank update. Therefore, a possible choice for a preconditioner could be the stiffness matrix. To this extend, the first iteration should be solved using a direct solution method. This implies that a decomposition of the stiffness matrix is known such that the application of the preconditioner is inexpensive to apply. Since the preconditioner is now an approximation of the stiffness matrix, especially for low-rank updated matrices, many of the eigenvalues will be equal to 1, speeding up *CG*. In particular, after a rank-k update this preconditioner will lead to $n-k$ of the eigenvalues equal to 1 [19]. As a result, *CG* only takes $k+1$ iterations.

6.3. Deflation

It is known that the smallest eigenvalues correspond to the slow converging components of the solution [12]. As a result, in situations where there are many small eigenvalues, such as in composite materials with strongly varying properties, the *CG* method may require a significant number of iterations until convergence. Therefore, it is desirable to remove these small eigenvalues from the problem. To this extend, the method of deflation can be used.

Deflation is a technique to improve iterative methods by splitting the solution in a part that is to be computed directly and a part that is computed iteratively by solving a *deflated* system [7, 14]. It is proven to be successful for systems with a few isolated extreme eigenvalues.

To construct the splitting of the solution, recall the system of interest $K\mathbf{u} = \mathbf{f}$ where K is a SPD matrix. Defining the deflation space $Z \in \mathbb{R}^{n\times m}$ that is to be projected out of the residual, the projection P can be constructed as follows

$$P = I - KZ\left(Z^T KZ\right)^{-1} Z^T. \tag{6.23}$$

Defining

$$A_c = Z^T KZ \tag{6.24}$$

such that $A_c \in \mathbb{R}^{m\times m}$ allows the solution $\mathbf{u}$ to be written as

$$\begin{aligned}
\mathbf{u} &= \left(I - P^T\right)\mathbf{u} + P^T\mathbf{u} \\
&= \left(I - \left[I - KZ\left(Z^T KZ\right)^{-1} Z^T\right]^T\right)\mathbf{u} + P^T\mathbf{u} \\
&= \left(I - \left[I - KZA_c^{-1} Z^T\right]^T\right)\mathbf{u} + P^T\mathbf{u} \\
&= \left(I - \left[I - ZA_c^{-1} Z^T K\right]\right)\mathbf{u} + P^T\mathbf{u} \\
&= ZA_c^{-1} Z^T K\mathbf{u} + P^T\mathbf{u} \\
&= ZA_c^{-1} Z^T\mathbf{f} + P^T\mathbf{u}.
\end{aligned} \tag{6.25}$$

In Equation (6.25) the solution $\mathbf{u}$ has been written as the sum of two terms of which only one is a function of $\mathbf{u}$. Now assume $m \ll n$ and that Z is of rank m and SPD. The matrix A_c is then easily computed and factored due to its relatively small size. A back and forward substitution then suffices to calculate A_c^{-1}.

To calculate the right-hand side term $P^T\mathbf{u}$ note first that the following equality holds.

$$\begin{aligned}
KP^T &= K\left(I - KZ\left(Z^T KZ\right)^{-1} Z^T\right)^T \\
&= K\left(I - Z\left(Z^T KZ\right)^{-1} Z^T K\right) \\
&= \left(I - KZ\left(Z^T KZ\right)^{-1} Z^T\right)K \\
&= PK
\end{aligned} \tag{6.26}$$

With this equality, it suffices to calculate the solution $\tilde{\mathbf{u}}$ of the *deflated* system:

$$(KP^T\tilde{\mathbf{u}} =)PK\tilde{\mathbf{u}} = P\mathbf{f} \tag{6.27}$$

with CG and subsequently multiplying the solution $\tilde{\mathbf{u}}$ with P^T to obtain $P^T\tilde{\mathbf{u}}$. The notation $\tilde{\mathbf{u}}$ has been used here to distinguish the solution of the original problem from the deflated system.

The projection P results in an singular system in Equation (6.27). This does not necessarily have to be an issue for CG, as long as the system is consistent (i.e., $P\mathbf{f}$ is in the range of PK)[17]. Now, since the same projection P is applied to both sides of the original system it still holds that $PK\tilde{\mathbf{u}} = P\mathbf{f}$ for some $\tilde{\mathbf{u}}$.

Note that if the complete space is deflated, $Z = I$, Equation (6.25) reduces to $\mathbf{u} = K^{-1}\mathbf{f}$. This implies that, for this choice, the deflation method effectively becomes a direct solution method.

6.3.1. Choices for deflation subspace

The previous derivation of deflation assumed a general deflation subspace Z of order m. Several choices for Z exist, of which three will be discussed now.

Eigenvector deflation
The most generic possibility is to choose the Z as the span of the eigenvectors of the smallest eigenvalues. Not only are the eigenvalues usually not readily known, but also the order of the deflation subspace grows when a large amount of eigenvalues are deflated out.

Subdomain deflation
Another possibility is to choose Z as a subdomain deflation. Diving the geometry Ω in n_{el} non-overlapping domains Ω_j the deflation subspace can be chosen as

$$Z_{ij} = \begin{cases} 1 & \text{if } i \in \Omega_j \\ 0 & \text{else} \end{cases} \tag{6.28}$$

resulting in a deflation subspace or order n_{el}, $Z \in \mathbb{R}^{n \times n_{el}}$.

Rigid body modes deflation
A third possibility is to deflate the rigid body modes. The idea of rigid body modes deflation is to consider collections of elements as rigid bodies. Due to the (possibly) large jumps in properties between these collections, the stiffness matrix K has similar discontinuities in the coefficients making it ill-conditioned. Furthermore, since the collections of elements are approximated as rigid bodies, the resulting eigenvalues are close to zero. Rigid body modes deflation aims at removing these small eigenvalues, which correspond to the slow converging components of the solution, from the system. From [12] it is known that the rigid body modes of a finite element are spanned by the kernel base vectors of the corresponding element stiffness matrix.

To illustrate how rigid body modes can be constructed, consider a problem consisting of a rigid material inside of a significantly less rigid material. The stiffness matrix K of the problem can then be split as $K = C + R$, where C is the singular submatrix corresponding to the rigid material and R the submatrix corresponding to the less rigid material. The deflation subspace is then chosen as the null space of C: $\text{null}(C) = \text{span}\{Z\}$. For three dimension problems, a stiffness matrix has 6 rigid body motions, 3 translational and 3 rotational. Hence Z consists of 6 basis vectors of the null space of C which corresponds to all six rigid body modes.

A fast and cheap solution for the computation of these rigid body modes is presented in [12]. Assuming that the rigid material consists of a single 4 noded tetrahedral element, the coordinate vector of the element can be given by

$$\mathbf{x} = \{x_1 \ y_1 \ z_1 \ \dots \ x_4 \ y_4 \ z_4\}^T.$$

A trivial choice to obtain three orthogonal translations is to choose the x, y and z directions to obtain the translational vectors:

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

To calculate the rotational rigid body modes, a transformation is used to a spherical coordinate system. In the new coordinate system, expressions can then be found for the three rotations along the trivial planes. For a derivation the reader is referred to [12].

Here the rigid body modes are given for a single element. To obtain the rigid body modes of a collection of elements, the rigid body modes of the individual elements can be assembled, taking into account the multiplicity of those degrees of freedom that lie in multiple elements.

7

Low-rank matrix update

In this section mathematical techniques will be discussed that present a numerically cheap way of dealing with low-rank matrix updates. In this context, a low-rank update to a matrix represents a small number of changes to the elements relative to the total number of matrix elements. In Section 7.1 a formula will be presented which allows cheap computation of the inverse of a rank-1 updated matrix. Section 7.2 then presents a generalization to this formula for the computation of the inverse of a k-rank updated matrices for arbitrary values k.

7.1. Sherman-Morrison formula

The Sherman-Morrison formula presents a method for computing the inverse of a rank-1 updated matrix. The statement of the formula is as follows.

Theorem 2 (Sherman-Morrison Formula)**.** *Let $A \in \mathbb{R}^{n\times n}$ an invertible square matrix and $\mathbf{u},\mathbf{v} \in \mathbb{R}^n$ column vectors. Then $A+\mathbf{u}\mathbf{v}^T$ is invertible, $\left(A+\mathbf{u}\mathbf{v}^T\right)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^T A^{-1}}{1+\mathbf{v}^T A^{-1}\mathbf{u}}$ if and only if $1+\mathbf{v}^T A^{-1}\mathbf{u} \neq 0$*

This formula allows for the computation of the rank-1 updated matrix $A+\mathbf{u}\mathbf{v}^T$. Since the inverse A^{-1} is usually already known, this formula presents a computationally cheap way of determining the inverse of the updated matrix without having to perform the computationally intensive inverse operation. The proof of Theorem 2 is given below.

Proof. ($\Longleftarrow$): To prove the backward direction assume that $1+\mathbf{v}^T A^{-1}\mathbf{u} \neq 0$. It remains to be proven that the matrix $A+\mathbf{u}\mathbf{v}^T$ is invertible. This implies that there should exist some matrix B such that $(A+\mathbf{u}\mathbf{v}^T)B = B(A+\mathbf{u}\mathbf{v}^T) = I$. It will be shown that $B = A^{-1} - \frac{A^{-1}\mathbf{u}\mathbf{v}^T A^{-1}}{1+\mathbf{v}^T A^{-1}\mathbf{u}}$

satisfies this condition by substitution:

$$\begin{aligned}\left(A+\mathbf{u}\mathbf{v}^T\right)\left(A^{-1}-\frac{A^{-1}\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\right) &= AA^{-1}-\frac{AA^{-1}\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}+\mathbf{u}\mathbf{v}^TA^{-1}-\frac{\mathbf{u}\left(\mathbf{v}^TA^{-1}\mathbf{u}\right)\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\\ &= I-\frac{\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}+\mathbf{u}\mathbf{v}^TA^{-1}-\frac{\left(\mathbf{v}^TA^{-1}\mathbf{u}\right)\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\\ &= I+\left(-\frac{1}{1+\mathbf{v}^TA^{-1}\mathbf{u}}+1-\frac{\mathbf{v}^TA^{-1}\mathbf{u}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\right)\mathbf{u}\mathbf{v}^TA^{-1}\\ &= I+\left(-\frac{1+\mathbf{v}^TA^{-1}\mathbf{u}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}+1\right)\mathbf{u}\mathbf{v}^TA^{-1}=I\end{aligned}$$

where in the second equality it was used that $\mathbf{v}^TA^{-1}\mathbf{u}$ is a scalar. In the last equality the assumption was used that $1+\mathbf{v}^TA^{-1}\mathbf{u}\neq 0$ such that the fraction is well defined.

$$\begin{aligned}\left(A^{-1}-\frac{A^{-1}\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\right)\left(A+\mathbf{u}\mathbf{v}^T\right) &= A^{-1}A+A^{-1}\mathbf{u}\mathbf{v}^T-\frac{A^{-1}\mathbf{u}\mathbf{v}^TA^{-1}A}{1+\mathbf{v}^TA^{-1}\mathbf{u}}-\frac{A^{-1}\mathbf{u}\left(\mathbf{v}^TA^{-1}\mathbf{u}\right)\mathbf{v}^T}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\\ &= I+A^{-1}\mathbf{u}\mathbf{v}^T-\frac{A^{-1}\mathbf{u}\mathbf{v}^T}{1+\mathbf{v}^TA^{-1}\mathbf{u}}-\frac{\left(\mathbf{v}^TA^{-1}\mathbf{u}\right)A^{-1}\mathbf{u}\mathbf{v}^T}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\\ &= I+\left(1-\frac{1}{1+\mathbf{v}^TA^{-1}\mathbf{u}}-\frac{\mathbf{v}^TA^{-1}\mathbf{u}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\right)A^{-1}\mathbf{u}\mathbf{v}^T\\ &= I+\left(1-\frac{1+\mathbf{v}^TA^{-1}\mathbf{u}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}\right)A^{-1}\mathbf{u}\mathbf{v}^T=I\end{aligned}$$

where again it was used that in the second equality that $\mathbf{v}^TA^{-1}\mathbf{u}$ is a scalar, and in the last equation the assumption that $1+\mathbf{v}^TA^{-1}\mathbf{u}\neq 0$ was used such that the fraction is well defined. This proves that $A+\mathbf{u}\mathbf{v}^T$ is invertible with $\left(A+\mathbf{u}\mathbf{v}^T\right)^{-1}=A^{-1}-\frac{A^{-1}\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}$.

($\Longrightarrow$): To prove the forward direction assume that A and $A+\mathbf{u}\mathbf{v}^T$ are invertible and let $\left(A+\mathbf{u}\mathbf{v}^T\right)^{-1}=A^{-1}-\frac{A^{-1}\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}$. Furthermore, assume $\mathbf{u}\neq\mathbf{0}$ otherwise the statement is trivial. Then:

$$\left(A+\mathbf{u}\mathbf{v}^T\right)A^{-1}u=\mathbf{u}+\mathbf{u}\left(\mathbf{v}^TA^{-1}\mathbf{u}\right)=\left(1+\mathbf{v}^TA^{-1}\mathbf{u}\right)\mathbf{u}\,.$$

Since the product of two invertible identities is again invertible it holds that $\left(A+\mathbf{u}\mathbf{v}^T\right)A^{-1}$ is invertible. For an invertible matrix A it holds that the only solution to $A\mathbf{x}=0$ is the trivial solution $\mathbf{x}=\mathbf{0}$. Therefore, with the assumption that $\mathbf{u}\neq 0$ and the above identity it follows that $1+\mathbf{v}^TA^{-1}\mathbf{u}\neq 0$. □

7.2. Woodbury matrix identity

The previous section presented a formula which allows for the numerically cheap computation of a rank-1 corrected matrix. However, in real-life applications it often occurs that a

matrix is corrected on a higher number of points. To this extend, this section presents the Woodbury matrix identity which is a generalization of the Sherman-Morrison formula in the sense that it allows for a k-rank correction with arbitrary value k.

Theorem 3 (Woodbury matrix identity)**.** *Let $A \in \mathbb{R}^{n\times n}, U \in \mathbb{R}^{n\times k}, C \in \mathbb{R}^{k\times k}, V \in \mathbb{R}^{k\times n}$. Furthermore, assume A and C are invertible. Then $(A+UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1}$.*

Proof. To prove the theorem, $A+UCV$ will be multiplied with the stated inverse and it will be shown that this indeed equals the identity. The proof is only given for multiplication on the right-hand side since the proof with multiplication on the left-hand side is analogous.

$$
\begin{aligned}
&(A+UCV)\left(A^{-1} - A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1}\right)\\
&= I - U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1} + UCVA^{-1} - UCVA^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1}\\
&= I + UCVA^{-1} - \left(U\left(C^{-1}+VA^{-1}U\right)^{-1} + UCVA^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}\right)VA^{-1}\\
&= I + UCVA^{-1} - \left(\left(U+UCVA^{-1}U\right)\left(C^{-1}+VA^{-1}U\right)^{-1}\right)VA^{-1}\\
&= I + UCVA^{-1} - UC\left(\left(C^{-1}+VA^{-1}U\right)\left(C^{-1}+VA^{-1}U\right)^{-1}\right)VA^{-1}\\
&= I + UCVA^{-1} - UCVA^{-1}\\
&= I
\end{aligned}
$$

□

Taking $k = 1$ it is clear that this is indeed a generalization of the Sherman-Morrison formula.

In the situation in which the low-rank update is symmetric, it can be written as UCU^T with C a symmetric matrix. The Woodbury matrix identity then simplifies to

$$\left(A+UCU^T\right)^{-1} = A^{-1} - A^{-1}U\left(C^{-1}+U^TA^{-1}U\right)^{-1}\left(A^{-1}U\right)^T. \tag{7.1}$$

The application of the Woodbury matrix identity can vary greatly depending on the application. Hagar [10] presented the framework of most common applications. Assuming two linear systems are given $K_1\mathbf{y} = \mathbf{f}_1$, $K_2\mathbf{x} = \mathbf{f}_2$, where the matrix $K_2 \in \mathbb{R}^{n\times n}$ is obtained by a (not necessarily symmetric) rank-k update of $K_1 \in \mathbb{R}^{n\times n}$, the framework is as follows.

1. Define two matrices $U \in \mathbb{R}^{n\times k}, V \in \mathbb{R}^{k\times n}$ such that $UV = K_2 - K_1$, which implies the choice $C = I_k$.
2. Solve the system $K_1\mathbf{y} = \mathbf{f}_1$ using a direct solution method such that a factorisation of K_1 is known.
3. Calculate $\mathbf{w} := K_1^{-1}\mathbf{f}_2$ by back and forward substitution of the factorisation of K_1.
4. Calculate $A := K_1^{-1}U$ by back and forward substitution of the factorisation of K_1.
5. Calculate the matrix-vector product $\mathbf{v}_w := V\mathbf{w}$

6. Calculate the matrix-matrix product $V_A := VA$
7. Calculate the matrix-update $B := I + V_A$
8. Obtain $z := B^{-1}\mathbf{v}_w$ by solving the lower-order system $B\mathbf{z} = \mathbf{v}_w$.
9. Calculate the matrix-vector product $\mathbf{c} := A\mathbf{z}$.
10. Calculate the solution by vector subtraction $\mathbf{x} = \mathbf{w} - \mathbf{c}$.

To motivate that this approach is equivalent to using the Woodbury matrix identity, the expression for $\mathbf{x}$ will be expanded by substitution of the intermediate steps of the above framework.

$$\begin{aligned}
\mathbf{x} &= \mathbf{w} - \mathbf{c} \\
&= K_1^{-1}\mathbf{f}_2 - A\mathbf{z} \\
&= K_1^{-1}\mathbf{f}_2 - K_1^{-1}UB^{-1}\mathbf{v}_w \\
&= K_1^{-1}\mathbf{f}_2 - K_1^{-1}U\left(I + V_A\right)^{-1}V\mathbf{w} \\
&= K_1^{-1}\mathbf{f}_2 - K_1^{-1}U\left(I + VA\right)^{-1}VK_1^{-1}\mathbf{f}_2 \\
&= K_1^{-1}\mathbf{f}_2 - K_1^{-1}U\left(I + VK_1^{-1}U\right)^{-1}VK_1^{-1}\mathbf{f}_2 \\
&= \left(K_1^{-1} - K_1^{-1}U\left(I + VK_1^{-1}U\right)^{-1}VK_1^{-1}\right)\mathbf{f}_2
\end{aligned}$$

From this derivation it is clear that the proposed framework does indeed calculate the solution of the system $K_2\mathbf{x} = \mathbf{f}_2$ by exploiting the decomposition of K_1 using Woodbury's identity.

For any practical problem more than one iteration is required. As a result, the rank of the update from the stiffness matrix from the first iteration to the stiffness matrix from the i-th iteration typically rises with i. The application of the Woodbury identity achieves improved computing times by reusing the decomposition of the original stiffness matrix and only requiring some relatively cheap back substitutions and solving a significantly smaller system of order k. However, after a significant number of iterations it is possible that k is of the same order as n possibly resulting in the above method requiring more time than a direct method. Therefore, it might be necessary to restart the above procedure by computing a new decomposition once the rank of the update becomes too large. The point at which the above procedure is restarted balances the costs of computing a new matrix decomposition against solving a smaller linear system with some additional overhead in the form of back/forward substitutions and matrix-matrix, matrix-vector products.

8

Research proposal

This literature study has provided an introduction to sequentially linear analysis which can be used when nonlinear finite element analyses suffer from robustness issues. The advantage of SLA over NLFEA is that the nonlinear material behaviour is approximated with a series of linear systems. Two different classes of solution methods for these linear systems have been discussed as well as several techniques to accelerate these methods. The aim of this thesis, part of which is this literature study, is to apply these techniques to the solver implemented in *DIANA* with the aim of reducing computing time. Therefore, the following main research question has been formulated.

How can sequentially linear analysis be improved such that it requires reduced computing time?

To answer this question, the following three approaches are proposed along with sub-questions.

Woodbury matrix identity
The first approach is to apply the *Woodbury matrix identity*, as described in Chapter 7, to the direct solution methods in *DIANA*. The issue of the order of the rank-update increasing every iteration was discussed. Therefore, the following sub-question is formulated.

At which point has the rank-update become too large and is a new decomposition of the stiffness matrix required?

Deflation
A possible improvement to iterative solution methods is applying deflation to the rigid body modes. The aim is to investigate whether applying deflation can result to improved computing times. Therefore, the following sub-question has been formulated.

Is it necessary to apply deflation to SLA in order to achieve improved computing time?

Furthermore, another relevant question regarding deflation is the following.

Is choosing U from Woodbury's identity as deflation space a good choice?

Common framework
Both previous approaches show similarities. Therefore, the third proposed approach is to analyse if both methods can be assembled into a common framework. To this extend, the following sub-question is formulated.

Is the Woodbury matrix identity mathematically equivalent to deflation applied to iterative methods?

Preconditioning
Every linear analysis, a decomposition is made of the stiffness matrix. After a small rank update of this matrix, the decomposition is still similar to the new stiffness matrix. Therefore, the decomposition can possibly be used as a preconditioner to the new problem. Therefore, the following sub-question is formulated.

Is it effective to use the decomposition of the stiffness matrix as a preconditioner for the system of linear equations of the following iteration?

The analysis of nonlinear static material behaviour is only one of many areas of interest of *DIANA*. It is therefore desirable to have the infrastructure of the *DIANA* solver remain untouched such that the improvements of this thesis do not interfere with other parts of the *DIANA* source-code. One of the aims of this thesis will thus be to implement the proposed methods as a framework around the existing solver infrastructure.

In order to be able to assess how the proposed methods changed the computing times, some test problems have to be defined. These problems will not only serve to validate whether the results are the same, but also how the changes have affected the computing times. Therefore, the following test problems have been defined.

1. **Reinforced slab:** As a first test, it was chosen to include a problem that takes a considerable amount of time such that the improvements to computing time could clearly be observed. Therefore, the problem of an reinforced concrete slab is included which uses solid elements resulting in a high number of degrees of freedom, and a higher number of integration points. A visualisation of the problem can be seen in Figure 8.1.

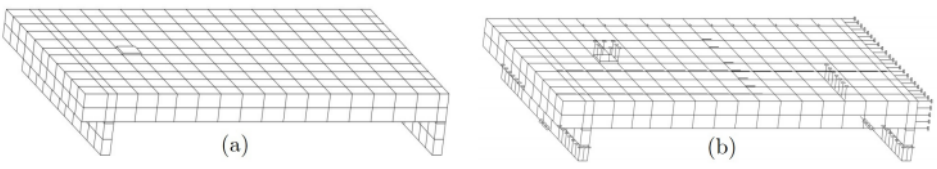


Figure 8.1: The adopted mesh (left) and the loading/boundary conditions of the reinforced concrete slab (right). Figure courtesy of [21].

The table below provides an overview of the problem.

Table 8.1: Specifications of slab-reinf properties

Property	Specification
Elements	465
Type	Solid brick element (HX24L)
Nodes	8
DOF	24
Integration scheme	$3 \times 3 \times 3$
Analysis steps	25000

Due to the high number of degrees of freedom, this problem takes several hours of computing time.

2. **Shear wall:** The second example is aimed at investigating the effects of mesh refinement on the numerical results. To this extend, a masonry wall of dimensions 1.1m × 2.76m × 0.102m (w × h × d) has been discretized using two different meshes, the first with 250 (10×25) and a refined mesh with 1210 (22×55) elements. A description of the model is given in the table below.

Table 8.2: Specifications of shrwal properties

Property	Specification
Elements	250 & 1210
Type	Plane stress element (CQ16M)
Nodes	8
DOF	16
Integration scheme	$2 \times 2 \times 2$
Analysis steps	12500

Due to the lower number of degrees of freedom and analysis steps, these two examples take significantly less computing time than the first example.

In order to answer to implement the proposed methods and answer the mentioned research questions, the following planning is proposed for the remainder of the thesis.

Topic	Duration (months)
Implement & validate Woodbury matrix identity	±2
Implement & validate deflation	±2
Check common framework both methods	±1

Bibliography

[1] Pardiso user guide version 5.0.0. http://pardiso-project.org/manual/manual.pdf, February 2014.

[2] K.E. Atkinson. *An Introduction to Numerical Analysis*. John Wiley & Sons, 2nd edition, 1989.

[3] DIANA FEA BV. Diana finite element analysis theory manual. https://dianafea.com/manuals/d102/Diana.html.

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

[5] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. *ACM In Proc. 24th National Conference*, pages 157–172, 1969.

[6] Biswa Nath Datta. *Numerical Linear Algebra and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 2010.

[7] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *Society for Industrial and Applied Mathematics*, Vol. 23, 2001.

[8] J.A. George. Computer implementation of the finite element method. Technical report, Computer Science Dept. Stanford University, February 1971. STAN-CS-71-208.

[9] G.H. Golub and C.F. van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, 4th edition, 2013.

[10] W.W. Hager. Updating the inverse of a matrix. *SIAM Review*, Vol. 31(2):221–239, June 1989.

[11] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM Society for Industrial and Applied Mathematics, 2nd edition, 2002.

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

[13] K.J.Bathe. *Finite Element Procedures*. Prentice Hall, 2nd edition, 2014.

[14] R. A. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *Society for Industrial and Applied Mathematics*, Vol. 24, 1987.

[15] J.G. Rots. Sequentially linear continuum model for concrete fracture. In de Borst et al, editor, *Fourth International Conference on Fracture Mechanics of Concrete and Concrete Structures*, volume 2, pages 831–839, 2001.

[16] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2nd edition, 2003.

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

[18] A. van de Graaf. *Sequentially linear analysis for simulating brittle failure*. PhD thesis, TU Delft, 2017.

[19] H.A. van der Vorst and K. Dekker. Conjugate gradient type methods and preconditioning. *Journal of Computational and Applied Mathematics*, Vol. 24:73–87, March 1988.

[20] J. van Kan, A. Segal, and F. Vermolen. *Numerical methods in Scientific Computing*. VSSD, 2nd edition, 2014.

[21] L.O. Voormeeren. Extension and verification of sequentially linear analysis to solid elements, 2011.

[22] M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal of Algebraic and Discrete Methods*, Vol. 2:77–79, 1981.

[23] O.C. Zienkiewicz. *The Finite Element Method in Engineering Science*. McGraw-Hill, 2nd edition, 1971.