Methods for improving the computational performance of sequentially linear analysis

by

W. Swart

to obtain the degree of Master of Science in Applied Mathematics

at the Delft University of Technology,

to be defended publicly on Thursday August 30, 2018 at 10:00 AM.

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

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

Summary

The numerical simulation of brittle failure with nonlinear finite element analysis (*NLFEA*) remains a challenge due to robustness issues. These problems are attributed to the softening material behaviour and the iterative nature of the Newton-Raphson type methods used in *NLFEA*. However, robust numerical simulations become increasingly important, for example due to recent developments in Groningen.

To address these issues, sequentially linear analysis (*SLA*) was developed which exploits the fact that a linear analysis is inherently stable. By assuming a stepwise material degradation the nonlinear response of a structure can be approximated with a sequence of linear analyses. Although this approach has been proven to be effective for several case studies, the numerical performance is still a problem that has to be solved. After every linear analysis, a single element is damaged resulting in incremental damage. As a result, the system of equations only changes locally between these linear analyses. Traditional solution techniques do not exploit this property and calculate a matrix factorisation every linear analysis, resulting in high computational times per analysis step. Since *SLA* typically requires many linear analyses to obtain the desired structural response, this leads to unacceptable analysis times.

The aim of this thesis is to improve the computational performance of *SLA* by developing numerical solution techniques which exploit the incremental approach of *SLA*. To this extend, the following methods have been developed.

1. A direct solution technique has been developed which is based on the *Woodbury matrix identity*. This identity allows for the numerically cheap computation of the inverse of a low-rank corrected matrix. In this approach, the expensive matrix factorisation does not have to be calculated every linear analysis step. Instead, the old factorisation can be reused along with some additional matrix- and vector multiplications and solving a significantly smaller linear system of equations. An optimal strategy is derived to determine at which point a new factorisation should be calculated.
2. An improved preconditioner for the conjugate gradient (*CG*) method has been developed. Instead of an incomplete factorisation, the complete factorisation is used as a preconditioner which reduces the number of required *CG* iterations significantly. The point at which too many *CG* iterations are required and a new factorisation is necessary, is determined using the same strategy as the first method.

From numerical experiments it follows that both methods perform significantly better than the direct solution method, especially for large 3-dimensional problems. The best performance is achieved using the *Woodbury matrix identity* resulting in the solver no longer being the dominant factor in *SLA*. Furthermore, significantly larger problems are not solvable in time frames in which previously only smaller problems were solved.

Acknowledgements

This thesis was submitted as the final requirement to obtain the degree of Master of Science in Applied Mathematics at the Delft University of Technology. I would like to take this opportunity to thank the people that have been a part of this journey. First and most importantly, I want to thank my thesis advisor Martin van Gijzen for his enthusiasm and continuous support. In the numerous difficult times this thesis had to offer, you were always available and our discussions helped me greatly. I would also like to thank Gerd-jan Schreppers for giving me the opportunity to write my thesis at *DIANA FEA BV*. It has been an experience I will never forget. Moreover, I would like to thank Jos Jansen for his extensive help in getting me up-to-speed in the complex world of *DIANA*, without which this thesis would have been much more difficult. Moreover, I would like to thank Wim van Horssen and Jan Rots for taking the time and effort to be part of my graduation committee.

I would also like to show gratitude to a few people who may not have contributed to this thesis directly, but whose assistance has helped me greatly. First of all, a huge thanks to Manimaran Pari for his patience in helping me get familiar with the basics of civil engineering as well as his help in the many practical problems that we encountered during our time at *DIANA*. Secondly, I would like to thank Panos Evangeliou for helping me with most of the practical problems and for the many hours of debugging you have helped me with. I would also like to thank Wijtze Pieter Kikstra for all the questions that you helped me with. A special thanks goes out to all the *protein lunch*-members for all the healthy lunches that we had together. They were always a welcome distraction and have made my time at *DIANA* that extra bit enjoyable and memorable. Furthermore, I would like to thank my friends and roommates for being a listening ear when I needed to vent my ideas and offering me the necessary distractions after work.

Last but not least, I would like to thank my dear parents Karin and Norbert and my sister Marjolein for their unconditional love and support during my years of study and through the process of writing this thesis. This accomplishment would not have been possible without you.

Thank you.

Wouter Swart
Delft, August 2018

Contents

1 Introduction

1.1. Background and motivation

In the recent years, structural failures of buildings have lead to increased attention to the (re)assessment of structural risks. Recent examples are the cracking of masonry buildings in Groningen due to gas extractions, or the partial collapse of a concrete parking garage in Eindhoven due to technical errors. Not only consumers but also insurance companies and governing bodies are interested in having a clear view of the risks involved and what measures should be taken.

To assess these risks, accurate numerical simulations are required to predict the structural behaviour. To this extend, a nonlinear finite element analysis (*NLFEA*) is typically carried out. In order for these predictions to produce reliable results, it is important that the adopted numerical method used is robust. However, when analysing for example masonry or concrete structures robustness issues can arise. These issues are inherent to the iterative nature of *NLFEA* in which loads are typically applied incrementally. After increasing the structural load, the iterative process attempts to establish an equilibrium between external and internal forces. Once the imbalance is sufficiently small the solution is said to have converged after which the applied load is incremented 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. In these situations the Newton-Raphson type methods that are used to establish force equilibrium may not find a converged solution. Furthermore, convergence may be difficult when numerous cracks develop simultaneously such that the method has to determine where new cracks will appear and which existing ones will propagate or close. The masonry and concrete structures which are often subject to the *NLFEA* are characterized by brittle behaviour meaning that cracks can occur instantaneous and propagate rapidly. As a result, the iterative procedure may not be able to find a converged solution.

To address the outlined issues, Rots proposed a robust finite element analysis (*FEA*) technique named sequentially linear analysis (*SLA*) [14]. The main assumption of this method is that the material degrades incrementally, that is the stiffness and strength properties of the material decrease stepwise. By applying only one damage increment per analysis step the damage is controlled such that it is not possible for multiple cracks to form simultaneously. To locate at

which point in the model the damage is applied an automated selection procedure is applied 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 can be scaled such that the stress reaches maximal strength only in one element, resulting in progressive damage. After the damage is incremented, the system of equations is reassembled 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 nonlinear system of equations has to be solved, making *SLA* robust.

SLA has been implemented into *DIANA* (DIsplacement ANAlyzer), a specialized civil engineering software package developed by *DIANA FEA BV*. Although the implementation of *SLA* has been proven to be effective in robustly simulating brittle failure [16], it can be computationally intensive. Since only a single element is damaged per analysis step, and cracks constitute of numerous elements failing, typically many linear analyses have to be performed to obtain a desired structural response. To illustrate the effect on performance of the different procedures in *SLA*, Figure 1.1 shows how the computational times of the dominant procedures scale with the problem size.

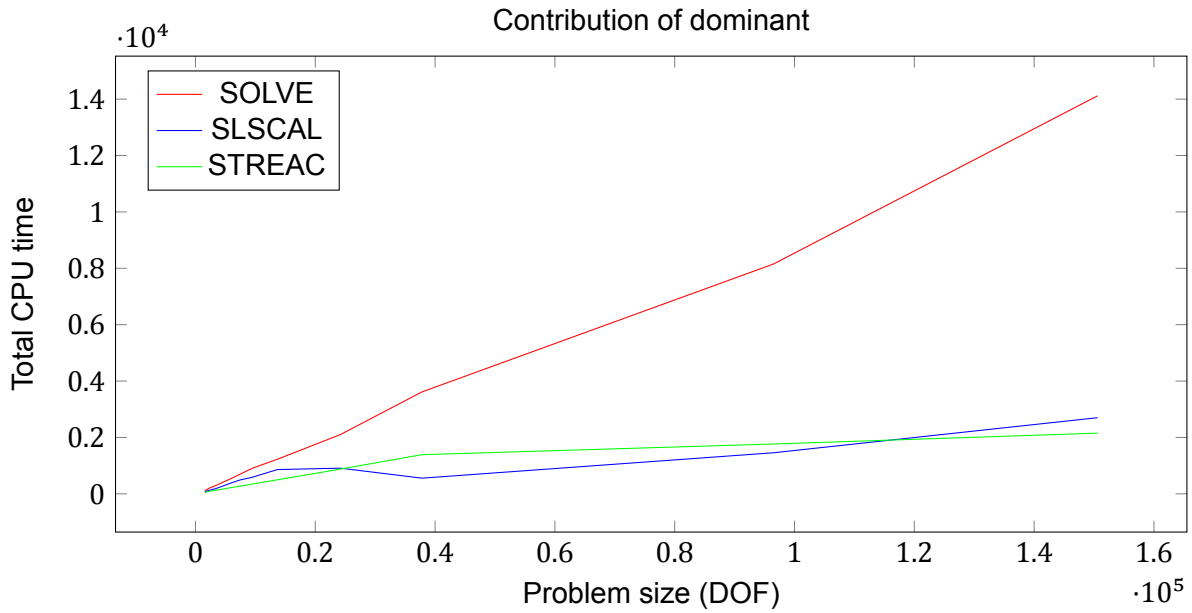


Figure 1.1: Total CPU time of the most dominant building blocks of *SLA*.

From the Figure 1.1 it is clear that for increasing problem sizes, solving the linear system (SOLVE) of equations becomes the bottleneck of *SLA*. This poor numerical performance can be attributed to the absence of efficient reusage of solutions from previous analysis steps. Since only one element is damaged after each linear analysis, the resulting system of linear equations remains mostly unchanged between analysis steps. Only a small number of entries corresponding to the particular damaged element are adjusted meaning that the system matrix is given a low-rank correction. Therefore, calculating the solution without exploitation of this property comes at the cost of significant additional computing time, especially for large problems.

As a result of these performance issues, a master thesis position was made available at *DIANA*

FEA BV with the aim of addressing the mentioned problems. Although the required number of analysis steps is an inherent result of the stepwise material degradation assumption of *SLA*, the reusage of solutions from earlier analysis steps is the result of the implementation of the *SLA* solver procedure. Therefore, the aim of this master thesis is to develop numerical solution techniques which exploit the incremental approach of *SLA* to achieve improved computing times.

1.2. Research question

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

How can the computational performance of sequentially linear analysis be improved such that it requires reduced computing time?

In this thesis the computing time is understood to be the elapsed time of the analysis. To answer the main research question the following sub-questions will be addressed

1. Can the computational performance of *SLA* be improved using the *Woodbury matrix identity* for low-rank matrix corrections?
2. How can the conjugate gradient method be used to exploit the low-rank nature of *SLA* in terms of performance?
3. Which of the two mentioned approaches performs best and how do they compare to each other?

Reducing the analysis times allows for larger problems to be solved in the same amount of time as smaller problems were previously solved. The thesis presented here is aimed at increasing the size of problems that can be solved using *SLA*. Next, an outline is given of this thesis.

1.3. Outline

In the overview below an outline is given of the structure of this thesis. First an introduction is given on the finite element method including a brief description on its application in structural mechanics. 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 by exploiting the event-by-event strategy of *SLA*. Finally, results of both methods are presented and conclusions are drawn.

Chapter 1: Introduction and motivation to the problem.

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

Chapter 3: Explanation of incremental-iterative solution techniques for solving nonlinear problems.

Chapter 4: Explanation of the general idea of sequentially linear analysis, an alternative approach to nonlinear finite element analysis.

Chapter 5: Elaboration on two solution classes for solving systems of linear equations.

Chapter 6: Derivation and implementation details of Woodbury's identity, a mathematical technique for reusing solutions of linear systems after a low-rank matrix correction.

Chapter 7: Parametric analysis and validation of Woodbury's identity.

Chapter 8: Numerical results of both solution methods.

Chapter 9: Conclusions and discussion.

1.4. Notation and conventions

To maintain consistency throughout the report the following notation and conventions will be used.

Vectors will be denoted as bold lower case letters:

$$\mathbf{v} = [v_1, v_2, \dots v_n]^T$$

and matrices as upper case letters:

$$M = \begin{pmatrix} m_{1,1} & \cdots & m_{1,n} \\ \vdots & \ddots & \vdots \\ m_{m,1} & \cdots & m_{m,n} \end{pmatrix}.$$

Vector and matrix elements are indexed using the Matlab convention:

$$\mathbf{v}(i:j) = \left[v_i, v_{i+1}, \dots, v_j\right]^T$$

and

$$M(i:j, k:l) = \begin{pmatrix} m_{i,k} & \cdots & m_{i,l} \\ \vdots & \ddots & \vdots \\ m_{j,k} & \cdots & m_{j,l} \end{pmatrix}.$$

1.5. Definitions

Definition 1. *Let $L : U \to V$ be a linear space and suppose*

$$L(\alpha\mathbf{u} + \beta\mathbf{v}) = \alpha L\mathbf{u} + \beta L\mathbf{v} \quad \forall \mathbf{u}, \mathbf{v} \in U,\ \alpha, \beta \in \mathbb{R}.$$

Then L is a linear operator on U.

Definition 2. *Let $L : U \to V$ be a linear space. Then L is self-adjoint if and only if*

$$\int_\Omega \mathbf{u} L\mathbf{v} \,\mathrm{d}\Omega = \int_\Omega \mathbf{v} L\mathbf{u} \,\mathrm{d}\Omega \quad \forall \mathbf{u}, \mathbf{v} \in U \,.$$

Definition 3. *Let $L : U \to V$ be a linear space. Then L is positive if and only if*

$$\int_\Omega \mathbf{u} L\mathbf{u} \,\mathrm{d}\Omega \geq 0 \quad \forall \mathbf{u} \in U \,.$$

Definition 4. *Let Ω be bounded by $\partial\Omega$. Then the following function spaces are defined:*

$$C(\Omega) := \left\{ f : \Omega \to \mathbb{R} \;\middle|\; f \text{ continuous over } \Omega \right\}$$
$$C^p(\Omega) := \left\{ f : \Omega \to \mathbb{R} \;\middle|\; f \text{ is up to p times continously differentiable over } \Omega \right\}$$
$$C_0(\Omega) := \left\{ f \in C(\Omega) \;\middle|\; f|_{\partial\Omega} = 0 \right\}$$

Definition 5. *Let $\mathbf{u} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$. Then A is:*

$$\text{Symmetric if and only if} \quad A^T = A$$
$$\text{Positive definite if and only if} \quad \mathbf{u}^T A\mathbf{u} > 0 \,, \quad \forall \mathbf{u} \neq \mathbf{0}$$

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

Definition 6. *Given $1 \leq p < \infty$, the p-norm (Hölder norm) of a vector $\mathbf{u} \in \mathbb{R}^n$ denoted 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

$$\|\mathbf{u}\|_1 = \sum_{i=1}^{n} |u_i| \,.$$

Definition 7. *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 $p = 1$ the following expression for the matrix norm exists

$$\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{i,j}| \,.$$

Definition 8. *Let $A \in \mathbb{R}^{n\times n}$ and $p, q \in \mathbb{N}$. 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 $a_{i,j} = 0$ whenever $j > i + q$.*

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

2

Finite element method

The finite element method (*FEM*) is a numerical method for solving boundary value problems (*BVP*). The method is employed extensively in various areas such as the analysis of solids and structures or heat transfer. 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 *BVP*. The first is known as Ritz' method and determines the solution by converting the *BVP* to a minimisation problem. The second method determines the solution to the *BVP* 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 classic textbooks such as Zienkiewicz [19] or Bathe [12].

2.1. Ritz' method

Ritz' method is a method which can be used to find an approximate solution to a *BVP* by solving an equivalent minimisation problem that is in some sense equivalent to the *BVP* [18]. These minimisation problems often aim at minimising an underlying energy potential or shortest path to solve the equivalent *BVP*. The advantage of the minimisation 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 minimisation problem are called *essential*. Boundary conditions that follow implicitly from the minimisation problem are referred to as *natural* boundary conditions. To derive the equivalent minimisation problem consider a *BVP* written in the general form

$$Lu = f\,, \tag{2.1}$$

with $L : U \to V$ a linear operator. Assuming self-adjointness and positivity of L, it can be shown that an equivalent minimisation exists. Under these assumptions, the solution u to the *BVP* of Equation (2.1) minimises the functional

$$F(u) = \int_\Omega \frac{1}{2} uLu - uf \mathrm{d}\Omega \tag{2.2}$$

over the space U. The original *BVP* is thus rewritten to finding u that satisfies the boundary conditions and for which Equation (2.2) is minimised. Direct minimisation is only possible if there are a finite number of unknowns. To this extend, the solution u to the minimisation problem is approximated as

$$u(\mathbf{x}) \approx u^n(\mathbf{x}) = \sum_{j=1}^{n} a_j \varphi_j(\mathbf{x}) \,, \tag{2.3}$$

where $\{\varphi_j(\mathbf{x})\}$ are chosen linear independent basis functions, a_i are the weights of these functions and $\mathbf{x} = (x, y)$[1]. Using this approximation, the only unknowns in the minimisation are the weights $a_1, \dots, a_n$. The necessary condition for the existence of a minimum is then given by

$$\frac{\partial F\left(u^n\left(\mathbf{x}\right)\right)}{\partial a_i} = 0 \,, \quad i = 1, \dots, n \,. \tag{2.4}$$

Equation (2.4) forms a set of n equations with n unknowns which can be solved uniquely if the linear operator L from Equation (2.1) is positive definite.

As mentioned, an equivalent minimisation problem does not necessarily exist for arbitrary *BVP*'s. In these cases, an alternative formulation is required. Galerkin's method provides this alternative formulation which is applicable to all kinds of *BVP*'s.

2.2. Galerkin's method

Galerkin's method is a direct generalisation of Ritz' method for solving *BVP*'s. In the case that an equivalent minimisation exists, Ritz' and Galerkin's method are equivalent, making Galerkin's method more generally applicable. Before applying Galerkin's method, a weak formulation is derived from the *BVP*. The aim of the weak formulation is to allow a larger solution class than the *BVP* admits. To illustrate how to obtain the weak formulation of a given *BVP*, consider the classical Poisson problem

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = g(x, y) & \text{on } \partial\Omega \end{cases} \tag{2.5}$$

Multiplication with a test function $\varphi \in C_0^1(\Omega)$ and integration over the domain Ω the *BVP* from Equation (2.5) can be written as

$$-\int_\Omega \varphi \Delta u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega \,, \quad \forall \varphi \in C_0^1(\Omega) \,. \tag{2.6}$$

Applying integration by parts and Gauss's divergence theorem to Equation (2.6) allows it to be rewritten as

$$\begin{aligned} -\int_\Omega \nabla(\varphi \nabla u) - \nabla\varphi \nabla u \mathrm{d}\Omega &= \int_\Omega \varphi f \mathrm{d}\Omega \,, \quad \forall \varphi \in C_0^1(\Omega) \\ \Leftrightarrow -\int_{\partial\Omega} (\varphi \nabla u) \cdot \mathbf{n} \mathrm{d}\Gamma + \int_\Omega \nabla\varphi \nabla u \mathrm{d}\Omega &= \int_\Omega \varphi f \mathrm{d}\Omega \,, \quad \forall \varphi \in C_0^1(\Omega) \end{aligned} \tag{2.7}$$

[1] $\mathbf{x} = (x, y, z)$ for 3-dimensional problems.

Since it holds that $\varphi \in C_0^1(\Omega)$, the integral over the boundary $\partial\Omega$ vanishes and Equation (2.7) simplifies to

$$\int_\Omega \nabla\varphi \nabla u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega\,, \quad \forall\varphi \in C_0^1(\Omega)\,. \tag{2.8}$$

The weak formulation corresponding to the *BVP* of Equation (2.5) is thus

$$\begin{cases} \text{Find } u \in C_g(\Omega) \text{ such that} \\ \int_\Omega \nabla\varphi \nabla u \mathrm{d}\Omega = \int_\Omega \varphi f \mathrm{d}\Omega\,, \quad \forall\varphi \in C_0^1(\Omega) \end{cases} \tag{2.9}$$

In the original *BVP* the solution u had to be twice differentiable due to the *Laplace* operator. However, in the weak formulation of Equation (2.9) 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 *BVP*.

The weak formulation from Equation (2.9) can be solved using Galerkin's method. Dividing the area of interest Ω into n_{el} non-overlapping elements then allows the domain to be decomposed as

$$\overline{\Omega} \simeq \bigcup_{k=1}^{n_{el}} \overline{e_k}\,. \tag{2.10}$$

Similarly to Ritz' method, the solution to the weak formulation is approximated as a linear combination of basis functions defined on these elements

$$u(\mathbf{x}) \approx u^n(\mathbf{x}) = \sum_{j=1}^{n} u_j \varphi_j(\mathbf{x})\,. \tag{2.11}$$

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 should equal the Kronecker delta function

$$\varphi_i(\mathbf{x}_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{2.12}$$

Substitution of the approximation of u in weak formulation of Equation (2.9) then yields

$$\begin{aligned} \int_\Omega \nabla\varphi_i \nabla \left(\sum_{j=1}^{n} u_j \varphi_j \right) \mathrm{d}\Omega &= \int_\Omega \nabla\varphi_i \sum_{j=1}^{n} u_j \nabla\varphi_j \mathrm{d}\Omega \\ &= \sum_{j=1}^{n} u_j \int_\Omega \nabla\varphi_i \nabla\varphi_j \mathrm{d}\Omega = \int_\Omega \varphi_i f \mathrm{d}\Omega \end{aligned} \tag{2.13}$$

Typically, two new variables are introduced for the integrals over the elements

$$K_{ij} = \int_\Omega \nabla\varphi_i \nabla\varphi_j \mathrm{d}\Omega = \sum_{k=1}^{n_{el}} \int_{e_k} \nabla\varphi_i \nabla\varphi_j \mathrm{d}\Omega := \sum_{k=1}^{n_{el}} K_{ij}^{e_k} \tag{2.14}$$

$$f_i = \int_\Omega \varphi_i f \mathrm{d}\Omega = \sum_{k=1}^{n_{el}} \int_{e_k} \varphi_i f \mathrm{d}\Omega := \sum_{k=1}^{n_{el}} f_i^{e_k} \tag{2.15}$$

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 (2.14) and (2.15) into Equation (2.13) the expression simplifies to

$$\sum_{j=1}^{n} K_{ij} u_j = f_i \,, \quad i = 1, \dots n \,, \tag{2.16}$$

which defines a set of n equations with n unknowns. These equations can be assembled to obtain the matrix-vector notation of the linear system of equations

$$K\mathbf{u} = \mathbf{f} \,. \tag{2.17}$$

The next section will provide an introduction on how the finite element method is applied to derive the system of equations in structural mechanics.

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 from which the stresses and strains can be derived.

The two quantities (mechanical) stress and strain are closely related. Where stress, denoted by σ, is a quantity that expresses the internal forces per area that neighbouring particles exert on each other, strain, denoted by ε, 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.

To derive a system of equations, the area of interest Ω with boundary Γ is again divided into non-overlapping elements on which basis functions are defined. The displacement can then be approximated as a linear combination of these basis functions as

$$u(\mathbf{x}) \approx u^n(\mathbf{x}) = \sum_{j=1}^{n} u_j \varphi_j(\mathbf{x}) \,. \tag{2.18}$$

In structural mechanics, Equation (2.18) is typically written in matrix-vector notation as

$$u^n(\mathbf{x}) = N(\mathbf{x})\mathbf{u} \,, \tag{2.19}$$

where $N(\mathbf{x}) = [\varphi_1(\mathbf{x}) \dots \varphi_n(\mathbf{x})]$ and $\mathbf{u} = [u_1, \dots, u_n]^T$. The strain in an element can be calculated using the well-known *strain-displacement* relation [19]

$$\boldsymbol{\varepsilon}_i = B_i(\mathbf{x})\mathbf{u}\,. \tag{2.20}$$

The matrix B_i contains the spatial derivatives of the basis functions. In the case of linear basis functions, these derivatives are constant over the element. However, for higher-order basis functions B_i should be evaluated for every node in the element.

Assuming linear elastic behaviour in the structure, the local stresses can be calculated from the strains as:

$$\boldsymbol{\sigma}_i = D_i\,(\boldsymbol{\varepsilon}_i - \boldsymbol{\varepsilon}_0) + \boldsymbol{\sigma}_0\,. \tag{2.21}$$

In this equation D_i is the *stress-strain*-relation which is a function of the related material properties such as Young's modulus[2] and Poisson's ratio[3]. 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.

Instead of deriving the system of equations using the previously mentioned methods, a simpler way of deriving these is using the *principle of virtual displacements*. This principle states that an elastic structure is in equilibrium under a given loading system if, for any virtual displacement from a compatible state of deformation, the virtual work is equal to the virtual strain energy [2]. The virtual work equation is then

$$\int_\Omega \delta\boldsymbol{\varepsilon}^T\boldsymbol{\sigma}\mathrm{d}\Omega = \int_\Omega \delta\mathbf{u}^T\mathbf{g}\mathrm{d}\Omega + \int_\Gamma \delta\mathbf{u}^T\mathbf{t}\mathrm{d}\Gamma\,, \tag{2.22}$$

where $\delta\boldsymbol{\varepsilon}$ are the virtual strains corresponding to the virtual displacements $\delta\mathbf{u}$, $\mathbf{g}$ is the vector of known body forces and $\mathbf{t}$ the vector of traction forces[4]. Using the strain-displacement and stress-strain relations from Equations (2.20) and (2.21) and using the fact that Equation (2.22) should hold for any displacement $\delta\mathbf{u}$, it can be written as a set of n equations with n unknowns. It follows that the elemental contributions to the stiffness matrix are given by

$$K_{ij}^{e_m} = \int_\Omega B_m^T D_m B_m \mathrm{d}\Omega\,. \tag{2.23}$$

Introducing a mapping T_i which maps the local element numbering to the global numbering, assembling the element stiffness matrices yields

$$K_{ij} = \sum_{m=1}^{n_{el}} T_m^T K_{ij}^{e_m} T_m\,. \tag{2.24}$$

Similarly, the principle of virtual displacements can be applied to find an expression for the element force vectors. Assembling the element matrices and force vectors defines n equations with n unknowns of the familiar form

$$K\mathbf{u} = \mathbf{f}\,. \tag{2.25}$$

[2]Measure for stiffness of solids.

[3]Measure for expansion/contraction perpendicular to the direction of applied compression/tension.

[4]For example surface, edge or point loads.

In the above derivation of the system of equations many of the details are left out. The classical textbooks by Zienkiewicz [19] and Bathe [12] provide extensive elaborations on how the principle of virtual displacements can be applied to derive the system of equations, as well as numerous examples.

In the definitions of the element stiffness matrices and element force vectors, an integral over the element has to be evaluated. For 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 a system of linear equations can be assembled from elemental contributions. These elemental contributions are defined as integrals over the element which can in general not be evaluated analytically. Therefore, before the system of equations can be solved these integrals need to be approximated numerically. Numerical integration is typically performed using Newton-Cotes, Simpson, Lobatto or Gauss integration. These integration schemes approximate the integral over an element by evaluating the integrand at equally spaced points $\boldsymbol{\xi}_i$, referred to as *integration points*. The integral can then be approximated as

$$\int_{e_m} g\left(\mathbf{x}\right) \mathrm{d}\Omega \approx \sum_{i=1}^{n_\xi} w_{\xi_i} g\left(\boldsymbol{\xi}_i\right) \ , \tag{2.26}$$

where n_ξ is the chosen number of integration points and w_{ξ_i} is the weight function of the chosen integration scheme. The minimal number of integration points depends on the integration scheme and the order of the used basis functions. Increasing the number of integration points improves the approximation of Equation (2.26), however, more calculations have to be performed to obtain the stresses and strains which are defined on integration point level.

3

Nonlinear analysis

In nonlinear finite element analysis the relation between the exerted forces and the displacement is no longer linear, resulting from material-, geometric- or contact nonlinearities, or a combination. 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 find this equilibrium, the problem is not only discretised spatially but also temporal (increments). Note that the notion of time here is artificial. Since static structural analyses are performed there is no real notion of time. Instead, it refers to the incremental application of the external load. To obtain a feasible solution within such an increment, an iterative procedure is used to balance the forces. Due to the combination of incremental increase of the load and the iterative nature within increments, 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 the displacement increment $\Delta\mathbf{u}$ such that the resulting displacement equates the internal and external forces

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

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 problems 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 determines the equilibrium between the external and internal forces by calculating the stiffness matrix in every iteration. As a result, every prediction is based on the most recent information. This property is illustrated in Figure 3.1.

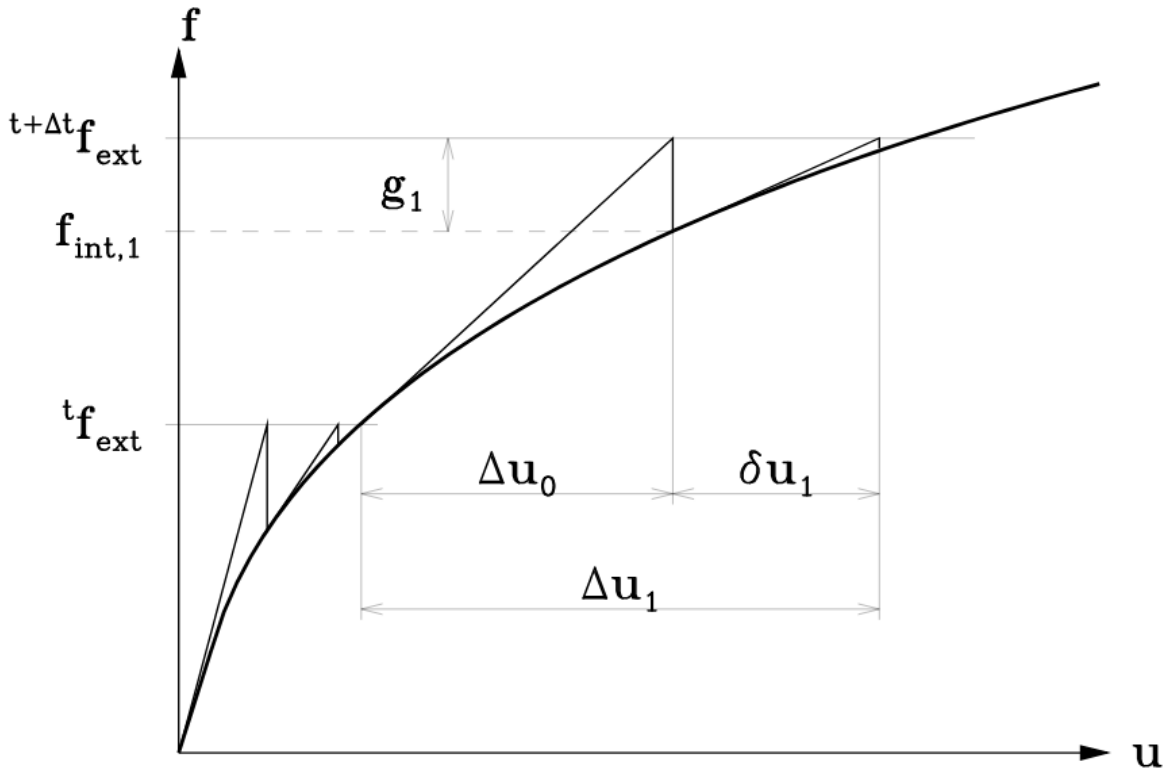


Figure 3.1: Example of regular Newton-Raphson.

Figure 3.1 shows the *force-displacement* curve, which is common in structural mechanics. From the figure it is not immediately clear that the roots of Equation (3.3) are searched. As illustration, consider a displacement $\mathbf{u}^t$ is found for which the internal force equals ${}^t\mathbf{f}_{\text{ext}}$. The

external force is then incremented to ${}^{t+\Delta t}\mathbf{f}_{\text{ext}}$. In order to determine the new displacement $\mathbf{u}^{t+\Delta t}$ for which the new internal force equals ${}^{t+\Delta t}\mathbf{f}_{\text{ext}}$, the stiffness matrix is evaluated at $\mathbf{u}^{t}$ with which a new estimate for the displacement is calculated. The internal force at this new displacement is then calculated, and if it is not equal to ${}^{t+\Delta t}\mathbf{f}_{\text{ext}}$, the process is repeated until a displacement is found which equilibrates internal and external forces.

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, on the other hand, 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.

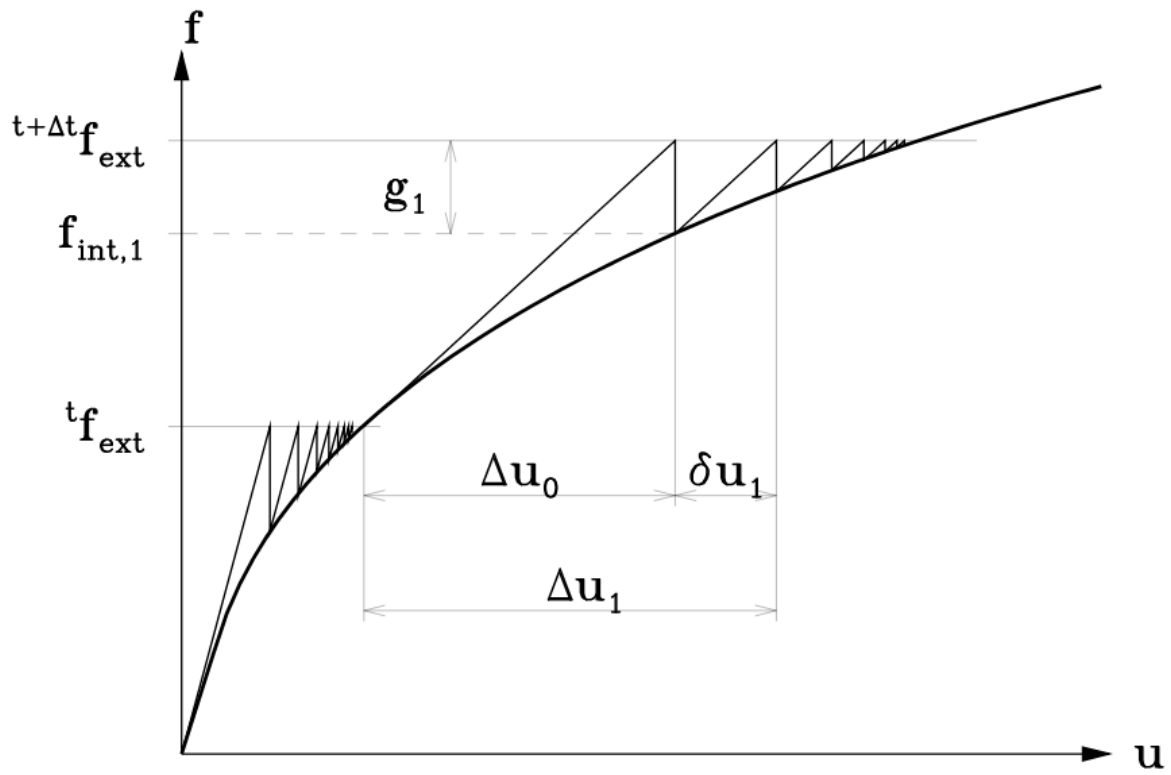


Figure 3.2: Example of modified Newton-Raphson.

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. 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. Quasi-Newton methods use the information from previous solution vectors and out-of-balance force vectors to achieve this approximation. 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 linearised form of the relation between the force and displacement vector. Two well-known quasi-Newton methods are

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

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

where $\delta \mathbf{g}_i = \mathbf{g}_{i+1} - \mathbf{g}_i$. 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.

3.3. Additional methods

Besides the mentioned Newton-Raphson methods some other iterative methods are available to obtain the force equilibrium of Equation (3.3).

The first method is the linear stiffness method. This method uses the same stiffness matrix for every increment. As a result, the costs per iteration of this method are the cheapest. However, since no new information is used many iterations are typically required.

The second method is the constant stiffness method, which uses the stiffness matrix left behind by the previous increment. Since relatively new information is used, fewer iterations are typically required than the linear stiffness method. Note that if the constant stiffness method is used from the first increment, the method is the same as 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 next 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 [3] 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 discussed below.

The first stopping criterion is based on the force norm, which is the Euclidean norm of the out-of-balance 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 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:

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

3.5. Incremental procedures

The incremental-iterative procedure consists of two parts; an *incremental* and *iterative* part. The preceding sections have focused on iterative methods. For the increment part several solution methods exist. 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 [2].

4

Sequentially linear analysis

Most nonlinear finite element codes use some kind of incremental-iterative scheme. As described in Section 3, the general idea of such schemes is to apply the loads in increments. After each increment an iterative procedure, typically some variation of Newton-Raphson, is used to solve the nonlinear system of equations. Convergence issues may occur frequently in the analysis of quasi-brittle structures using incremental-iterative schemes. These problems can be attributed to the non-smooth response of quasi-brittle structures, an example of which is shown in Figure 4.1. This figure shows a typical force-displacement curve of a brittle material. The high level of irregularity in the curve results from the characteristic behaviour of these materials, referred to as local *snapbacks*.

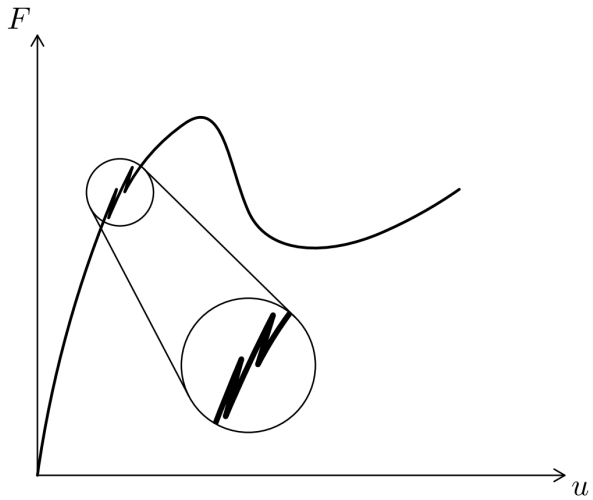


Figure 4.1: Example of a non-smooth load-displacement curve. Source: van de Graaf [16].

Due to these types of irregularly shaped equilibrium paths, path-following algorithms such as the Newton-Raphson type methods may experience convergence issues, if any is reached at all. To address these issues incremental-iterative methods pose, Rots proposed sequentially linear analysis (*SLA*) as an alternative [14]. The main idea of *SLA* is to assume incremental

material degradation and to reduce the material stiffness in areas where cracking is expected. In this way, the stress redistribution due to the cracking can be taken into account with a simple linear-elastic analysis [16]. Reducing the stiffness incrementally and using an automated selection procedure for reducing material stiffness, the nonlinear material behaviour can be approximated with a series of inherently stable linear analyses.

Generally speaking *SLA* consists of five main steps. First of all, a fundamental assumption is made by adopting some stepwise material law to approximate the underlying nonlinear relation of the model. Secondly, assuming a unit load the system of equations is assembled and solved. Thirdly, an automated procedure is applied to the linear analysis results to obtain a minimal scaling factor under which the first element reaches its stress limit. Fourthly, the results of the linear analysis are scaled according to the scaling factor. The final step is to apply a damage increment to the mentioned element according to the adopted stepwise material law. The last four steps are repeated until some stopping criteria is met. In the context of *SLA* the initiation or propagation of damage to an element is referred to as an *event*. Hence *SLA* can be seen as a event-by-event strategy. An overview of this strategy is shown in Figure 4.2.

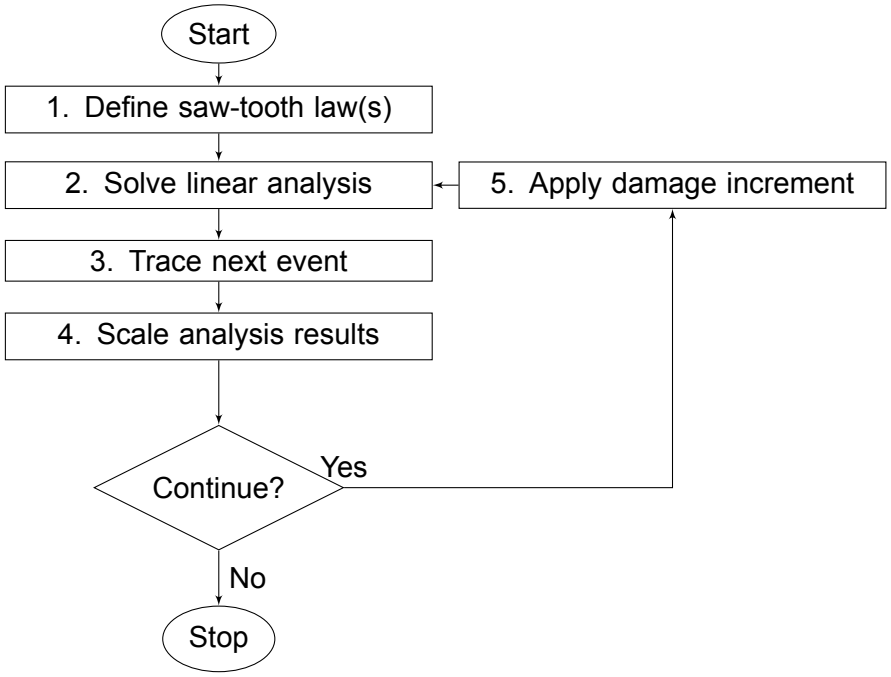


Figure 4.2: Flowchart of *SLA*'s event-by-event strategy.

In the following sections, more detail will be given on some of the main steps of *SLA*. An elaboration of step four, the scaling of a linear analysis, will be omitted due to the trivial nature of this step.

4.1. Saw-tooth laws

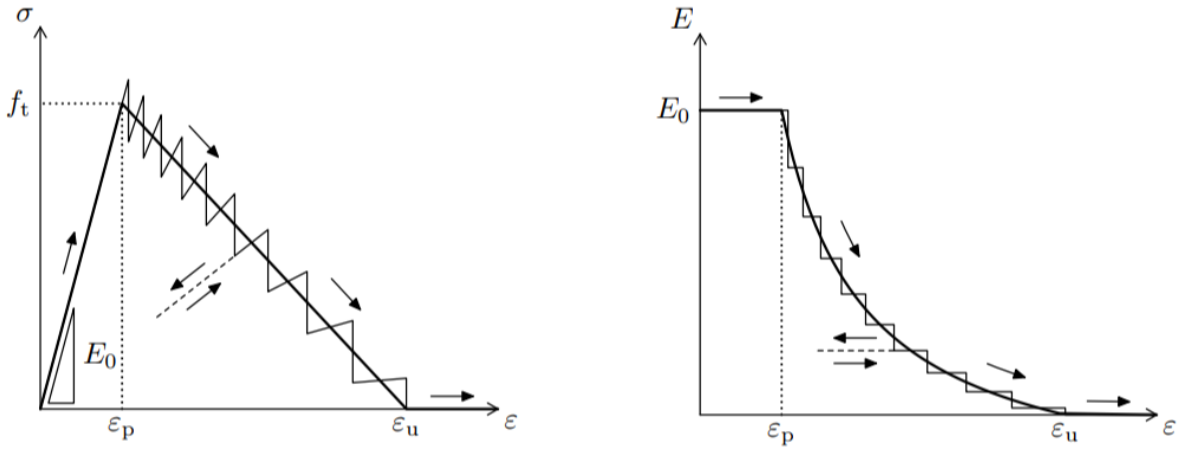
A fundamental assumption of *SLA* is to approximate the nonlinear material response with a series of linear relations. In structural engineering material behaviour is represented using *stress-strain* curves which define the relation between strain ϵ and (tensile) stress σ. Three

simple stress-strain curves are shown in Figure 4.3.

(a) Constant **(b)** Brittle **(c)** Linear

Figure 4.3: Three examples of stress-strain curves.

The stress strain curves of Figures 4.3b and 4.3c display a decrease in stress after the tensile failure point. This type of behaviour is referred to as tensile softening, which is a typical characteristic of brittle materials such as concrete and masonry. When adopting an approximating stepwise material law (referred to as saw-tooth law) to this behaviour, a balance should be found between the step size of saw-tooth law (accuracy) and the resulting number of linear analyses (performance). An example of such a saw-tooth law is shown in Figure 4.4a.



(a) Example saw-tooth law. **(b)** Reduction of Young's modulus

Figure 4.4: Example of a saw-tooth law and corresponding decrease in Young's modulus. Source: van de Graaf [16].

Each jump in Figure 4.4, the so-called secant branches, corresponds to a stepwise decrease in the Young's modulus. The Young's modulus E is defined as

$$E = \frac{\sigma(\epsilon)}{\epsilon}. \tag{4.1}$$

Hence, the stepwise reduction in Young's modulus shown in Figure 4.4b corresponds to the slopes of the saw-tooth law of Figure 4.4a. Each jump in the saw-tooth law can thus be seen as a damage increment to the element, reducing its material strength properties. Decreasing the step size of the saw-tooth law increases the number of secant branches. As a result, an increased number of linear analyses is required to reach (partial) failure of an element. Therefore, a good saw-tooth law balances the accuracy of the approximation of the nonlinear behaviour with the analysis time required for the linear analyses.

4.2. Linear analysis

Once the saw-tooth law(s) are adopted, only a load on the structure has to be defined to assemble the entire system of linear equations. Since the linear analysis will be scaled with a suitable factor after the analysis, the magnitude of this force is irrelevant and a simple unit load is assumed. After assembling the system stiffness matrix and the force vector, the system of linear is equations is solved. Several solution methods are available, most importantly direct- or iterative solution techniques. The topic of this thesis is aimed at improving these two methods in the framework of *SLA*. Section 5 provides a detailed description of both methods.

4.3. Trace event

The results from the system of linear equations yield the displacements of all degrees of freedom. From these displacements, the stresses and strains can be calculated. To determine in which element the next damage increment occurs, a critical load factor has to be determined for every integration point. This factor indicates by what factor the linear analysis can be scaled to reach the peak stress limit for the particular integration point. The critical load factor in analysis step j is defined as:

$$\lambda_{\text{crit};i}^{(j)} = \frac{f_i^{(j)}}{\sigma_{\text{gov};i}^{(j)}}\,, \tag{4.2}$$

where $\sigma_{\text{gov};i}^{(j)}$ is the governing stress component for integration point i and $f_i^{(j)}$ the current peak stress limit as defined by the current secant branch of the saw-tooth law. To ensure that only a integration point reaches its peak stress limit, the linear analysis is scaled with the minimum of all load factors. The minimal load factor is referred to as the critical load factor and is defined as

$$\lambda_{\text{crit}}^{(j)} = \min_i \left(\lambda_{\text{crit};i}^{(j)}\right) \quad \text{for all} \quad \lambda_{\text{crit};i}^{(j)} > 0\,. \tag{4.3}$$

In the situation where multiple integration points have the same minimal load factor, one of the integration points is selected. Scaling of the linear analysis with the critical load factor then results in the smallest load that will lead to progressive damage.

4.4. Stopping criteria

The event-by-event strategy of *SLA* captures the structural response using a sequence of *events*. At some point in the analysis, the desired response is achieved and the analysis should be terminated. Therefore, some stopping criteria should be defined and checked after each linear analysis. The most trivial stopping criteria is checking whether the number of linear analyses has reached a predefined maximal number [16]. However, this requires some a priori knowledge of the number analysis steps needed to reach the desired structural response. An alternative is to stop the analysis as soon as a predefined displacement is attained or to stop as soon as an element reaches a certain secant branch, meaning that the material stiffness has decreased with a certain percentage.

After scaling of the linear analysis with the critical load factor, the stopping criteria is checked. If the stopping criteria is not yet attained, the damage to the critical element is applied by

recalculating the reduced element stiffness matrix according to the adopted saw-tooth law. The new system of linear equations is then reassembled and the analysis is repeated.

5

Solution methods for linear systems

In Section 4 an overview of the event-by-event strategy of *SLA* was provided. To approximate the nonlinear material behaviour, the analysis of *SLA* requires repeated solving of a linear system of equations,

$$K\mathbf{u} = \mathbf{f}, \tag{5.1}$$

where the stiffness matrix K is large, sparse, symmetric positive definite, of dimension $n \times n$ and the solution vector $\mathbf{u}$ and force vector $\mathbf{f}$ are of dimension n. This section will distinguish between two classes of numerical solution methods; direct solution methods and iterative solution methods. Direct solution methods solve Equation (5.1) by matrix factorisation and obtain the solution $\mathbf{u}$ using forward elimination and back substitution. Iterative solution methods solve Equation (5.1) by constructing a sequence of successive approximations.

Section 5.1 will discuss direct solution methods and Section 5.2 Krylov subspace methods. These sections are based on the well-known works of Golub and van Loan, and Saad [7, 15]. The scope of this thesis limits itself to the two formerly mentioned solution classes. For a description of the class of multigrid methods, or more detailed descriptions on direct- or iterative solution methods the reader is referred to [7, 8, 15, 17].

5.1. Direct methods

Direct solution methods solve Equation (5.1) by factorisation of matrix K and applying forward elimination and backward substitution to obtain the solution $\mathbf{u}$. In this section, a detailed description of the most basic matrix factorisation will be presented which factorises the matrix as $K = LU$ with L a lower triangular matrix and U an upper triangular matrix. This type of factorisation will be referred to as the *LU* factorisation. It will be shown that under certain properties of the matrix K, the *LU* factorisation factorisation reduces to the more favourable Cholesky factorisation $K = CC^T$ with C a lower triangular matrix. The performance of a matrix factorisation depends on the dimension, bandwidth and conditioning of the matrix. Some remarks will be made on the latter two.

5.1.1. LU factorisation

Let K be a non-singular matrix of dimension $n \times n$ and assume all submatrices of K are non-singular. The general idea of the *LU* factorisation is to reduce K to upper triangular form by successive Gauss eliminations [1]. Each Gauss elimination performs row operations such that only 0's appear below the diagonal element. To formalize the process of Gauss elimination write Equation (5.1) as

$$K^{(1)}\mathbf{u} = \mathbf{f}^{(1)}\,, \tag{5.2}$$

with $k_{i,j}^{(1)}, f_i^{(1)}, u_i$ the elements of $K^{(1)}, \mathbf{f}^{(1)}, \mathbf{u}$ respectively.

The first Gauss elimination step ensures that only 0 values appear below the first diagonal element. Therefore, assume the first row has a non-zero pivot; $k_{1,1}^{(1)} \neq 0$. Under this assumption, the definition of the row multipliers is well-defined

$$m_{i,1} = \frac{k_{i,1}^{(1)}}{k_{1,1}^{(1)}}\,, \quad i = 2, \dots, n\,. \tag{5.3}$$

The row multipliers quantify how many times the first row has to be subtracted to the other $n-1$ rows in order to obtain 0 entries below the diagonal in the first column. The Gauss transform M_1 is then the matrix notation of the row multipliers

$$M_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -m_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -m_{n,1} & 0 & \cdots & 1 \end{pmatrix}. \tag{5.4}$$

Multiplication of the Gauss transform with $K^{(1)}$ eliminates u_1 from equations $2, \dots, n$. The resulting system of equations is then

$$K^{(2)}\mathbf{u} = \mathbf{f}^{(2)}\,, \tag{5.5}$$

where $K^{(2)} = M_1 K^{(1)}$ and $\mathbf{f}^{(2)} = M_1 \mathbf{f}^{(1)}$. By construction of M_1 it then holds that $K^{(2)}$ is of the form

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

with

$$\begin{aligned} k_{i,j}^{(2)} &= k_{i,j}^{(1)} - m_{i,1} k_{1,j}^{(1)}\,, \quad i,j = 2, \dots, n \\ f_i^{(2)} &= f_i^{(1)} - m_{i,1} f_1^{(1)}\,, \quad i = 2, \dots, n\,. \end{aligned}$$

The described Gauss elimination step is repeated $n-1$ times to obtain a sequence of Gauss transforms $M_1, \dots, M_{n-1}$ with the property that

$$M_{n-1} M_{n-2} \dots M_1 K = U\,, \tag{5.7}$$

with $M_1, \dots, M_{n-1}$ lower triangular matrices and U an upper triangular matrix. Using Equation (5.7), Equation (5.1) can be written as

$$U\mathbf{u} = M_{n-1}M_{n-2} \dots M_1\mathbf{f}. \tag{5.8}$$

By construction, the matrices $M_1, \dots M_{n-1}$ have a unit diagonal and a lower triangular. It can easily be verified by direct multiplication that the inverses M_i^{-1} are equal to M_i with all off-diagonal elements changed in sign. As a result, the inverses M_i^{-1} are lower triangular. Using $(M_{n-1}M_{n-2} \dots M_1)^{-1} = M_1^{-1} \dots M_{n-2}^{-1}M_{n-1}^{-1}$, Equation (5.8) can be written as

$$M_1^{-1} \dots M_{n-2}^{-1}M_{n-1}^{-1}U\mathbf{u} = \mathbf{f}. \tag{5.9}$$

Since the product of lower triangular matrices is again lower triangular, setting L as the product $L = M_1^{-1} \dots M_{n-2}^{-1}M_{n-1}^{-1}$ completely defines the *LU* factorisation of K such that Equation (5.1) can be written in the factored form

$$LU\mathbf{u} = \mathbf{f}, \tag{5.10}$$

with L a lower triangular matrix and U an upper triangular matrix.

In Section 5.1.5 it will be illustrated that such factorisation will prove convenient in solving a linear system of equations.

5.1.2. Cholesky factorisation

The LU factorisation exists whenever all submatrices of K are non-singular [1]. The LU factorisation can be written slightly different by factorising the matrix U further as $U = DM^T$ with $D = \mathrm{diag}(U)$ 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 matrix factorisation then becomes

$$K = LDM^T, \tag{5.11}$$

where L is lower triangular, D diagonal and M^T upper triangular. The factorisation from Equation (5.11) is referred to as the *LDM* factorisation.

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. Transforming the LU factoriastion to a LDM factorisation it immediately follows from the symmetry of K that $L = M$. Hence the LDM factorisation for *SPD* matrices reduces to a *LDL* factorisation

$$K = LDL^T. \tag{5.12}$$

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 factorisation of K can be written as the *Cholesky* factorisation:

$$K = CC^T, \tag{5.13}$$

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

5.1.3. Scaling and pivoting

In the derivation of the LU decomposition, every Gauss elimination step assumes a non-zero pivot. This assumption is not necessary when the rows are permuted such that the pivot element is non-zero. 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. To address these issues, the methods of *scaling*, *partial-* and *complete pivoting* are introduced.

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 [1]. 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 [5, 7].

In Gauss elimination, every step assumed a non-zero pivot. In order to omit this assumption and to avoid small pivot elements *partial pivoting* (row interchanges) can be applied prior to calculating the row multipliers from Equation (5.3). In the k-th step of Gaussian elimination, $1 \leq k \leq n-1$, define

$$c_k = \max_{k \leq i \leq n} |k_{i,k}^{(k)}| \,. \tag{5.14}$$

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 $K^{(k)}$ and $\mathbf{f}$. By construction it then holds that all row multipliers satisfy

$$|m_{i,k}| \leq 1 \,, \quad i = k+1, \ldots, n \,. \tag{5.15}$$

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 of Gaussian elimination the system of equations is given by

$$M_{n-1}P_{n-1}M_{n-2}P_{n-2} \ldots M_1P_1K\mathbf{u} = M_{n-1}P_{n-1}M_{n-2}P_{n-2} \ldots M_1P_1\mathbf{f} \,. \tag{5.16}$$

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

$$c_k = \max_{k \leq i,j \leq n} |k_{i,j}^{(k)}| \,. \tag{5.17}$$

Let i, j be the smallest row- and column indices such that the maximum for c_k is attained. If $i, j > k$ switch rows i, k of $K^{(k)}, \mathbf{f}$ and the columns i, j of $K^{(k)}$. 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 only to use partial pivoting.

When K is SPD (and hence its Cholesky decomposition exists) the use of scaling or pivoting is not necessary [1].

5.1.4. Matrix reordering

The discretisation of the structural equations using the finite element method typically results in a system stiffness matrix K that is large, symmetric and sparse. When solving the system of linear equations, it is desirable to exploit the sparseness property as it reduces the number of floating point operations to be performed. However, as a result of Gaussian elimination the sparsity pattern is often destroyed and *fill-in* occurs such that the resulting factors L, U contain significantly more non-zeros than K. The penalty to fill-in is twofold; additional storage is required for the non-zero values and extra floating point operations have to be performed when solving the system of linear equations. To illustrate the importance of preventing fill-in, consider a sparse stiffness matrix K of dimension 4500×4500 as shown in Figure 5.1.

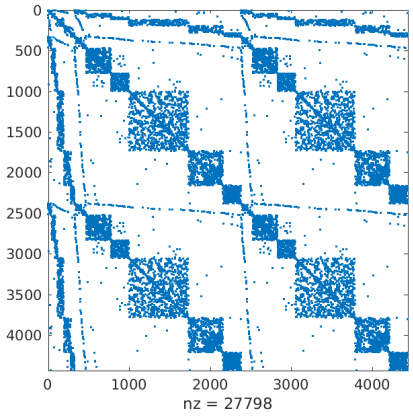


Figure 5.1: Example of a sparse stiffness matrix before reordering.

From the dimension and the number of non-zero elements of K it can be derived that only $\sim 0.14\%$ of the elements are non-zero. In Figure 5.2 the corresponding *LU* factorisation of K is shown.

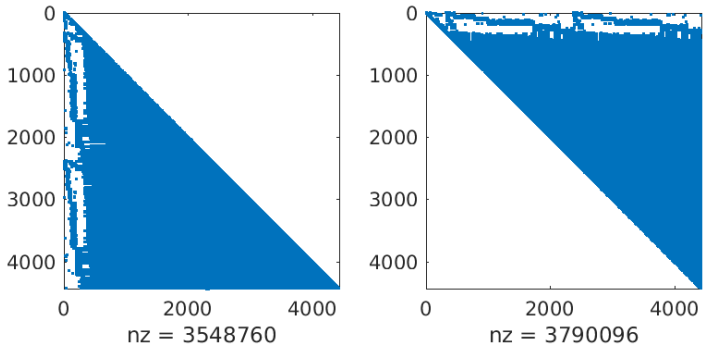


Figure 5.2: *LU* factorisation of a sparse stiffness matrix without reordering.

The total number of non-zero elements after factorisation increased dramatically resulting in a severe memory and performance penalty of the direct solution method. To motivate the importance of applying a reordering scheme prior to factorisation, the same matrix K is shown in Figure 5.3 after reordering.

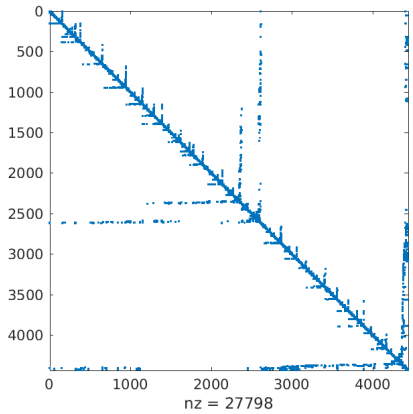


Figure 5.3: Example of a sparse stiffness matrix after *AMD* reordering.

As a result of the reordering scheme, the sparsity pattern has completely changed such that most non-zero elements are clustered around the main diagonal. In Figure 5.4 the corresponding *LU* factorisation is shown.

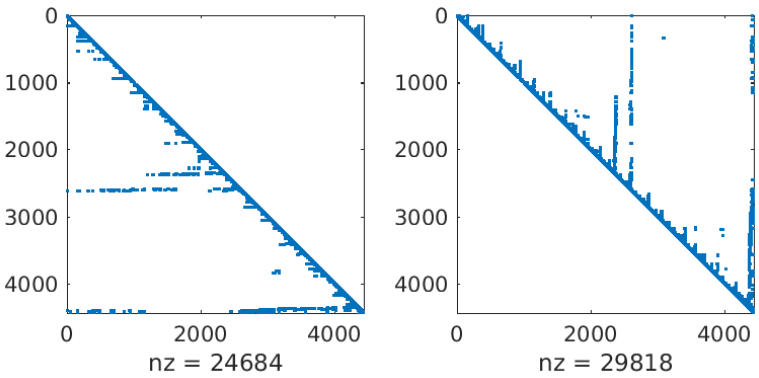


Figure 5.4: *LU* factorisation of a sparse stiffness matrix with *AMD* reordering.

From the number of non-zero elements it is clear that the factorisation after reordering has largely maintained the sparsity. 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 the product of permutation matrices P, Q with the matrix K

$$\left(PKQ^T\right)(Q\mathbf{u}) = P\mathbf{f}, \tag{5.18}$$

where P permutes the rows and Q the columns of K. Note that for symmetric matrices it must hold that $P = Q$ in order to maintain symmetry of the reordered matrix PKQ^T. Several reordering schemes to compute the permutations P, Q. Generally, two classes of reordering schemes can be distinguished. The first class of methods aim to minimise the bandwidth and exploit the fact that fill-in only occurs within this band. The second class aims to find a reordering which minimises fill-in. The bandwidth of the resulting reordered matrix does not necessarily decrease using the second class of reordering schemes. Finding permutations P, Q for which fill-in after factorisation is minimal is a *NP*-hard problem, meaning that no known efficient way of finding such permutations exists. Despite this, several well-known reordering heuristics exist. From each of the two classes, a widely used reordering scheme is briefly discussed.

Cuthill-McKee reordering

The Cuthill-McKee (*CM*) reordering scheme is a well-known reordering heuristic for reducing the bandwidth of sparse symmetric matrices [4]. 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 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 pseudocode from Algorithm 1.

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 [6]. 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.

Minimum degree reordering

The Minimum Degree (MD) algorithm is a well-known reordering scheme that can be used to reduce the occurrence of fill-in for 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*) and *Symmetric Approximate Minimum Degree* (*SYMAMD*).

5.1.5. Forward and back substitution

In Section 5.1.1 it was shown how Gaussian elimination can be applied to factorise a matrix K into the factors L, U such that Equation (5.1) can be written as

$$LU\mathbf{u} = \mathbf{f}\,, \tag{5.19}$$

with L a lower triangular matrix and U an upper triangular matrix. These favourable properties can be used to efficiently solve the system of equations. Introducing an auxiliary vector $\mathbf{y}$ of same size as $\mathbf{f}$, the system of equations can be written as

$$L\mathbf{y} = \mathbf{f}\,, \tag{5.20}$$

$$U\mathbf{u} = \mathbf{y}\,. \tag{5.21}$$

Due to the triangular properties of L and U, Equations (5.20) and (5.21) are efficiently solvable using *forward* and *back substitution* respectively. The general algorithm for *forward substitution* is shown in Algorithm 2 and for *back substitution* in Algorithm 3.

Algorithm 2 Forward substitution

for $i = 1, \ldots, n$ **do**
  $\mathbf{y}(i) = [\mathbf{f}(i) - L(i, 1 : i-1) \cdot \mathbf{y}(1 : i-1)] / L(i, i)$
end for

Algorithm 3 Backward substitution

for $i = n, \ldots, 1$ **do**
  $\mathbf{u}(i) = [\mathbf{y}(i) - U(i, i+1 : n) \cdot \mathbf{u}(i+1 : n)] / U(i, i)$
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

$$(L + F)\,\hat{\mathbf{y}} = \mathbf{f}\,, \tag{5.22}$$

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 (5.20) is numerically stable. For a proof of the numerical stability the reader is referred to Higham [8]. Analogously it can be proven that the back substitution from Equation (5.21) is also numerically stable.

The implementation of the direct solution method in *DIANA* uses Intel's Parallel Direct Sparse Solver Interface (*PARDISO*). This interface enables high-performance serial and parallel solving of sparse linear system of equations. One of the main advantages of *PARDISO* is its excellent parallel scaling, where speed ups of a factor of 7 have been realised when running on 8 threads[1].

5.2. Krylov subspace methods

The class of direct solution methods from Section 5.1.1 solve Equation (5.1) by factorisation of K. These methods can be impractical when K is large. In contrast to direct solution methods are Krylov subspace methods. These methods generate a sequence of approximate solutions by solving a minimisation problem over a particular type of subspace. Instead of factorisation, the matrix K is only involved in matrix-vector multiplications and Krylov subspace methods determine the solution to Equation (5.1) using inner-products, vector updates, scalar-vector and matrix-vector products. As a result, Krylov subspace methods require a relatively small amount of memory to solve the problem compared to direct solution methods. This makes Krylov subspace methods suited for solving problems where K is large.

As an introduction to Krylov subspace methods, first basic iterative methods (*BIM*) are briefly discussed. These methods are not efficient for solving Equation (5.1) but serve as building blocks for more advanced solution methods. The main step in the construction of a *BIM* is to define a splitting of K

$$K = P - R\,, \tag{5.23}$$

in which $P \in \mathbb{R}^{n\times n}$ is non-singular. Using the splitting, Equation (5.1) can be written as $P\mathbf{u} = R\mathbf{u} + \mathbf{f}$. Multiplying both sides of the equation with P^{-1} an iterative scheme can be derived as follows

$$\begin{aligned}
\mathbf{u}_{k+1} &= P^{-1}R\mathbf{u}_k + P^{-1}\mathbf{f} \\
&= P^{-1}\left(P - K\right)\mathbf{u}_k + P^{-1}\mathbf{f} \\
&= \left(I - P^{-1}K\right)\mathbf{u}_k + P^{-1}\mathbf{f} \\
&= \mathbf{u}_k + P^{-1}\left(\mathbf{f} - K\mathbf{u}_k\right) \\
&= \mathbf{u}_k + P^{-1}\mathbf{r}_k
\end{aligned} \tag{5.24}$$

Each iteration requires solving a linear system with matrix P. Hence, 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.

Some well-known choices for the preconditioner P exist. Two preconditioners that will be used in this thesis are discussed, the incomplete LU (*ILU*) decomposition and the complete decomposition.

[1] On artificial test problems.

Complete factorisation
Two extreme choices for preconditioners exist, $P = I$ or $P = K$. Taking $P = I$ the preconditioner is extremely cheap to compute and apply, however, the rate of convergence does not improve. On the other hand, taking $P = K$ the preconditioner is extremely expensive to compute since this requires a factorisation, however, convergence is reached within one iteration. Typically, a preconditioner is chosen somewhere in between these two extremes such that the applying and computing P is relatively cheap while the rate of convergence is also improved. However, due to the event-by-event strategy of *SLA* the choice $P = K$ can still be a good choice when not calculating the factorisation every analysis step.

ILU factorisation
One of main disadvantages of computing a factorisation of K is the fill-in the the factors L, U. Discarding some of these non-zero elements results in the *ILU* factorisation which computes the factors L, U such that

$$K = LU - R\,, \tag{5.25}$$

in which the matrix R indicates the error in the factorisation. The most basic choice for the *ILU* factorisation is to take the zero pattern of K and only calculate non-zero elements of L, U where the corresponding element of K is non-zero. This choice is referred to as zero fill-in (*ILU*(0)) as the number of non-zero elements of L, U combined is almost the same as that of K, only having to store an additional diagonal.

The accuracy of the *ILU*(0) may be inadequate to obtain an acceptable rate of convergence in the iterative scheme [15]. Therefore, it can be beneficial to include some level of fill-in which results in the *ILU*(p) factorisation where p-th order fill-ins are allowed. To illustrate this concept, consider L_0, U_0 to be the factors of the *ILU*(0) factorisation. Due to inaccuracy of the factorisation, the sparsity pattern of L_0U_0 will differ from that of K. The *ILU*(1) is then calculated by calculating only the non-zero elements of L_1, U_1 where the corresponding elements of L_0U_0 is non-zero. Applying this process recursively allows the computation of an increasingly more accurate incomplete factorisation.

The iterative scheme of (5.24) is repeated until the approximate solution is sufficiently accurate. To determine when the algorithm stops, a stopping criterion has to be defined. A commonly used stopping criterion is to iterate until

$$\frac{\|\mathbf{r_k}\|}{\|\mathbf{f}\|} \leq \varepsilon \tag{5.26}$$

for some ε. This stopping criterion is scaling invariant and the required accuracy does not depend on the initial guess. Higham [8] provides a detailed elaboration on this and more preferable stopping criteria.

Writing out some iterations of a *BIM* it follows that the approximate solution $\mathbf{u}_i$ belongs to the subspace

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

Definition 9. *Let $K \in \mathbb{R}^{n\times n}$, $\mathbf{v}^n$. The Krylov subspace $\mathcal{K}_i$ generated by K and $\mathbf{v}$ is defined as*

$$\mathcal{K}_i\left(K; \mathbf{v}\right) = \operatorname{span}\left\{\mathbf{v}, K\mathbf{v}, \ldots, K^{i-1}\mathbf{v}\right\}, \tag{5.27}$$

where

$$\forall \mathbf{x} \in \mathcal{K}_i\,(K;\mathbf{v})\ \exists \lambda_0, \dots, \lambda_{i-1} \in \mathbb{R} : \mathbf{x} = \lambda_0 \mathbf{v} + \lambda_1 K\mathbf{v} + \dots + \lambda_{i-1} K^{i-1}\mathbf{v}\,. \tag{5.28}$$

Hence, a *BIM* finds the i-th approximate solution in the subspace $\mathbf{u}_i \in \mathbf{u}_0 + \mathcal{K}_i\left(P^{-1}K; P^{-1}\mathbf{r}_0\right)$.

Krylov subspace methods generate a sequence of approximate solutions by solving a minimisation problem over a Krylov subspace. Since the Krylov subspace $\mathcal{K}_i$ is of dimension i, i constraints must be imposed to find an unique approximate solution. Typically, these constraints are imposed as independent orthogonality conditions, meaning that the residual $\mathbf{r}_i = \mathbf{f} - K\mathbf{u}_i$ is constrained to be orthogonal to i linear independent vectors.

Assuming K is symmetric, *Lanczos* algorithm can be used to construct an orthogonal basis $\{\mathbf{v}_1, \dots, \mathbf{v}_i\}$ for $\mathcal{K}_i$ using Algorithm 4.

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 loss of orthogonality. An overview of some of these improved orthogonalization schemes is provided by Saad [15].

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

$$\beta_{j+1}\mathbf{v}_{j+1} = K\mathbf{v}_j - \beta_j \mathbf{v}_{j-1} - \alpha_j \mathbf{v}_j\,. \tag{5.29}$$

Rearranging terms yields

$$K\mathbf{v}_j = \beta_j \mathbf{v}_{j-1} + \alpha_j \mathbf{v}_j + \beta_{j+1}\mathbf{v}_{j+1}\,. \tag{5.30}$$

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

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & 0 \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_m \end{pmatrix} \tag{5.31}$$

allows the short recurrences to be rewritten as

$$KV_m = V_m T_m + \beta_{m+1}\mathbf{v}_{m+1}\mathbf{e}_m^T . \tag{5.32}$$

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

$$V_m^T K V_m = T_m . \tag{5.33}$$

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 [15].

The Krylov subspace method of choice for solving large sparse SPD linear systems is the conjugate gradient (*CG*) method. The next section will provide an intuitive introduction to *CG*.

5.2.1. Conjugate Gradient

The stiffness matrix K from Equation (5.1) is symmetric and positive definite for the problems considered with *SLA*. For problems with this property, the Conjugate Gradient (*CG*) method is the Krylov subspace method of choice. This section will give the derivation of *CG* as well as how the method can be combined with preconditioners.

To derive *CG* the method of *steepest descent* is first introduced. Therefore, consider the solution to Equation (5.1) as the unique minimiser of the function $\phi(\mathbf{u})$

$$\phi(\mathbf{u}) = \frac{1}{2}\mathbf{u}^T K\mathbf{u} - \mathbf{u}^T\mathbf{f} . \tag{5.34}$$

Since it holds that $\nabla\phi(\mathbf{u}) = K\mathbf{u} - \mathbf{f}$ it follows that minimising Equation (5.34) is equivalent to solving Equation (5.1). The fastest way of finding the minimum is to follow the direction of steepest descent $-\nabla\phi(\mathbf{u})$. It holds that

$$-\nabla\phi(\mathbf{u}_k) = \mathbf{f} - K\mathbf{u}_k = \mathbf{r}_k , \tag{5.35}$$

hence we find a new approximate solution $\mathbf{u}_{k+1}$ from $\mathbf{u}_k$ by choosing a point along this *search direction*

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha\mathbf{r}_k . \tag{5.36}$$

The value α should be chosen such that it minimises $\phi(\mathbf{u}_{k+1})$ 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 it follows that

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

such that α should be chosen as

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

Using the search directions from Equation (5.36) convergence may be slow if K is badly conditioned [7]. To avoid this, *CG* minimises $\phi(\mathbf{u})$ along a set of search directions $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ that are linear independent, K-orthogonal and for which it holds $\mathbf{p}_k^T\mathbf{r}_{k-1} \neq 0$. A new approximate solution is then searched along these search directions

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k\mathbf{p}_k . \tag{5.37}$$

The energy norm of a vector $\mathbf{v}$ is defined as

$$\|\mathbf{v}\|_K = \left(\mathbf{v}^T K\mathbf{v}\right)^{\frac{1}{2}} , \tag{5.38}$$

and define the error $\delta\mathbf{u}_k$ in the k-th approximate solution as

$$\delta\mathbf{u}_k = \mathbf{u} - \mathbf{u}_k , \tag{5.39}$$

with $\mathbf{u}$ the exact solution. In Equation (5.37) α is chosen such that it minimises the energy norm of the error $\delta\mathbf{u}_k$ over the Krylov subspace $\mathcal{K}_{k-1}(K; \mathbf{r}_0)$, which can be shown to be equivalent to minimising Equation (5.34) [11]. The complete *CG* method is summarised 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: **for** $k = 0, 1, \dots$ until convergence **do**
3: $\quad \alpha_k = \frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{p}_k^T K\mathbf{p}_k}$
4: $\quad \mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k\mathbf{p}_k$
5: $\quad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k K\mathbf{p}_k$
6: $\quad \beta_k = \frac{\mathbf{r}_{k+1}^T\mathbf{r}_{k+1}}{\mathbf{r}_k^T\mathbf{r}_k}$
7: $\quad \mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
8: **end for**

Due to the fact that the search directions $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ are linear independent, *CG* guarantees convergence in at most n iterations. A well-known error-bound exists of the approximate solution [7]. Denote the i-th eigenvalue of K in nondecreasing order by λ_i.

Definition 10. *Let $K \in \mathbb{R}^{n\times n}$. The spectral condition number measured in the p-norm $\kappa_p(K)$ of K is defined as*

$$\kappa_p(K) = \|K\|_p \|K^{-1}\|_p .$$

After k iterations, the error is bounded by

$$\|\delta\mathbf{u}_k\|_K \leq 2\|\delta\mathbf{u}_0\|_K \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k . \tag{5.40}$$

In badly conditioned problems where κ is large the error reduction of *CG* is limited. A solution to this problem is applying a preconditioner to Equation (5.1) to improve the conditioning.

Preconditioning

The convergence speed of CG is highly dependent on the extreme eigenvalues of K. To improve the convergence behaviour of *CG* preconditioning can be applied to the problem to obtain more favourable eigenvalues. The first step in preconditioning is finding a preconditioner P such that the preconditioned problem is defined as

$$P^{-1}K\mathbf{u} = P^{-1}\mathbf{f}, \tag{5.41}$$

where P is assumed to be non-singular. The error bound of Equation (5.40) still holds for the preconditioned problem. As preconditioned *CG* converges fast when $\kappa\left(P^{-1}K\right) \approx 1$, ideally P should be chosen such that the eigenvalues of $P^{-1}K$ are clustered. The preconditioned *CG* (*PCG*) algorithm is shown in Algorithm 6.

Algorithm 6 Preconditioned conjugate gradient algorithm

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

Only one step is added to the *CG* algorithm to obtain *PCG*. Since in every iteration the linear system $P\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$ is solved, P should be chosen such that this is as cheap as possible. Furthermore, since we want the eigenvalues to be clustered, the matrix P should resemble K in some way. A popular choice for P is an incomplete *LU* factorisation (*ILU*) such that $LU \approx K$ in some sense, as described in Section 5.2. The accuracy with which the factorisation LU approximates K can be controlled by varying the allowed fill-in within the bandwidth of K using the drop tolerance. A good choice for the drop tolerance makes a trade-off between the increased computational cost of calculating the factorisation and the decrease in work of *PCG*. A detailed analysis on the choice of the drop tolerance can be found in [13].

Besides the error-bound in Equation (5.40), it is known that if K has r distinct eigenvalues convergence is guaranteed in at most $r + 1$ iterations [7]. This suggests the use of the complete factorisation of K as a preconditioner. Taking $P = K$, it follows that in the first analysis step $P^{-1}K = I$ and the solution method basically becomes a direct solution method. However, subsequent analysis steps require considerably less iterations due to the event-by-event nature of *SLA*. Therefore, the optimal point at which a new factorisation is calculated should be determined such that the computing times are minimised. In Section 7.2.4 some remarks are made on determining this restarting point. Section 8 then presents the results of preconditioned *CG* using the complete factorisation, as well as the results of Woodbury's identity, a method that will be introduced in Section 6.

6

Low-rank matrix update

In Section 4 a description of the event-by-event strategy of *SLA* was given. After every linear analysis, a scaling procedure is applied such that only one integration point reaches its peak stress limit. As a result of this strategy only one element stiffness matrix is updated between analysis steps resulting in a low-rank update to the system stiffness matrix K. The direct solution methods discussed in Section 5.1 do not exploit this property and compute the solutions by repeated factorisation of the K. This section will introduce a method that exploits the low-rank nature of *SLA* such that the stiffness matrix need not be factorised every analysis step but instead the solution is obtained by factorisation of a considerably smaller matrix along with some additional matrix and vector operations. First, Section 6.1 derives the method for rank-1 updates. Section 6.2 generalizes the method to rank-k updates for an arbitrary value k. Finally, some implementation details are given.

6.1. 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 1 (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 $A + \mathbf{u}\mathbf{v}^T$. Assuming the inverse A^{-1} is already known, this formula presents a computationally cheap way of determining the inverse of the rank-1 updated matrix without having to perform the computationally intensive inverse operation. The proof of Theorem 1 is given below.

Proof. ($\Leftarrow$): 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 exists 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 it follows that

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

where in the second equality it was used that $\mathbf{v}^TA^{-1}\mathbf{u}$ is a scalar. Using the assumption $1+\mathbf{v}^TA^{-1}\mathbf{u}\neq 0$, it follows that all fractions are well-defined. Similarly, for the left-multiplication the following holds.

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

This proves that $A+\mathbf{u}\mathbf{v}^T$ is invertible with $\left(A+\mathbf{u}\mathbf{v}^T\right)^{-1}=A^{-1}-\frac{A^{-1}\mathbf{u}\mathbf{v}^TA^{-1}}{1+\mathbf{v}^TA^{-1}\mathbf{u}}$.

($\Longrightarrow$): Now 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

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

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

The *Sherman-Morrison formula* can be used to efficiently compute the inverse of a rank-1 corrected matrix. However, in practical applications the rank correction is often off higher order. Hence, the next section will generalise the statement for arbitrary rank corrections.

6.2. 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 with a higher rank. 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 for an arbitrary value k.

Theorem 2 (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 generalisation of the *Sherman-Morrison formula*.

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

$$\left(A+UCU^T\right)^{-1} = A^{-1} - A^{-1}U\left(C^{-1}+U^TA^{-1}U\right)^{-1}U^TA^{-1}\,. \tag{6.1}$$

Next, the construction of the matrices U, C will be discussed in the context of the underlying finite element model and how the solution is calculated once the matrices are set-up.

6.2.1. Set-up and implementation

Since the construction of the matrices U, C of the *Woodbury identity* is dependent on the elements of the underlying finite element model, the following notation will be adopted.

Subscripts will be used to denote the element that the matrix corresponds to wheras superscripts will be used to indicate the analysis number. A matrix M corresponding to an element e_i in analysis step n will thus be denoted as $M_{e_i}^{(n)}$. Furthermore, the initial system stiffness matrix before any damage increments is denoted as $K^{(0)}$, and the low-rank corrected system stiffness matrix of the n-th analysis step as $K^{(n)}$.

Now assume some element e_i is damaged in the n-th analysis step. Using the introduced notation, the low-rank corrected system stiffness matrix can be written as

$$
\begin{aligned}
K^{(n+1)} &= K^{(n)} + N_{e_i}\left(K_{e_i}^{(n+1)} - K_{e_i}^{(n)}\right)N_{e_i}^T\\
&:= K^{(n)} + N_{e_i}D_{e_i}^{(n)}N_{e_i}^T\,,
\end{aligned}
\tag{6.2}
$$

where $D_{e_i}^{(n)}$ is the update to the element stiffness matrix of element e_i. The matrix N_{e_i} is the transformation matrix which maps the local numbering of the element to the global numbering.

Constructing the eigendecomposition of $D_{e_i}^{(n)}$

$$D_{e_1}^{(n)} = Q_{e_i}^{(n)} \Lambda_{e_i}^{(n)} \left(Q_{e_i}^{(n)}\right)^T , \tag{6.3}$$

Equation (6.2) can be written as

$$K^{(n+1)} = K^{(n)} + N_{e_i} Q_{e_i}^{(n)} \Lambda_{e_i}^{(n)} \left(Q_{e_i}^{(n)}\right)^T N_{e_i}^T . \tag{6.4}$$

The matrix $\Lambda_{e_i}^{(n)}$ is a diagonal matrix whose elements are the eigenvalues of $D_{e_i}^{(n)}$ and $Q_{e_i}^{(n)}$ contains the corresponding eigenvectors. In the above eigendecomposition, only the sufficiently large eigenvalues are considered and the small eigenvalues are discarded. Section 7 will discuss the effect of this choice on the convergence behaviour and discusses the exact choice of eigenvalues.

To bring Equation (6.2) in a suitable form for Woodbury's identity it is rewritten as follows

$$\begin{aligned} K^{(n+1)} &= K^{(n)} + N_{e_i} Q_{e_i}^{(n)} \Lambda_{e_i}^{(n)} \left(Q_{e_i}^{(n)}\right)^T N_{e_i}^T \\ &= K^{(n)} + \left(N_{e_i} Q_{e_i}^{(n)}\right) \Lambda_{e_i}^{(n)} \left(N_{e_i} Q_{e_i}^{(n)}\right)^T \\ &:= K^{(n)} + U^{(n)} C^{(n)} {U^{(n)}}^T \end{aligned} \tag{6.5}$$

Equation (6.5) is defined recursively. Writing out this recursion in terms of the initial stiffness matrix $K^{(0)}$ yields

$$\begin{aligned} K^{(n+1)} &= K^{(0)} + \sum_{j=1}^{n} U^{(j)} C^{(j)} {U^{(j)}}^T \\ &= K^{(0)} + \begin{bmatrix} U^{(1)} & \dots & U^{(n)} \end{bmatrix} \begin{bmatrix} C^{(1)} & & \\ & \ddots & \\ & & C^{(n)} \end{bmatrix} \begin{bmatrix} {U^{(1)}}^T \\ \vdots \\ {U^{(n)}}^T \end{bmatrix} \\ &:= K^{(0)} + U_n C_n U_n^T . \end{aligned} \tag{6.6}$$

Equation (6.6) is of the form as required by Theorem 2. Furthermore, by writing

$$U_n = \begin{bmatrix} U_{n-1} & U^{(n)} \end{bmatrix} , \tag{6.7}$$

$$C_n = \begin{bmatrix} C_{n-1} & \\ & C^{(n)} \end{bmatrix} , \tag{6.8}$$

it is clear that in every analysis step the rank increases with the number of eigenvalues of the eigendecomposition from Equation (6.3).

Assuming that the factorisation of $K^{(0)}$ is known and the above set-up is performed, the solution to the system of equations $K^{(n+1)}\mathbf{u} = \mathbf{f}$ of the $(n+1)$-th linear analysis step can be calculated by performing the following steps.

Step 1
Solve the system

$$K^{(0)}\mathbf{x} = \mathbf{f}, \tag{6.9}$$

for $\mathbf{x}$ by using the known factorisation of $K^{(0)}$.

Step 2
Solve the system

$$K^{(0)}Z = U_n\,, \tag{6.10}$$

for Z by using the known factorisation of $K^{(0)}$.

Step 3
Calculate

$$E = C_n^{-1} + U_n^T Z \tag{6.11}$$

Step 4
Calculate

$$\mathbf{y} = U_n^T\mathbf{x}\,. \tag{6.12}$$

Step 5
Solve the system

$$E\mathbf{z} = \mathbf{y} \tag{6.13}$$

for $\mathbf{z}$ by calculating a factorisation of E and applying subsequent forward and backward substitutions.

Step 6
Calculate the solution to the system of equations by

$$\mathbf{u} = \mathbf{x} - Z\mathbf{z}\,. \tag{6.14}$$

Instead of having to calculate a new costly matrix factorisation, Woodbury's identity allows for the reusage of an old factorisation to obtain the solution using some additional matrix and vector multiplications and solving a significantly smaller system of equations in step 5.

When solving repeated linear systems using Woodbury's identity two parameters can be tuned. In Equation (6.3) the eigendecomposition was taken of the stiffness matrix corresponding to the critical element. After this eigendecomposition, the rank increases with the number of sufficiently large eigenvalues. However, a decision has to be made what eigenvalues are considered sufficiently large. Considering too few eigenvalues can result in accuracy loss whereas too many may result in performance losses. Therefore the eigenvalues have to be chosen such that a balance is found. Furthermore, it is also possible to restart Woodbury's identity. This implies that a new factorisation of the stiffness matrix is computed and the rank is set back to 0. The penalty in restarting is having to recompute a costly matrix decomposition while on the other hand the following analysis steps are considerably cheaper than before restarting. The restarting point has be to be determined such that these effects are balanced. The next section provides a detailed analysis on the choice of these parameters.

7

Parametric analysis Woodbury's identity

The implementation of Woodbury's identity for solving low-rank corrected systems of linear equations enables the tuning of two parameters. Firstly, a choice is to be made on the size of the eigenvalues to be considered in the eigendecomposition of the critical element stiffness matrix. Secondly, the restarting point should be determined at which a new matrix factorisation is calculated. Since both these parameters have a direct influence on the performance and convergence of Woodbury's identity, it is important to choose these in such a way that the performance is optimized while convergence remains satisfactory. In this section, a parametric analysis is performed to obtain optimal values for both parameters.

7.1. Eigenvalue ratio

One of the first steps in the set-up of Woodbury's identity is calculating the eigendecomposition of the update to the critical element matrix as shown in Equation (6.3). Since the eigenvalues are numerically calculated using some iterative scheme, rounding errors can occur resulting in non-zero eigenvalues which should not be taken into account in the rest of the analysis. The methodology to determine which of the eigenvalues are sufficiently large to be considered in the analysis is as follows.

During each analysis step, the absolute value of all eigenvalues will be compared to that of the largest eigenvalue. If this ratio is above a certain threshold, the eigenvalue is taken to be large enough. This allows the choice of only dominant eigenvalues which differ a certain order of magnitudes from the largest value (in absolute sense).

In order to precisely define the selection of eigenvalues consider some element stiffness matrix of size n and let λ_i be the i-th eigenvalue, $i \in \{1, 2, \dots, n\}$. Furthermore, let $\lambda_{\text{max}} = \max\{|\lambda_1|, \dots, |\lambda_n|\}$. The ratio

$$\epsilon = \frac{|\lambda_i|}{\lambda_{\text{max}}} \tag{7.1}$$

will be referred to as the *eigenvalue ratio* corresponding to the i-th eigenvalue. Setting a minimal value for ϵ it is possible using this definition to consider eigenvalues which are only

a certain number of orders of magnitude smaller than the largest eigenvalue. For example, choosing $\epsilon > 10^{-4}$ all eigenvalues which are at most a factor 10^4 smaller than the largest eigenvalue are selected.

To analyse the influence of the choice of the eigenvalue ratio on the behaviour of the accuracy of the solution, three test problems will be considered as shown in Table 7.1. These examples will serve to empirically determine a proper value for the eigenvalue ratio ϵ. Subsequently, the resulting choice of eigenvalue ratio will be validated using a wide range of test problems.

Table 7.1: Test problems to empirically determine the eigenvalue ratio.

Problem	Size	DOF per element	Description
Intfnp	753	8	3-point bending beam
Lowthe	3268	16	Masonry shear wall
4PB2np	1009	8	4-point bending beam

In the event-by-event strategy of *SLA*, a relatively small number of elements is damaged repeatedly. As a result, the eigenvalue spectrum of the difference in element stiffness matrix is similar between analysis steps. For each of the mentioned test problems the (absolute) eigenvalue spectrum of such a typical analysis step is shown in Figure 7.1.

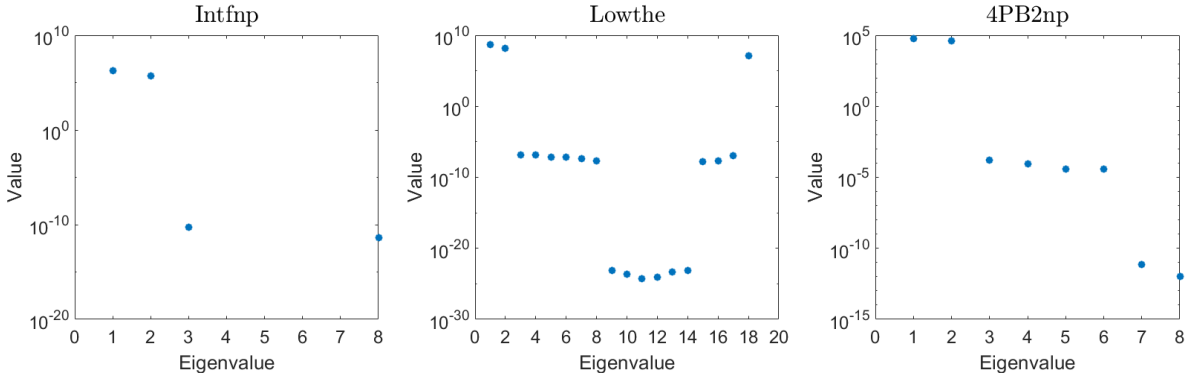


Figure 7.1: Absolute eigenvalue spectrum for the test problems of Table 7.1.

These three spectrums show similar results; a few dominant eigenvalues along with numerous significantly smaller eigenvalues. To analyse the effect of the choice of eigenvalues on the accuracy of the solution, several different thresholds for the eigenvalue ratio will be considered. To indicate what eigenvalues from Figure 7.1 will be taken into account using these thresholds, similar eigenvalue spectrums will be given for each chosen threshold indicating the sufficiently large eigenvalues with red. After the thresholds are chosen, the problem is solved using Woodbury's identity with the respective eigenvalue threshold. In these examples, Woodbury's identity will not be restarted in order to isolate the effect of the eigenvalues on the accuracy of the results.

For the *Intfnp* example, two thresholds are chosen based on the clustered behaviour of the eigenvalues in the first plot of Figure 7.1. The chosen eigenvalue ratio thresholds and the corresponding eigenvalue spectrums are shown in Figure 7.2.

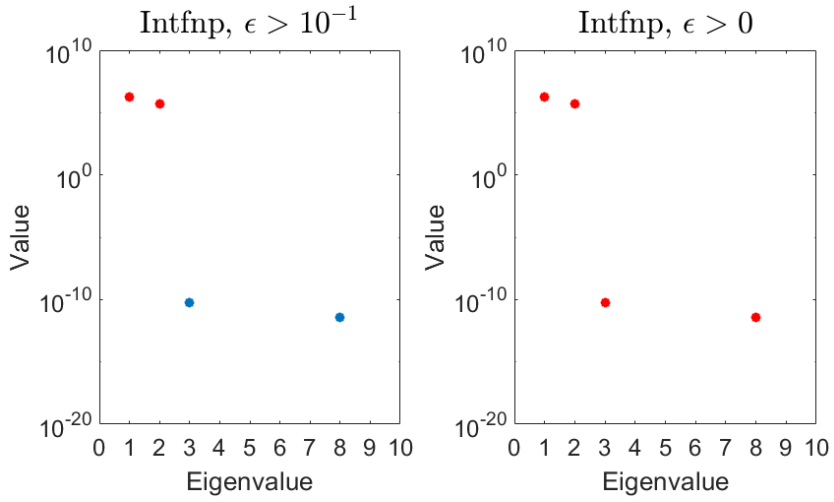


Figure 7.2: Selected eigenvalues (red) of the *Intfnp* problem for different eigenvalue ratios. Note that several 0 eigenvalues are not shown.

For each of these thresholds the problem was solved. Figure 7.3 shows the residual norms per analysis step for the *Intfnp* problem along with the residual norms when using the default direct solution method.

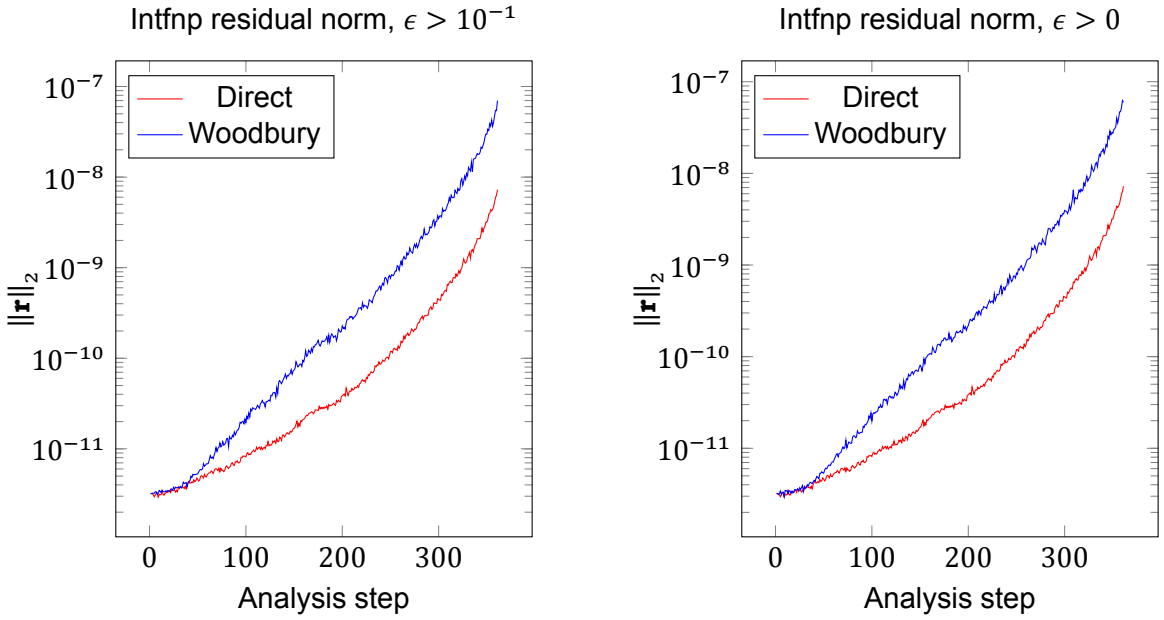


Figure 7.3: Residual norms of the *Intfnp* problem for the different eigenvalue thresholds. Accuracy of the standard direct method indicated with red.

Comparing the graphs in both plots, it can be observed that the residual norm of Woodbury's identity increases faster than that of the direct method. At the end of the analysis, the residual norms of Woodbury's identity method are approximately one order of magnitude larger. This is a promising result considering the fact that Woodbury's identity was not restarted in these analyses, which resets the residual norm back to the direct method's value as will be illustrated in Section 7.2. Furthermore, comparing both thresholds it follows that the extremely small

eigenvalues from Figure 7.2 have little effect on the accuracy of the solution.

Similar figures have been made for the other two test problems. In Figure 7.4 the chosen thresholds and corresponding eigenvalues of the *Lowthe* example are shown and Figure 7.5 shows the corresponding residual norms for every analysis step.

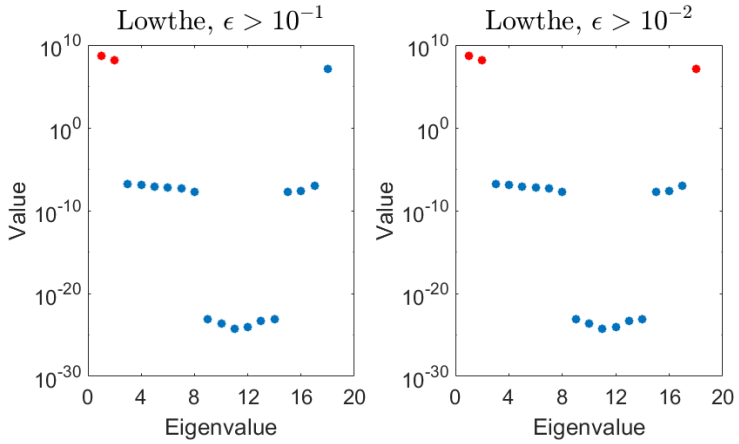


Figure 7.4: Selected eigenvalues (red) of the *Lowthe* problem for different eigenvalue ratios.

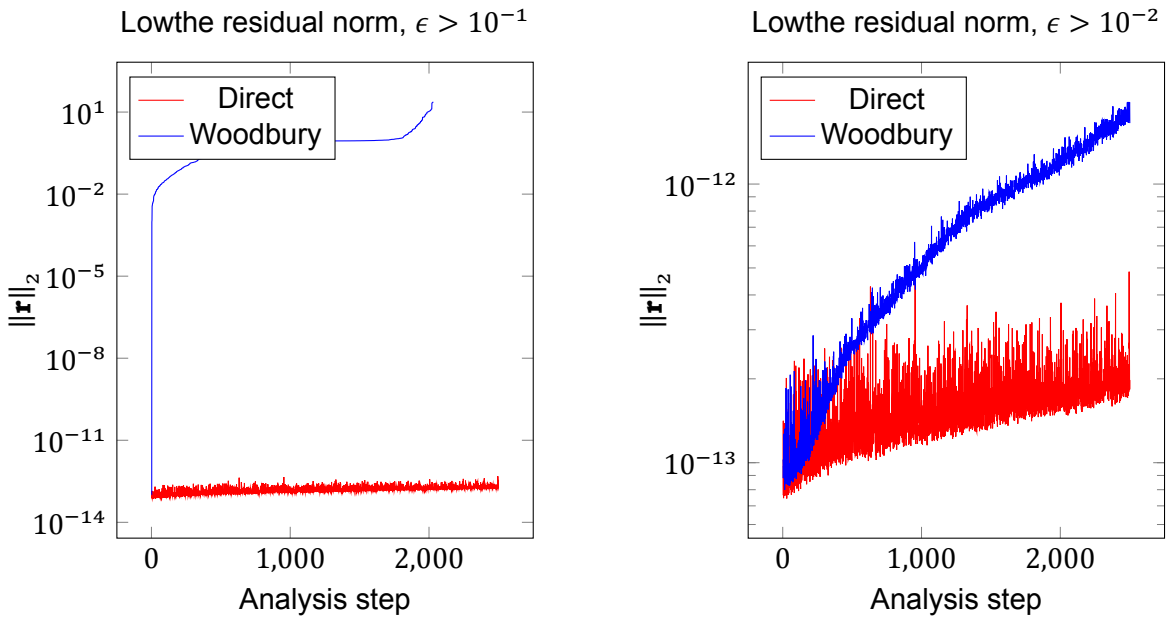


Figure 7.5: Residuals norms of the Lowthe problem for the different eigenvalue thresholds. Accuracy of standard direct method indicated with red.

From Figure 7.4 it follows that with the threshold $\epsilon > 10^{-1}$ some of the dominant eigenvalues are discarded. Looking at the corresponding residual norms in Figure 7.5 it can be seen that the accuracy of the solution is lost and as a consequence the analysis is stopped prematurely. When the threshold is increased such that all dominant eigenvalues are taken into account, the analysis produces accurate results as can be seen in the last plot of Figure 7.5. Similarly to Figure 7.3, the residual norms in Figure 7.5 increase faster for Woodbury's identity than using a direct solution method and at the end of the analysis the residual norms differ approximately

one order of magnitude. Especially considering that the *Lowthe* problem has significantly more analysis steps than the previous problem and no restarting is used these results are promising. Furthermore, again it was observed that including more smaller eigenvalues had no effect on the accuracy of the solution while significantly increasing computational times. Since the residual norms where approximately equal to those in the right-hand side of Figure 7.5, these figures are not shown.

Lastly the test problem *4PB2np* is considered. The chosen eigenvalue thresholds and the corresponding selected eigenvalues are shown in Figure 7.6 and the resulting residual norms can be seen in Figure 7.7.

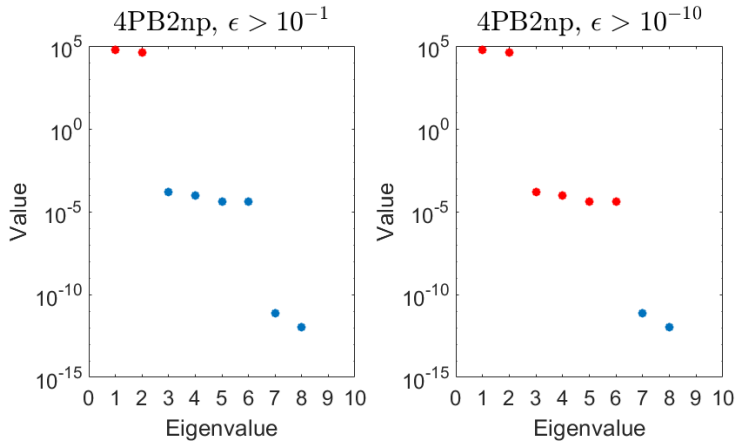


Figure 7.6: Selected eigenvalues (red) of the *4PB2np* problem for different eigenvalue ratios.

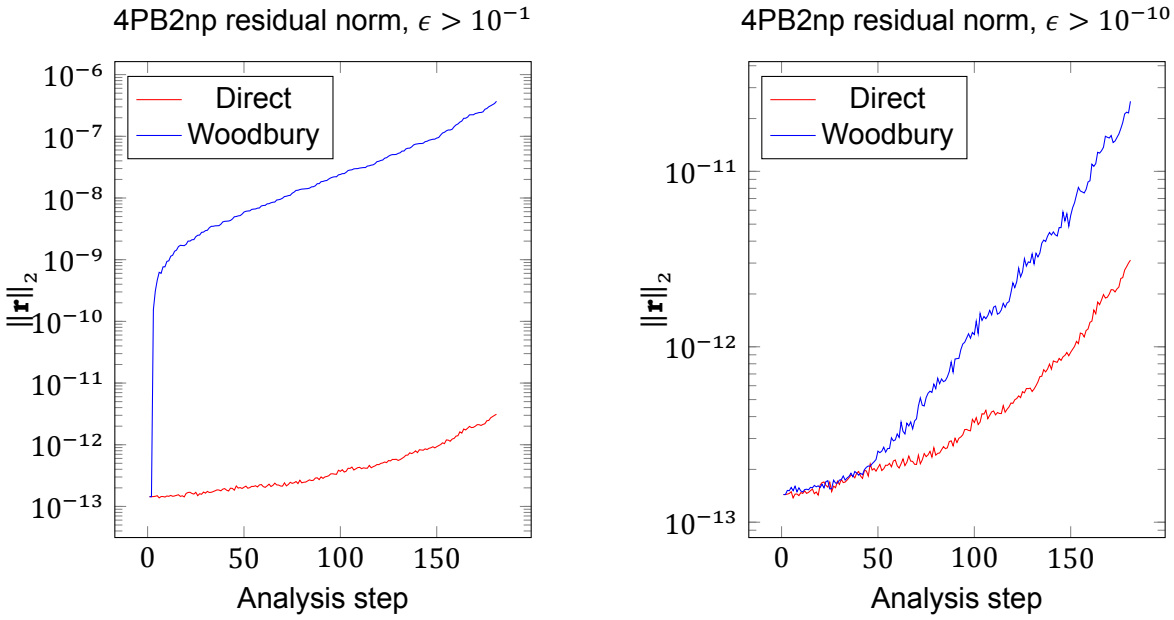


Figure 7.7: Residual norms of the 4PB2np problem for the different eigenvalue thresholds. Accuracy of standard direct method indicated with dashed lines.

In Figure 7.6 it can be seen that for the threshold $\epsilon > 10^{-1}$ both the largest eigenvalues are selected. However, the corresponding residual norms in Figure 7.7 are significantly higher than

for the direct method. When the threshold is reduced to $\epsilon > 10^{-10}$ such that also some of the smaller eigenvalues are selected, as shown in the second plot of Figure 7.6, the convergence behaviour is similar to that of the previous test problems. Similarly, reducing the threshold even further to include more eigenvalues barely affects the accuracy of the solution while increasing the computational times.

From these three test problems it follows that an eigenvalue ratio of 10^{-10} is sufficient to obtain accurate numerical results even without restarting Woodbury's identity. This implies that eigenvalues at least up to 10 orders of magnitude smaller than the largest eigenvalue have to be taken into account in the eigendecomposition of Equation (6.3). This empirical choice of the eigenvalue ratio is based only on three test problems. To further validate that this choice is correct, Section 7.1.1 will consider a wide range of different test problems.

7.1.1. Validation

In the previous section the eigenvalue ratio threshold $\epsilon > 10^{-10}$ was derived such that the results of Woodbury's identity showed similar convergence behaviour as the direct solution method. This derivation was based only on three test problems. This section will consider six more problems which will serve as a validation for the choice of the proposed eigenvalue threshold. The validation examples that will be used are shown in Table 7.2 along with some general information on these problems.

Table 7.2: Validation problems for the eigenvalue ratio.

Validation problem	Size	DOF per element	Description
Slab	2403	24	Reinforced concrete slab
4PB2pr	1009	8	4-point bending beam
Ifshnp	753	20	3-point bending beam
SG2_B1_10mm	2301	8	3-point bending nodged beam
Shrwal	1560	16	Shear wall

For each of the validation examples mentioned in Table 7.2 the problem was solved both using Woodbury's identity using the proposed eigenvalue threshold and the direct solution method. The solution residual norms of both solution methods for the given validation problems are shown in the Figures 7.8 - 7.12.

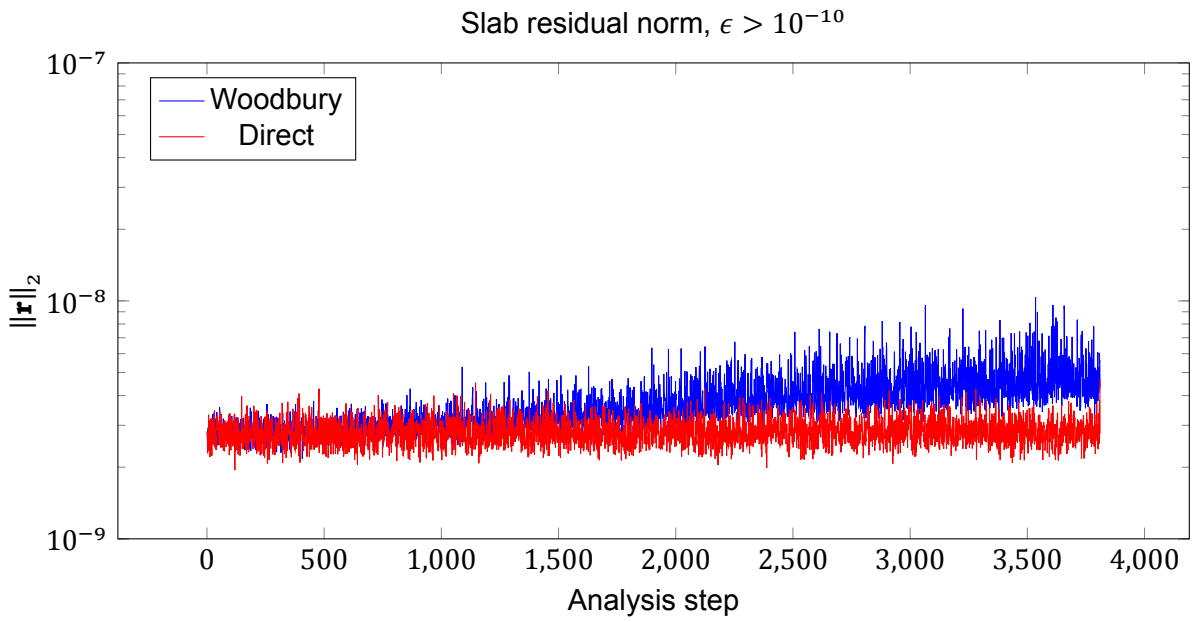


Figure 7.8: Residual norms of the *Slab* problem without restarting.

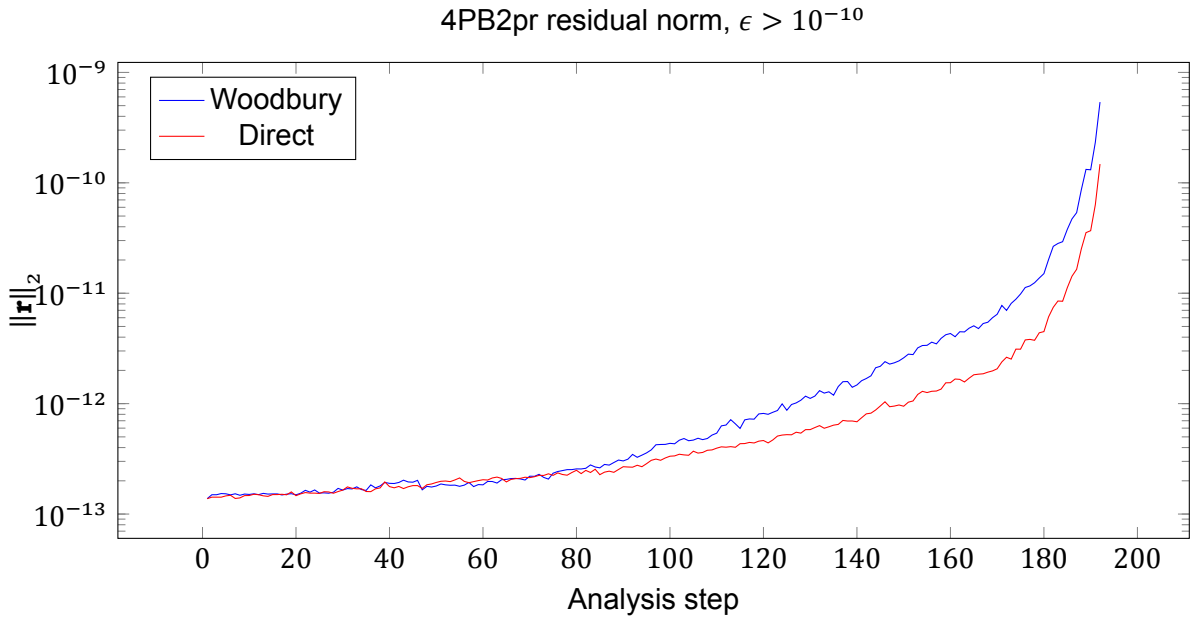


Figure 7.9: Residual norms of the *4PB2pr* problem without restarting.

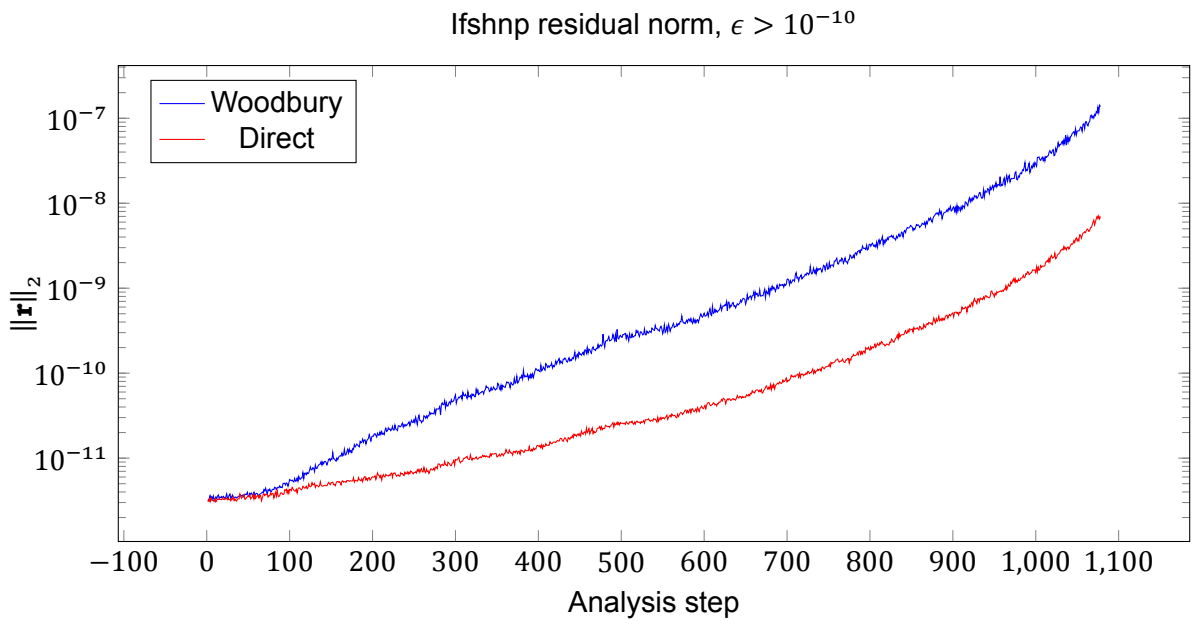


Figure 7.10: Residual norms of the *lfshnp* problem without restarting.

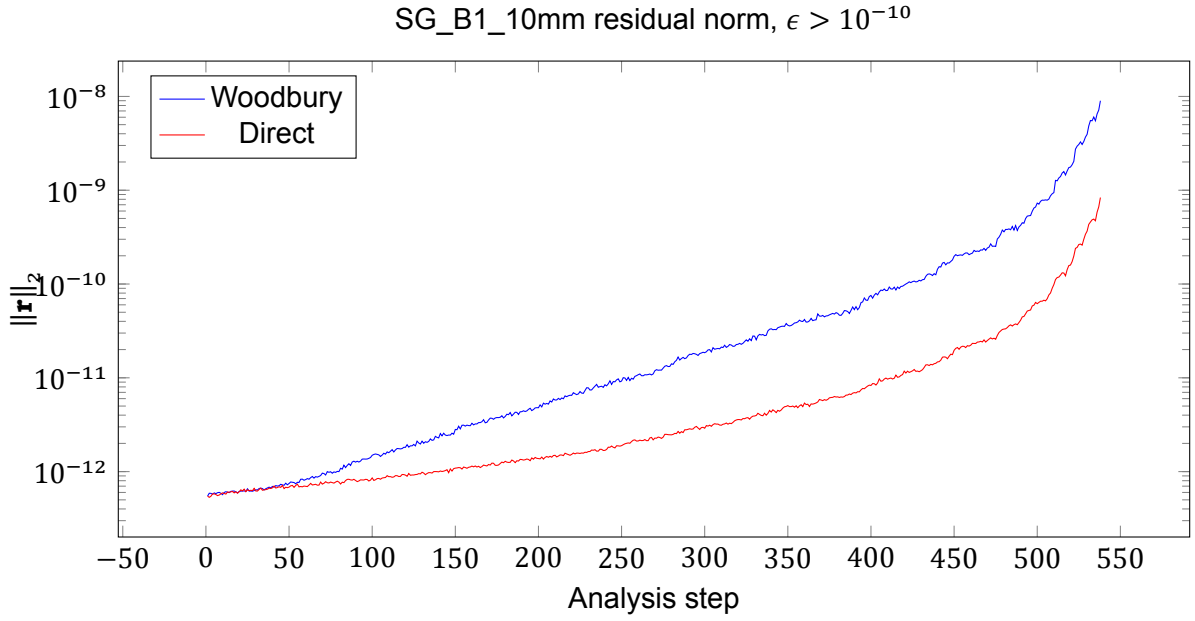


Figure 7.11: Residual norms of the *SG2_B1_10mm* problem without restarting.

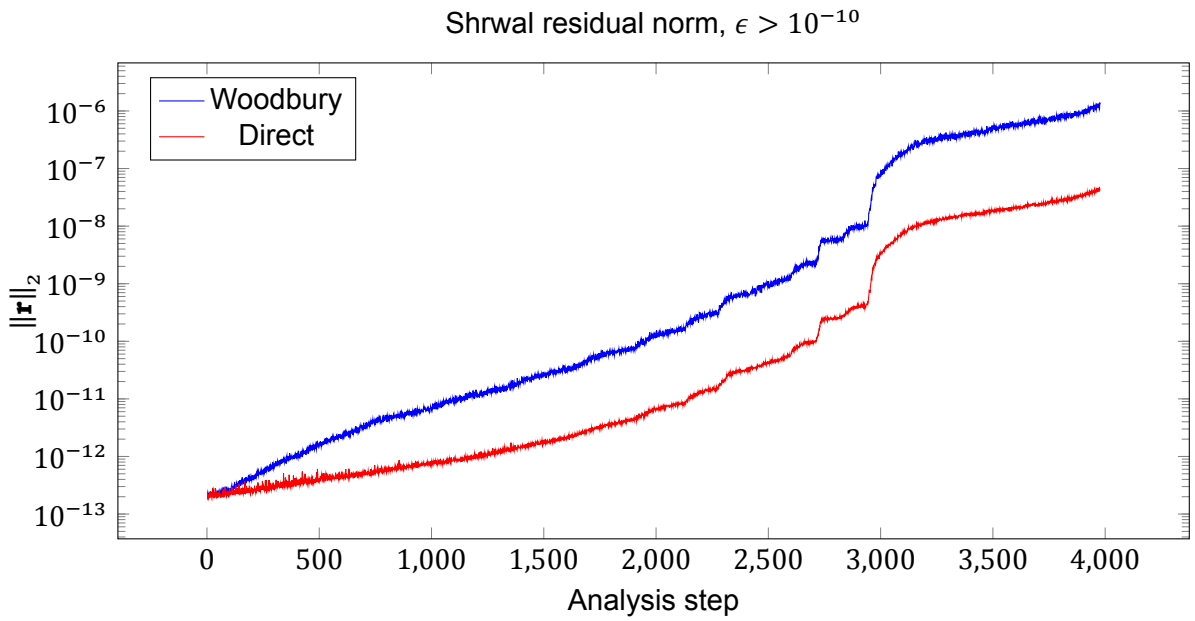


Figure 7.12: Residual norms of the *Shrwal* problem without restarting.

As can be seen in Figures 7.8 - 7.12, the solution residual norms of Woodbury's identity behaviour similarly to those of the direct solution method where the former increase slightly faster than those of the direct solution method, as was also seen in Section 7.1. In all of the above validation examples the solution residual of Woodbury's identity remains acceptably close to those of the direct solution method especially considering that no restarting of Woodbury's identity was applied. Besides the validation problems from Table 7.2, the eigenvalue ratio is also validated against the test suit present in *DIANA*. This test suit comprises of a wide range of test problems. The choice of $\epsilon > 10^{-10}$ also resulted in convergent solutions for all these problems, further validating this choice.

The choice for the eigenvalue ratio threshold $\epsilon > 10^{-10}$ resulted in convergent solutions for all problems that were considered. This serves as a motivation for the choice of the eigenvalue threshold. All of these problems used Woodbury's identity without recomputing the factorisation of the system stiffness matrix and setting the rank back to 0. In Section 7.2 an approach will be derived for finding the optimal restarting point. However, first some remarks will be made in Section 7.1.2 on the observed increase of the residual norms for the direct solution method.

7.1.2. Condition number estimation

In most of the considered examples it was observed that the norm of the residual increases even the direct solution method. This suggests that the condition number of the stiffness matrix of Equation (5.1) increases. Unfortunately, computation of the condition number is expensive as it requires a norm of K^{-1} to be calculated. Luckily, several techniques exist to estimate the condition number. To this extend, the 1-norm will be considered such that the condition number is defined as

$$\kappa_1(K) = \left\|K\right\|_1 \left\|K^{-1}\right\|_1 .$$

Calculation of the $\|K\|_1$ can be performed directly from the definition by calculating the maximum absolute column sum

$$\|K\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |k_{ij}| \,. \tag{7.2}$$

Defining the system of equation

$$K\mathbf{z} = \mathbf{y} \,,$$

a lower-bound for $\|K^{-1}\|$ can be derived as follows

$$\begin{aligned} & \mathbf{z} = K^{-1}\mathbf{y} \\ \Leftrightarrow \quad & \|\mathbf{z}\|_1 = \|K^{-1}\mathbf{y}\|_1 \le \|K^{-1}\|_1 \, \|\mathbf{y}\|_1 \\ \Leftrightarrow \quad & \frac{\|\mathbf{z}\|_1}{\|\mathbf{y}\|_1} \le \|K^{-1}\|_1 \,. \end{aligned} \tag{7.3}$$

Ideally, the vector $\mathbf{y}$ is chosen such that the lower-bound of Equation (7.3) is as tight as possible. A basic heuristic [9] is to choose $\mathbf{y}$ as follows. Firstly, $\mathbf{y}$ should be a fast-varying vector such that $\|\mathbf{z}\|_1$ is as large as possible. Secondly, $\mathbf{y}$ should be chosen such that $\|\mathbf{y}\|_1$ is as small as possible, resulting in the bound in Equation 7.3 to be as tight as possible. A choice for $\mathbf{y}$ that has these two properties is

$$y_i = (-1)^{i+1} \,, \qquad i \in \{1, \dots, n\} \,. \tag{7.4}$$

Using this choice for $\mathbf{y}$ and calculating $\|K\|_1$ directly, it is possible to calculate a lower-bound of $\kappa_1(K)$. The resulting condition number estimates corresponding to Figures 7.3 and 7.7 are shown in Figure 7.13.

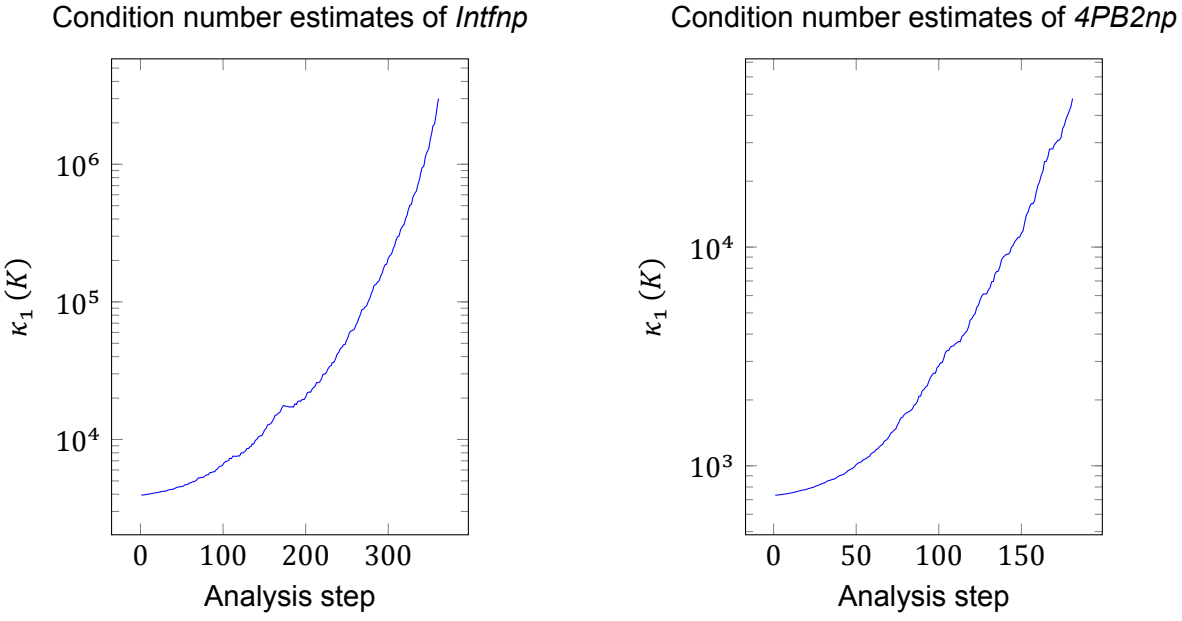


Figure 7.13: Conditioning number estimates of the *Intfnp* and *4PB2np* test problems.

The solution obtained with the direct solution method in Figures 7.3 and 7.7 have a relatively large increase in the residual norm. This is also reflected in the corresponding condition

number estimates in Figure 7.13. This implies that the stiffness matrix becomes increasingly ill-conditioned for these problems. As a result, accuracy is increasingly lost for these problems. The increased ill-conditioning of the problem can be attributed to the small number of elements of these problems. Once an element reaches complete failure, the system of equations is no longer properly defined as a result of the resulting singularity. A good illustration of this can also be seen in Figure 7.12 where around analysis step 2800 the residual norm increases sharply. At this point, the model reaches failure resulting in a singularity in the system of equations.

Furthermore, in Figures 7.3 and 7.7 it was observed that the solution residuals of Woodbury's identity increases faster than those of the direct solution method. This can be attributed to the numerous matrix and vector manipulations necessary in Woodbury's identity to obtain the solution. The rounding-errors resulting from all these operations add up and result in the increased residual norm.

7.2. Restarting approaches

In Section 7.1 and its subsections, the influence of the eigenvalue ratio threshold on accuracy of the solution was investigated. To isolate the effect of the eigenvalue ratio, restarting of Woodbury's identity was left out of consideration. This section will present two restarting approaches which aim at minimising the analysis time. The first method is based on theoretical estimates of the costs of Woodbury's identity and matrix factorisation and provides a rank at which Woodbury's method should be restarted. The second method is based on actual measured times and is less dependent on theoretical assumptions.

7.2.1. Rank-based restarting

The first restarting approach provides a restarting point based on the rank in Woodbury's identity. To this extend, estimates are required for the costs of both Woodbury's identity and the direct solution method. In order to derive these costs, consider a system stiffness matrix $K \in \mathbb{R}^{n\times n}$ with lower- and upper-bandwidths p, q. Let $X, F \in \mathbb{R}^{n\times r}$ be the solution- and force vectors respectively where r is the number of right-hand side vectors[1]. The system of equations to be solved is then defined as

$$KX = F\,.$$

Furthermore, assume that the stiffness matrix K has been factored as in Equation (6.6)

$$K = K_{\text{init}} + UCU^T\,,$$

with $U \in \mathbb{R}^{n\times k}$ and k the rank. The theoretical flop counts of the direct solution method and Woodbury's identity can then be derived by summing the costs of all necessary steps. The costs of the steps for the direct solution method are shown in Table 7.3, and for Woodbury's identity in Table 7.4.

[1] It is possible that multiple load cases are considered. For example, self-weight of a model and an external force.

Table 7.3: Flop count of direct solution method [7].

Operation	flop count
Factorisation	$2npq$
Back substitution	$2nqr$
Forward substitution	$2npr$

The values n, p, q, r represent properties of the model and are thus constant. As a result, the cost of the direct solution method is constant and will be denoted with f_d,

$$f_d = 2n\left(pq + r(p+q)\right) . \tag{7.5}$$

Table 7.4: Flop count of Woodbury's identity. The steps refer to the solution steps presented in Section 6.2.1. Costs are taken from [7].

Operation		flop count
Step 1	Forward substitution	$2npr$
	Back substitution	$2nqr$
Step 2	Forward substitution	$2npk$
	Back substitution	$2nqk$
Step 3	Sparse-dense mat-mat multiplication	$2nk^2$
	Vector update	k
Step 4	Sparse-dense mat-mat multiplication	$2nkr$
Step 5	Factorisation	$\frac{2}{3}k^3$
	Forward substitution	k^2r
	Back substitution	k^2r
Step 6	Mat-mat multiplication	$2nkr$
	Vector update	nr

The rank k is not constant as it grows with the number of analysis steps. Therefore, the cost function of Woodbury's identity will be denoted with $f_w(k)$,

$$f_w(k) = \frac{2}{3}k^3 + (2n+2r)\,k^2 + (2n(p+q)+1+4nr)\,k + 2nr(p+q) + nr\,. \tag{7.6}$$

The lower- and upper-bandwidths p, q are the bandwidths of the system stiffness matrix after reordering. Since the reordered matrix is never explicitly formed, approximations for p, q are used based on the geometry of the problem. For two-dimensional rectangular problems an estimate for both p, q is given by direction with the least number of degrees of freedom ($p = q = \sqrt{n}$ for square geometries), and for three-dimensional problems an estimate is given by the product of the two directions with the least number of DOF[2].

To determine the optimal restarting point assume first for simplicity that the rank increases with 1 every analysis step and let m be the maximal number of analysis steps. An example of restarting at analysis number t is shown in Figure 7.14.

[2]Especially for complex 3-dimensional geometries this method poorly approximates the bandwidth.

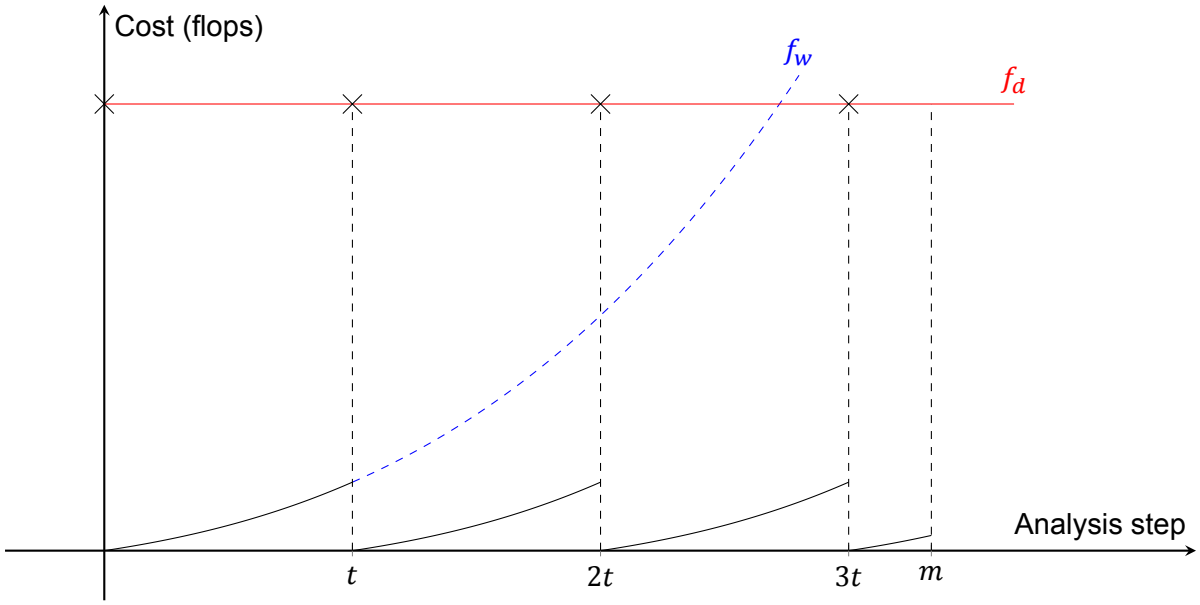


Figure 7.14: Example for finding restart point t. Assumption: rank increases with 1 per iteration.

From Figure 7.14 a function can be derived which determines the total cost of using Woodbury's identity as a function of restarting at analysis step t assuming that the rank increases with 1 every analysis step. Defining the indicator function as

Definition 11. *The indicator function* $\mathbb{1}_{\{f(x)\neq y\}}(x)$ *is a function defined as*

$$\mathbb{1}_{\{f(x)\neq y\}}(x) = \begin{cases} 1 & \text{if} \quad f(x) \neq y \\ 0 & \text{else} \end{cases} \tag{7.7}$$

The cost for restarting at t is then calculated as

$$c(t) = \left\lfloor \frac{m}{t+1} \right\rfloor \cdot \left(f_d + \sum_{i=1}^{t} f_w(i) \right) + \mathbb{1}_{\left\{ m - \left\lfloor \frac{m}{t+1} \right\rfloor \cdot (t+1) \neq 0 \right\}}(t) \cdot \left(f_d + \sum_{j=1}^{m - \left\lfloor \frac{m}{t+1} \right\rfloor \cdot (t+1) - 1} f_w(j) \right) . \tag{7.8}$$

The indicator function in Equation (7.8) is needed to adjust for the remaining analysis steps after the last restart as can be seen in Figure 7.14. For example, with $m = 100$ and restarting at $t = 2$, after 33 restarts there is still one analysis step left for which the second term in the right-hand side of Equation (7.8) corrects.

The cost function of Equation (7.8) assumes that the rank increases with 1 every analysis step such that f_w has to be evaluated for all integers up to t. However, in most problems the rank increases with more than 1. As a result, not all values of f_w have to be evaluated and the cost function of Equation (7.11) has to be adjusted accordingly. Therefore, assume that the rank increases with an arbitrary value Δk each analysis step. Under this assumption the following relation holds between the rank k and the analysis number t

$$k = \Delta k t \, . \tag{7.9}$$

The analysis step t from Equation (7.8) can then be substituted out by diving both sides of Equation (7.9) by n and taking the floor function due to t being an integer value

$$t = \left\lfloor \frac{k}{\Delta k} \right\rfloor . \tag{7.10}$$

Substitution of Equation (7.10) into Equation (7.8), the cost function for Woodbury's identity as a function of the rank-based restarting point k becomes

$$\begin{aligned} c(k) = &\left\lfloor \frac{m}{\left\lfloor \frac{k}{\Delta k} \right\rfloor + 1} \right\rfloor \cdot \left(f_d + \sum_{i=1}^{\left\lfloor \frac{k}{\Delta k} \right\rfloor} f_w(i) \right) \\ &+ \mathbb{1}_{\left\{ m - \left\lfloor \frac{m}{\left\lfloor \frac{k}{\Delta k} \right\rfloor + 1} \right\rfloor \cdot \left(\left\lfloor \frac{k}{\Delta k} \right\rfloor + 1 \right) \neq 0 \right\}}(k) \cdot \left(f_d + \sum_{j=1}^{m - \left\lfloor \frac{m}{\left\lfloor \frac{k}{\Delta k} \right\rfloor + 1} \right\rfloor \cdot \left(\left\lfloor \frac{k}{\Delta k} \right\rfloor + 1 \right) - 1} f_w(j) \right) . \end{aligned} \tag{7.11}$$

The optimal rank-based restarting point is the value k for which Equation (7.11) is minimised. This minimisation problem relies heavily on the estimated bandwidth of the stiffness matrix K. These estimates are used to estimate the flop counts f_d, f_w of respectively the direct and Woodbury's identity solution steps. For simple 2- and 3-dimensional geometries these bandwidths can be accurately estimated. However, for complex geometries this is no longer efficiently possible. Furthermore, the reordering applied in the PARDISO direct solver is a fill-in minimising reordering scheme which does not necessarily minimise the bandwidth. It is therefore likely that the direct solution method flop counts in Table 7.3 are inaccurate. As a result of the above, minimising Equation (7.11) is unlikely to predict an optimal restarting point.

Instead of relying on estimates of the system matrix bandwidth for estimating the flop counts $f_d and f_w$, it is possible to measure the actual required computing times. The next section will describe how a similar restarting strategy can be derived using measured computing times which do not rely on estimating the stiffness matrix bandwidth or the rank increase.

7.2.2. Time-based restarting

The rank-based restarting strategy of Section 7.2.1 relies on the estimation of the stiffness matrix bandwidth to calculate flop counts for both solution methods as well as it assumes a constant increase in rank. Due to complex geometries and the fill-in reducing nature of the PARDISO reordering scheme, these estimates are inaccurate. Therefore, a different restarting strategy will be proposed which does not rely on these estimates.

To this extend, the computing times of the analysis steps are measured and from these times an estimate is calculated for the expected total time. As an illustration, consider an analysis with a maximum number of analysis steps m. The analysis starts with a direct solution step (factorisation, back- and forward substitutions) which time is measured and denoted with t_d. The subsequent analysis steps, say n, obtain the solution using Woodbury's identity and the

respective times are denoted with $t_w(i)$ where $i = 1, \dots, n$. Restarting after the n-th analysis step and assuming that the measured sequence of times $\{t_d, t_w(1), \dots, t_w(n)\}$ repeats until the end of the analysis, the total computing time of the analysis can be computed similarly to Equation (7.8)

$$t(n) = \left\lfloor \frac{m}{n+1} \right\rfloor \cdot \left(t_d + \sum_{i=1}^{n} t_w(i) \right) + \mathbb{1}_{\left\{ m - \left\lfloor \frac{m}{n+1} \right\rfloor \cdot (n+1) \neq 0 \right\}}(n) \cdot \left(t_d + \sum_{j=1}^{m - \left\lfloor \frac{m}{n+1} \right\rfloor \cdot (n+1) - 1} t_w(j) \right). \tag{7.12}$$

Equation (7.12) is only based on two assumptions. Firstly, the maximal number of analysis steps m is assumed to be known which is not true for any realistic problem. The second assumption in calculating the expected total computing time is that the sequence of measured computing times repeats after each restart. Section 7.2.3 will illustrate that both these assumptions are reasonable and do not affect the restarting point significantly.

7.2.3. Results

This section presents the results of the time-based restarting Woodbury's identity solution method. To illustrate the restarting procedure, consider the reinforced concrete slab example from Section 7.1.1 again. To obtain more realistic computing times, a mesh refinement is applied to the model. The new model properties are given in Table 7.5.

Table 7.5: Mesh-refined reinforced concrete slab properties.

Problem	Size	DOF per element	Description
Mesh-refined Slab	335331	24	Reinforced concrete slab

Using this problem, it will be motivated that the assumption of the maximal number of analysis steps has no significant effect on the restarting point. To this extend, the elapsed times of the first 80 analysis steps of the problem without restarting are shown in Figure 7.15.

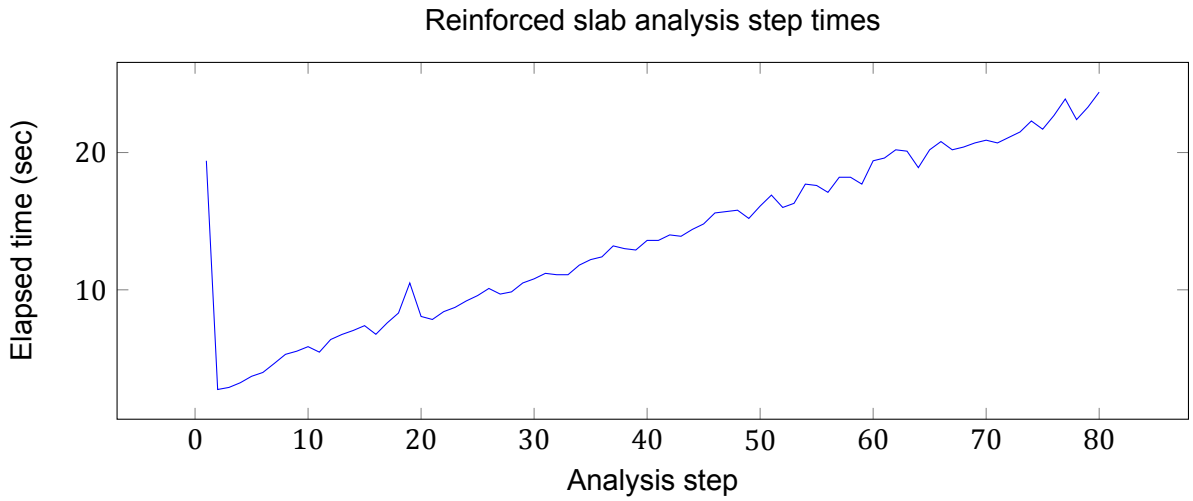


Figure 7.15: Mesh-refined reinforced slab analysis step times.

Using the measured times from Figure 7.15 and using Equation (7.12), the total computing times can be calculated for several different assumptions of the total number of analysis steps. Figure 7.16 shows the total expected analysis times as a function of the restarting point. The optimal restart point is then defined as the minimum of this function.

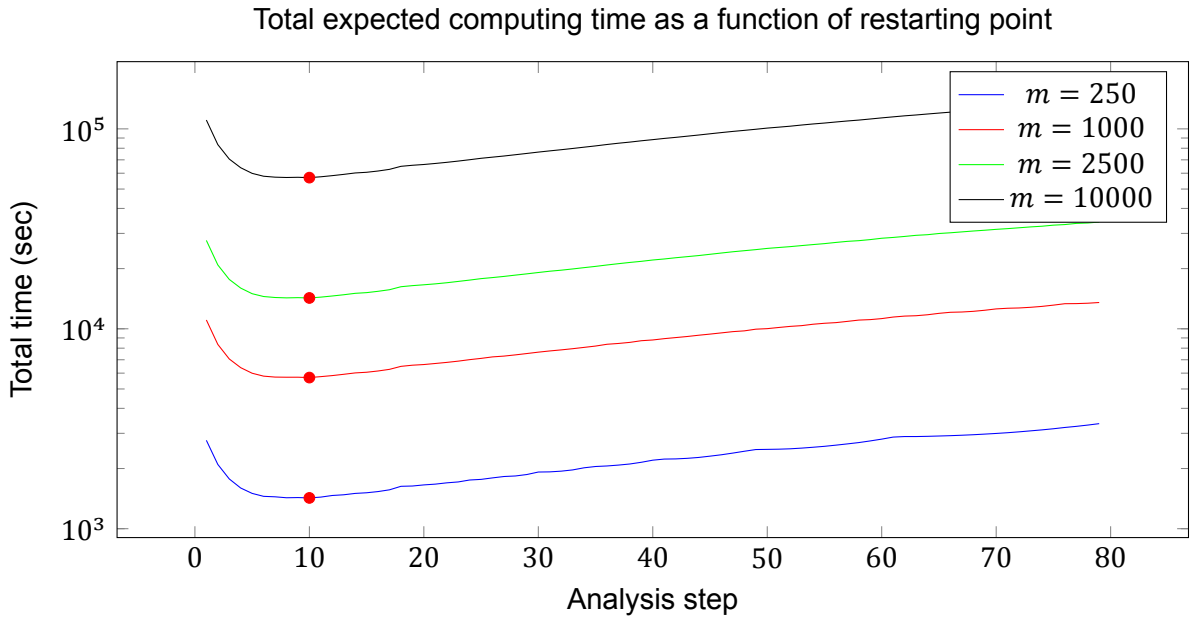


Figure 7.16: Mesh-refined reinforced slab total analysis times as a function of the restarting point.

In Figure 7.16, the optimal restart point is indicated with a red dot. Comparing the optimal restarting points for different values of m, it is clear that this choice has no significant effect on the calculation of the optimal restarting point. Therefore, it is chosen to take $m = 10000$ for all further analyses.

The other assumption in the derivation of the optimal starting point was that the sequence of

measured analysis steps repeats itself. To motivate that this is a reasonable assumption, the reinforced slab analysis is performed with restarting, as can be seen in Figure 7.17.

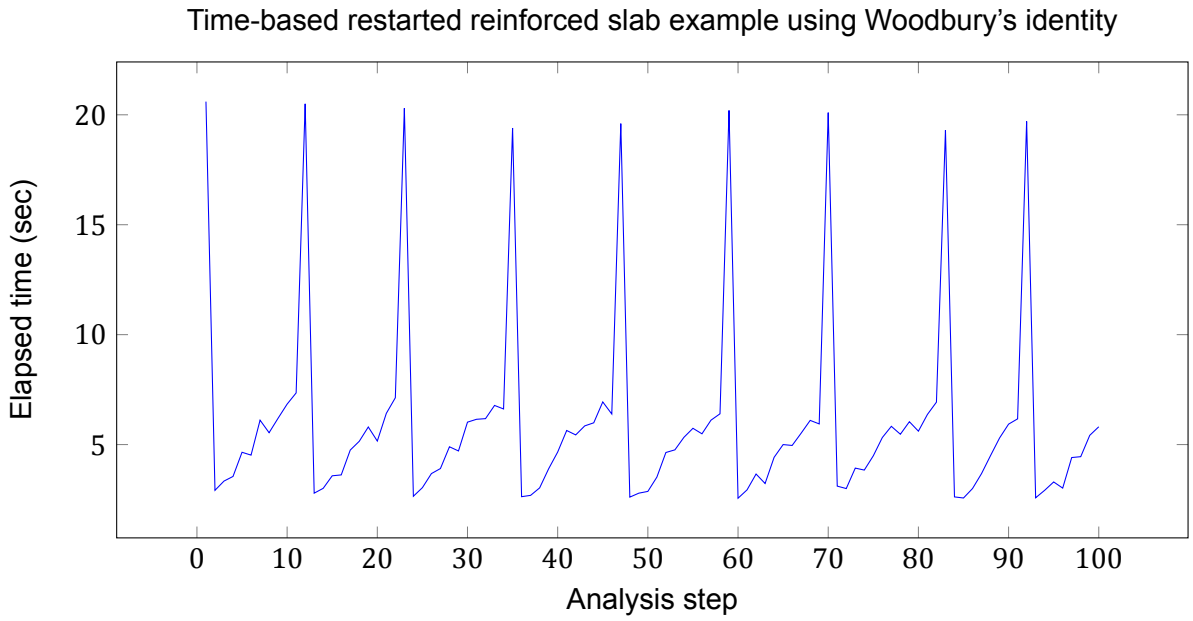


Figure 7.17: Mesh-refined reinforced slab analysis step times.

Indeed, the pattern of increasing analysis times repeats itself after restarting. With this, both assumptions in the time-based restarting strategy are motivated to be reasonable. In order to validate that the presented time-based restarting strategy indeed leads to an optimal restarting point, the problem is solved using the time-based restarting as well as two strategies which restart approximately 20 analysis steps earlier and later respectively. The elapsed times of the first 300 analysis steps of these examples are shown in Figure 7.18.

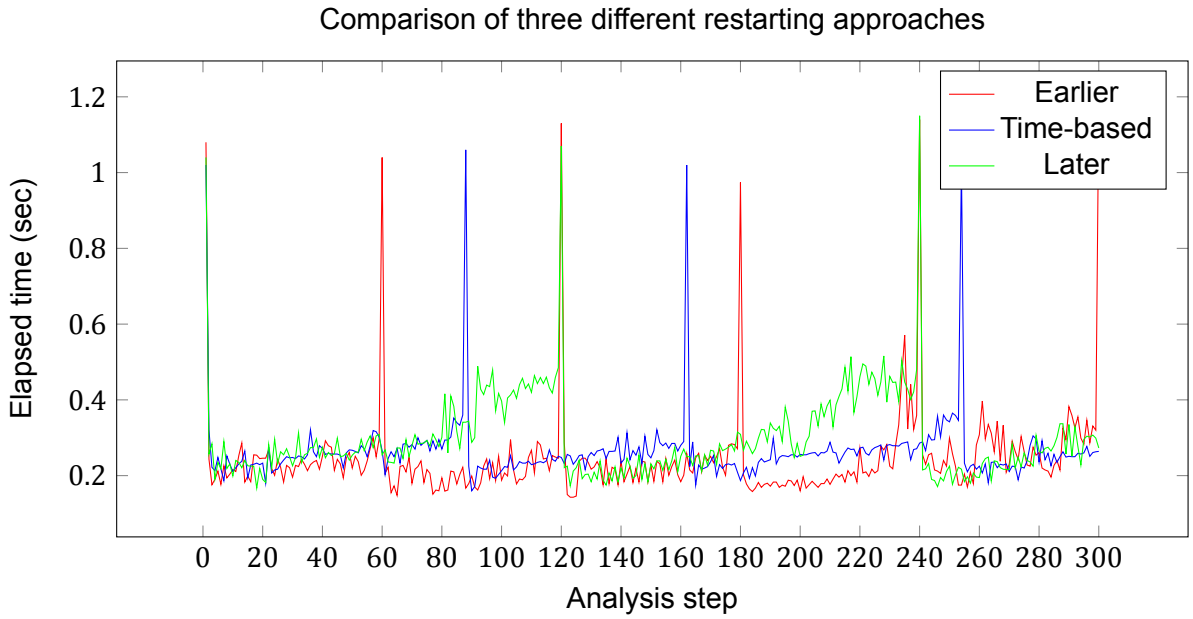


Figure 7.18: Validation of time-based restarting approach comparison to restarting earlier and later respectively.

From Figure 7.18 it can be seen that both alternative (earlier, later) restarting strategies recalculate the matrix factorisation at different times than the time-based restarting. To compare these strategies, the total analysis times corresponding to the three different restarting approaches of Figure 7.18 are given in Table 7.6.

Table 7.6: Total analysis times for different restarting strategies.

Restart point	**Earlier**	**Automatic**	**Later**
Total analysis time	20:47 (+6.9%)	19:26	21:29 (+10.5%)

From Table 7.6 it can be seen that restarting either earlier or later than the time-based restarting results in longer analysis times.

Since the analysis times increase gradually after restarting, continuing longer without restarting results in a performance penalty especially in those last analysis steps. On the other hand, restarting earlier results in too many expensive matrix factorisations. The time-based restarting is able to find an optimal point between these two situations.

In Section 7.1.1 it was mentioned that restarting has a positive effect on the accuracy of the solution. To motive this, the *Intfnp* from the mentioned section was solved using Woodbury's identity with time-based restarting as shown in Figure 7.19.

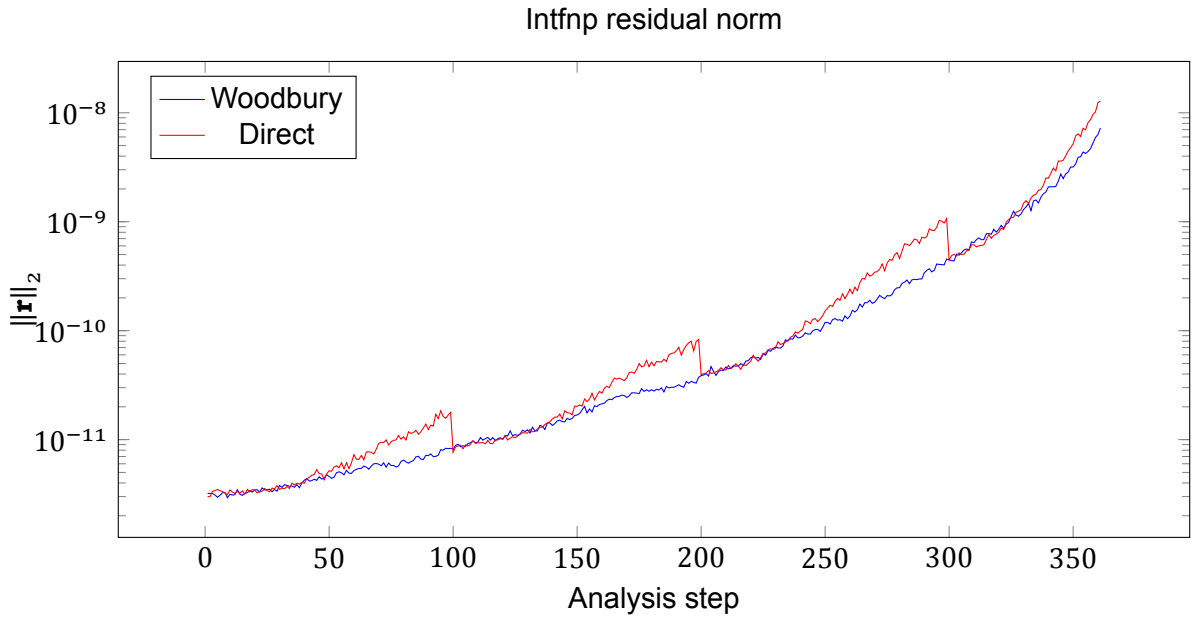


Figure 7.19: Residual norms of *Intfnp* with restarting.

As observed before, the residual norm of Woodbury's identity increases faster than the default direct solution method. However, once Woodbury's identity is restarted the residual norm resets to the value of the direct solution method. The reason for this is that when Woodbury's identity is restarted, a new factorisation is calculated and the rank set back to 0. As a result, all the error build up due to the numerous matrix and vector manipulations are removed and the error is determined by the conditioning of the stiffness matrix.

This thesis considers two alternative solution methods to the direct solution method, a direct approach using Woodbury's identity and *PCG*. In this section, an optimal restarting approach was derived for Woodbury's identity. However, *PCG* requires a similar strategy. Therefore, in Section 7.2.4 some remarks will be made about restarting *PCG*. Furthermore, a remark is made on the implementation of Woodbury's identity in Section 7.2.5. Section 8 will then compare and present results of both solution methods and compare these to the default direct solution method.

7.2.4. Restarting preconditioned conjugate gradients

The two solution methods discussed in this thesis, Woodbury's identity and preconditioned *CG*, are similar in the sense that both require an expensive factorisation step followed by significantly cheaper steps. To illustrate this similarity, Figure 7.20 shows how the elapsed time of the analysis steps increase after restarting for preconditioned *CG*.

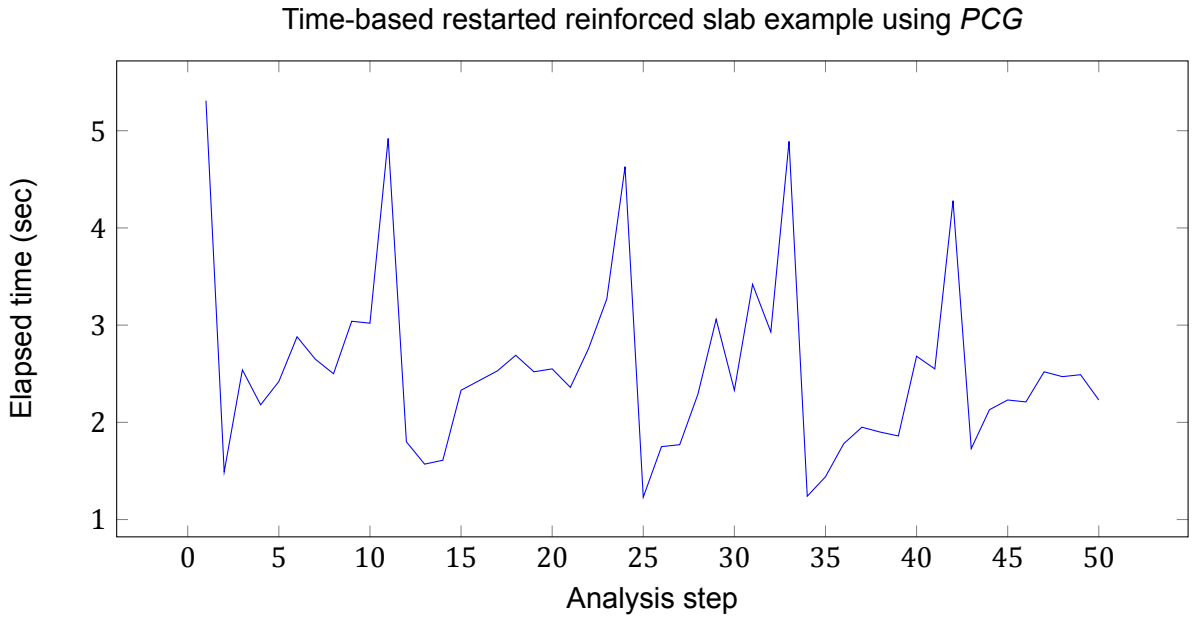


Figure 7.20: Reinforced slab analysis step times for *PCG*.

Indeed, the preconditioned *CG* analysis times show similar behaviour to that of Woodbury's identity from Figure 7.17, after calculating the factorisation the subsequent analysis steps are significantly cheaper and costs increase as more analysis steps are performed. Due to this similarity, *PCG* will be restarted using the same time-based strategy as Woodbury's identity.

7.2.5. Implementation details

The idea of Woodbury's identity is to reuse the factorisation of the stiffness matrix for decreasing the required computations to obtain the solution to Equation (5.1). To this extend, the factorisation is used repeatedly in Equations (6.9) and (6.10). However, it is important to note that the right-hand side vector of Equation (6.9) is constant throughout the analysis. Furthermore, the right-hand side of Equation (6.10) is appended every analysis step with the eigenvectors

corresponding to the considered eigenvalues. Since these right-hand sides are (mostly) constant between analysis steps, it is not necessary to perform back- and forward substitutions on these vectors every analysis step. Instead, only back- and forward substitutions should be performed on the new right-hand side columns of Equation (6.10). Therefore, the solutions to Equations (6.9) and (6.10) are stored in memory in order to prevent these unnecessary substitutions. To illustrate the effect this has on the performance, the reinforced slab problem was solved both with and without storing these intermediate solution steps. Figure 7.21 shows the analysis times for both these implementations.

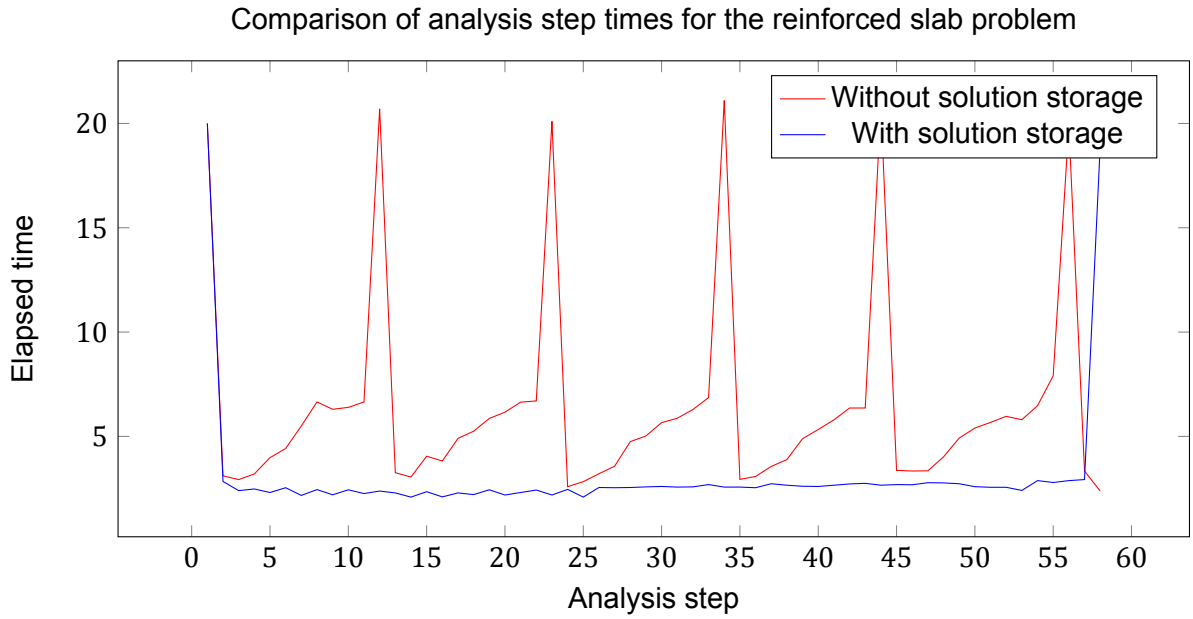


Figure 7.21: Illustration of how efficient storage of intermediate steps in Woodbury's identity can improve computational performance.

From Figure 7.21 it is obvious that not storing the solutions to Equations (6.9) and (6.10) and repeatedly performing back- and forward substitutions has a significant impact on performance. By storing the solutions and efficiently reusing these in the subsequent analysis steps, there is almost no increase in the analysis time. Although this implementation results in improved analysis times, the drawback is that the memory requirements increase. To motivate that this is not a problem note that the implementation with the solution storage is restarted at around analysis step 57. At this point, the rank was 168 resulting in $(168+1)\cdot 335331$ floating points to be stored in memory. Assuming a floating point requires 8 bytes (double precision), this equals approximately 453MB of storage. Obviously, the memory requirements increase with the problem size and the restarting point of Woodbury's identity. However, considering that the given problem size is of the same order as realistic problems, this memory requirement is acceptable especially considering the memory capacities of modern computers.

8

Results

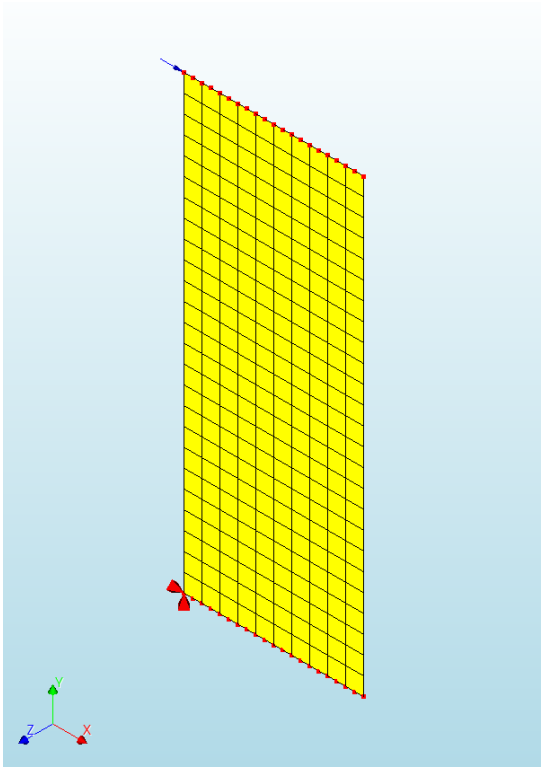
In this section, the performance of Woodbury's identity and preconditioned *CG* will be analysed and compared to the default direct solution method. To this extend, two performance benchmarks are defined. The two problems that will be used are the reinforced concrete slab problem from Section 7.2.3 and a mesh-refined version of the shear wall from Section 7.1.1. The two benchmark problems that will be used are summarized in Table 8.1.

Table 8.1: Performance benchmarks for solution methods.

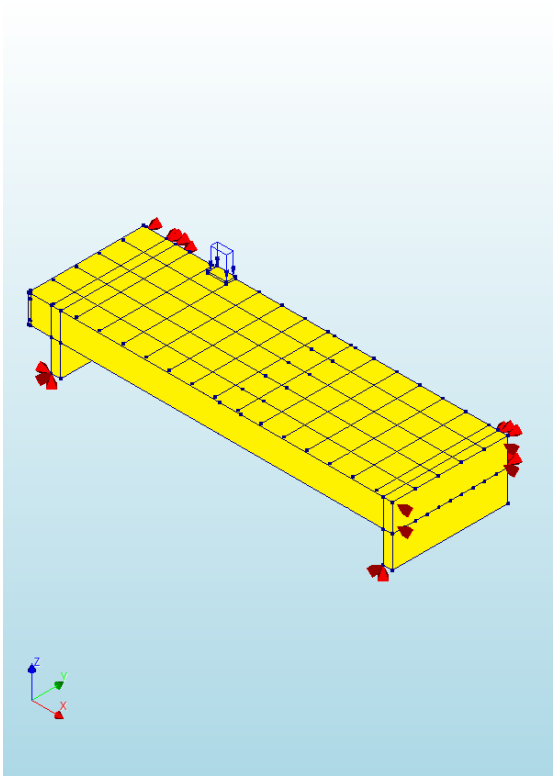
Problem	Size	Description
Shear wall	338400	2-dimensional shear wall
Slab	335331	3-dimensional reinforced concrete slab

The benchmark problems from Table 8.1 are shown in Figure 8.1. Note that some of the reinforcements in the concrete slab are visible. Furthermore, supports are shown with red arrows, loads with blue arrows and tyings[1] as dotted red lines.

[1] Forcing equal displacements for a group of elements.



(a) Shear wall model



(b) Reinforced concrete slab model

Figure 8.1: Visualisation of benchmark problems.

The performance of the solution methods will be tested on these benchmark problems. To investigate the influence of parallel computing, both problems will be solved while running single- and multi-threaded using 4 threads.

As mentioned in the introduction, the solver becomes the dominant factor in *SLA* as the problem size increases. To illustrate this further, first a remark is made on the structure of *SLA*. Sequentially linear analysis has a modular design where each *building block* has a specific task such as setting up element stiffness matrices or solving the linear system of equations. The three building blocks that are most influential on the performance of *SLA* are the following

- *SOLVE*: solving the linear system of equations for the unknown displacements.
- *STREAC*: calculating stresses and strains from the displacements.
- *SLSCAL*: finding the critical scaling factor and scaling the linear results.

The contributions of these three building blocks to the total analysis time in *SLA* are shown for several problem sizes in Figure 8.2.

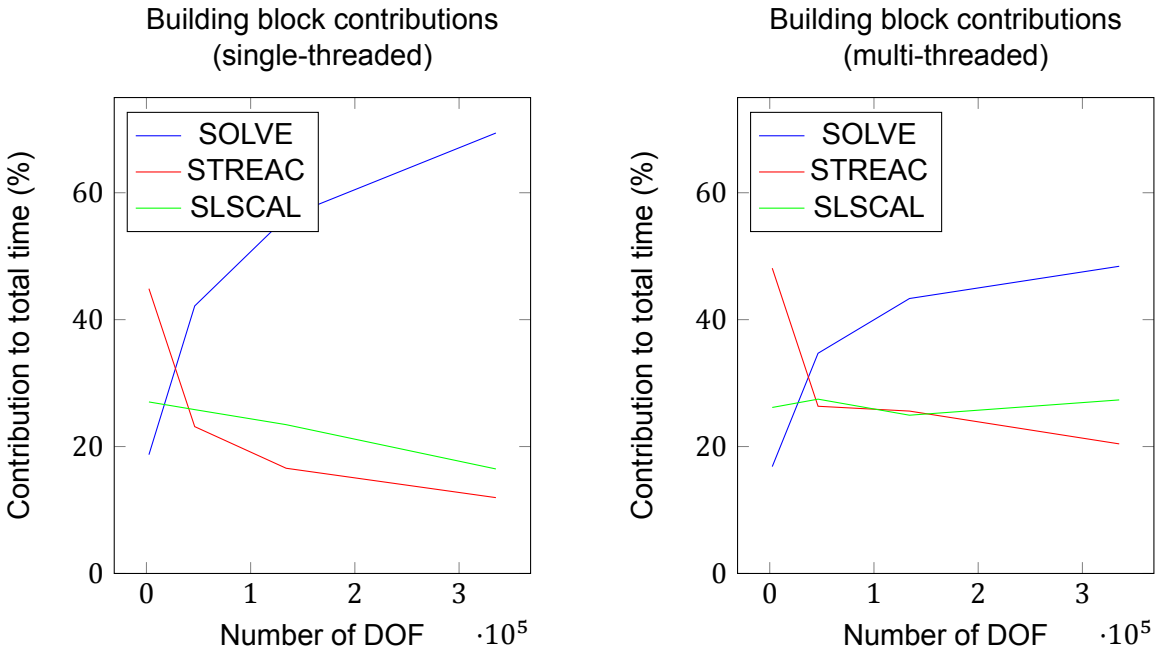


Figure 8.2: Contribution of building blocks to the total analysis time as a function of the problem size.

It can be seen in Figure 8.2 that the solver becomes increasingly dominant in the total analysis time, especially for problems run single-threaded. When running multi-threaded, the dominance of the solver on the total analysis time is reduced. This can be attributed to the fact that neither *STREAC* or *SLSCAL* runs in parallel. As a result, the work of these building blocks can not be distributed over multiple threads resulting in a larger contribution to the total analysis time of these building blocks. Reducing the influence of the solver on the total analysis time, it is likely that these two building blocks become the main bottleneck for the performance of *SLA*.

Due to the discussed similarity between Woodbury's identity and *PCG*, Section 8.1 will first compare both methods. Both methods are also compared to the default *CG* preconditioned with the *ILU* factorisation which will illustrate the ineffectiveness of the implementation of this preconditioner. Section 8.2 then analyses the performance of both *PCG* and Woodbury's identity and puts these results in perspective with respect to the default direct solution method.

8.1. Comparison PCG and Woodbury's identity

It was argued that Woodbury's identity and *PCG* using the complete factorisation are similar in the sense that both require an expensive factorisation, followed by significantly cheaper analysis steps. This section will compare both methods along with *CG* preconditioned with the *ILU* factorisation. The *ILU* factorisation is the default preconditioner for *CG* in *DIANA*. To illustrate how the number of *CG* iterations compares to the rank of Woodbury's identity, both have been given in Figure 8.3.

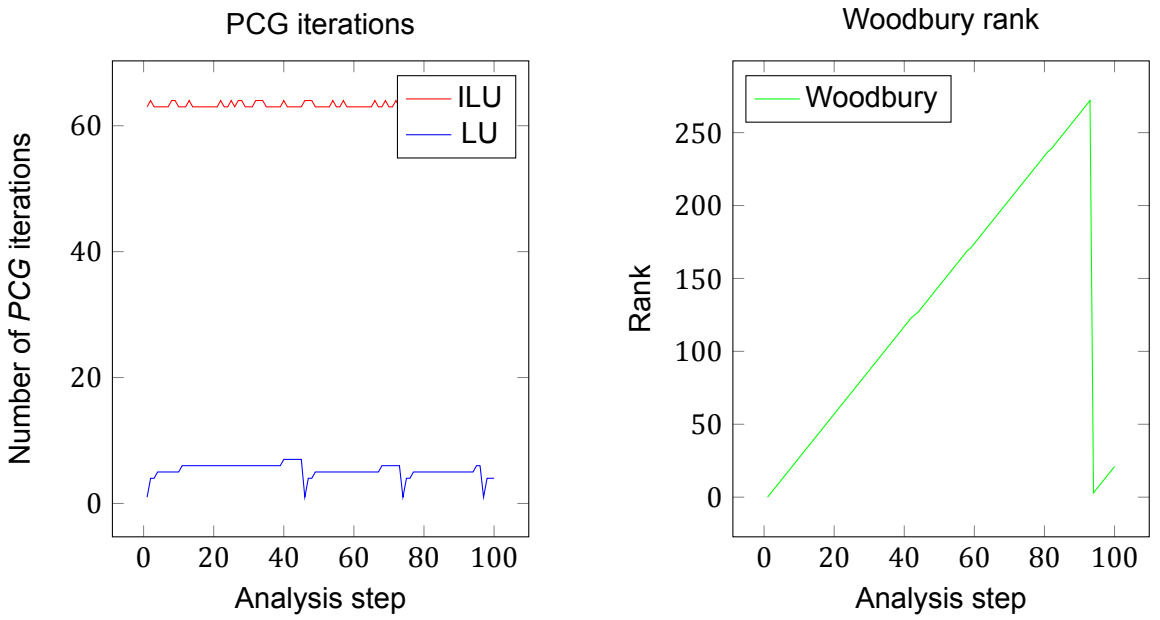


Figure 8.3: Comparison between Woodbury's identity rank and the number of *PCG* iterations for different preconditioners.

Clearly, the *ILU* preconditioner performs poorly in reducing the number of distinct eigenvalues, as still many iterations are required to obtain convergence, even in the first analysis step directly after factorisation. Furthermore, it can be observed that the number of required iterations using the *ILU* preconditioner does not increase. The reason for this is that in the implementation the *ILU* factorisation is updated each analysis step. As a result, the number of required iterations remains approximately constant.

Comparing the number of iterations for *LU* preconditioned *CG* and Woodbury's identity, a clear difference is observed. Where the rank of Woodbury's identity increases continuously, the number of *PCG* iterations increases only fast during the first few analysis steps after recalculating the *LU* factorisation. The reason for this is that the problem is considerably larger than the ones considered in Section 7. Therefore, the singularities discussed in Section 7.1.2 do not occur as fast. As a result, the conditioning of the system of equations remains relatively constant as can be seen in Figure 8.4.

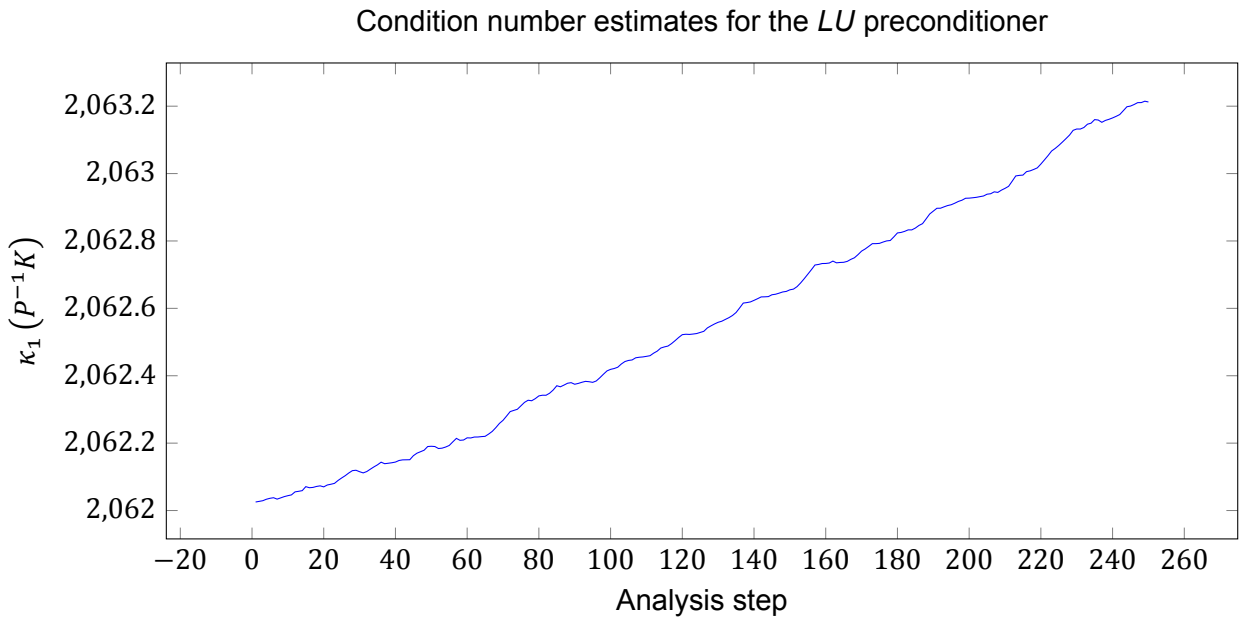


Figure 8.4: Condition number estimates for the *LU* preconditioner.

As can be seen from Figure 8.4 the conditioning of the system barely increases such that the required number of *PCG* does not increase.

In the implementation of Woodbury's identity, the rank increases according to the dominant eigenvalues as described in Section 7.1. This increase is independent on if an element is already damaged and as a result the rank increases steadily.

To further illustrate the inefficiency of the *ILU* preconditioner the analysis times of the three mentioned solution methods are given in Figure 8.5.

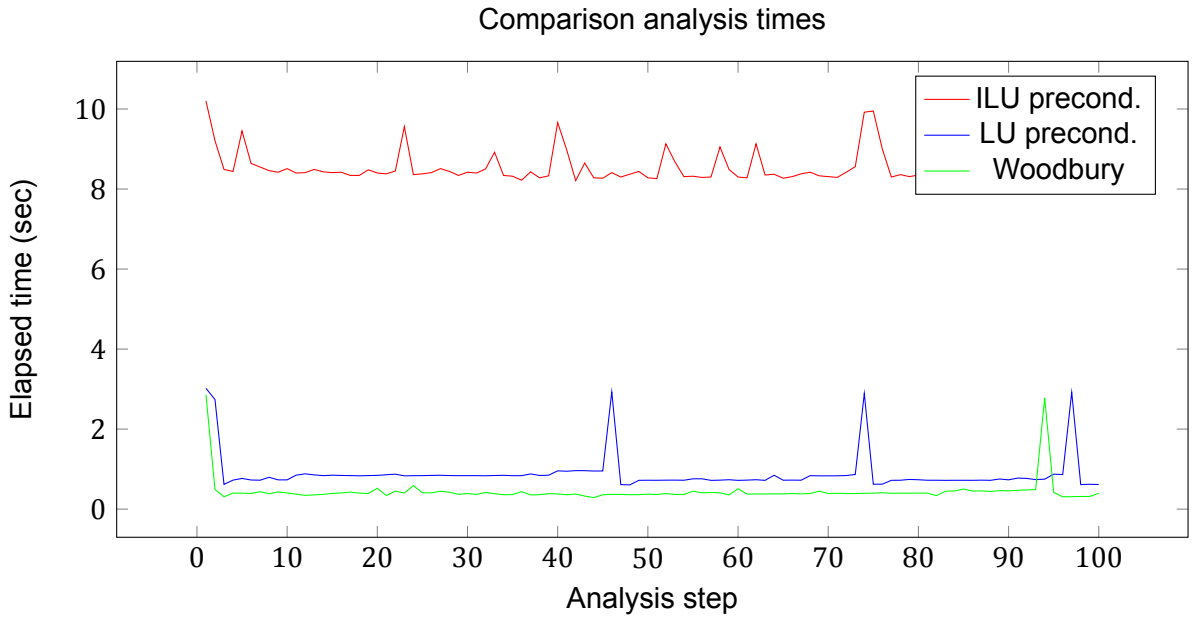


Figure 8.5: Comparison between analysis times of CG using different preconditioners.

Note that the analysis time of the *ILU* preconditioner is significantly higher than that of the *LU* preconditioner. Even in the first analysis step, where both the *ILU* and *LU* factorisation are calculated, the *LU* preconditioner performs better. The reason for this is twofold. Firstly, calculation of the *LU* factorisation is highly optimised as a result of *PARDISO*, whereas the calculation of the *ILU* factorisation uses a less optimised algorithm. Secondly, the required number of *CG* iterations after applying the *LU* factorisation is small while many iterations are required with the *ILU* preconditioner, even directly after factorisation as was observed in Figure 8.3. The combination of these two factors leads to higher analysis times.

8.2. Performance analysis

In this section the performance of Woodbury's identity and *PCG* will be analysed and compared to the default direct solution method. To indicate how the direct solution performs on the given benchmark problems, Table 8.2 shows the contributions of the three most dominant building blocks on the total analysis time when using the default direct solution method.

Table 8.2: Contribution of the solver to the total analysis time for the direct solution method.

	Shear wall		**Slab**	
	Single-thread	Multi-thread (4)	Single-thread	Multi-thread (4)
SOLVE	47.9%	41.4%	70.6%	48.8%
STREAC	10.0%	10.8%	11.8%	20.1%
SLSCAL	40.3%	45.6%	15.6%	27.8%

From Table 8.2 it is clear that the solver contributes significantly to the total analysis time, especially when using single-threading. However, when using multi-threading the contribution of STREAC and SLSCAL increases. Therefore, decreasing the contribution of the solver to the total analysis time will result in these building blocks to become increasingly dominant.

The results of the benchmark problems mentioned in Table 8.1 are shown in Figures 8.6 - 8.9.

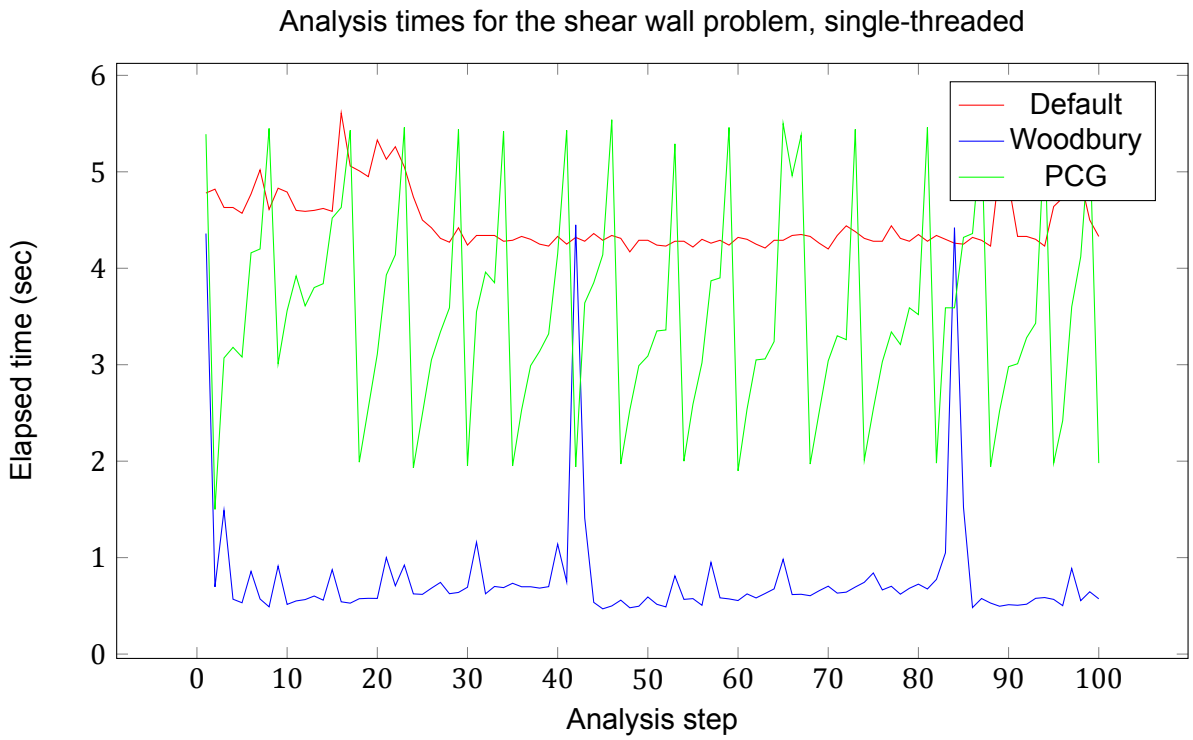


Figure 8.6: Benchmark shear wall problem, single-threaded

Figure 8.6 shows the results of Woodbury's identity and *PCG* on the shear wall problem run on a single-thread. Note that every time Woodbury's identity is restarted, it is equivalent to the direct solution method. This is also reflected in the figure since the cost of restarting is exactly that of the direct solution method. The fluctuations in the analysis times of the direct solution method can be attributed to occupation of the machine the problem was solved on. Numerous people use these machines, as a result of which calculations slow down when occupation is high and resources have to be shared amongst users. After restarting Woodbury's identity, the costs are approximately reduced by a factor 6. On the other hand, the costs of *PCG* are only a factor 2 less than the direct solution method. Furthermore, restarting *PCG* is more expensive and the analysis times increase significantly faster. The reason for this disappointing performance of *PCG* is the fact that the considered problem is 2-dimensional. As a result, the bandwidth of the stiffness matrix is relatively small. The costs of performing back- and forward substitutions are relatively more expensive compared to the cost of a matrix factorisation for small bandwidths. Since *PCG* uses these back- and forward substitutions every analysis step to apply the preconditioner, performance degrades.

In Figure 8.7 the same problem is solved as in Figure 8.6 only now using multi-threading.

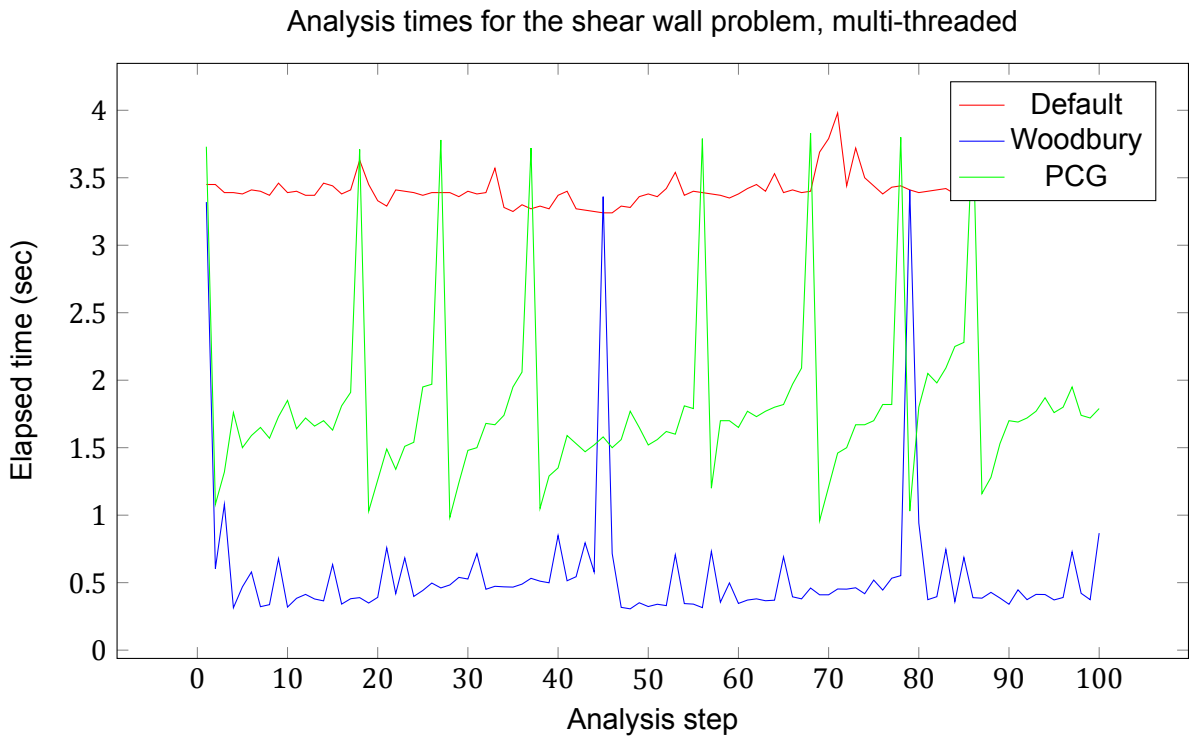


Figure 8.7: Benchmark shear wall problem, multi-threaded

The behaviour of Woodbury's identity is similar as in Figure 8.6; the cost of the matrix factorisation is the same as the direct solution method and subsequent analysis steps are significantly cheaper. However, the performance of *PCG* is now significantly better with respect to the default direct solution method. The reason for this is the use of multi-threading. In Figure 8.6 the performance of *PCG* was disappointing due to the relatively expensive back- and forward substitutions. However, due to the use multi-threading the cost of the repeated back- and forward substitutions have decreased. As a result, the overall performance of *PCG* is better compared to the direct solution method.

Figure 8.8 shows the results of the reinforced slab problem solved on single-threaded.

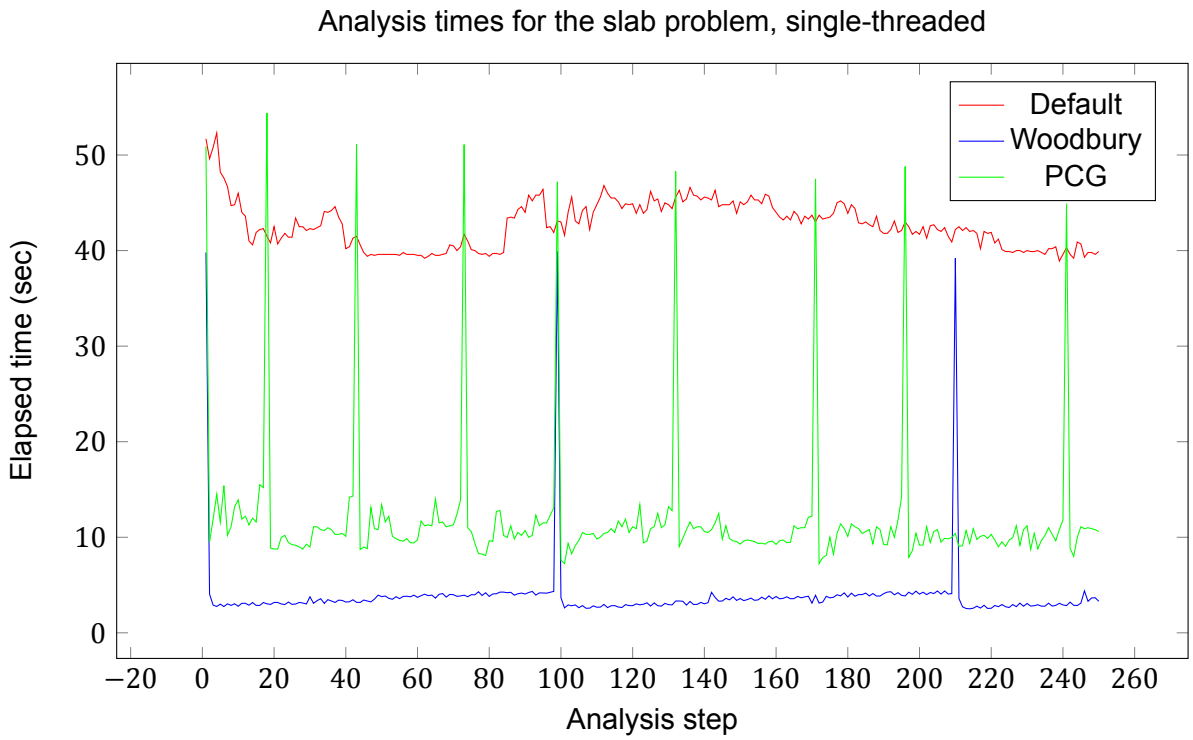


Figure 8.8: Benchmark reinforced concrete slab problem, single-threaded

The performance of Woodbury's identity is again similar as seen in Figures 8.6 and 8.7 in the sense that when Woodbury's identity is restarted, the cost is equal to that of the direct solution method. Furthermore, the performance of *PCG* is different than for the 2-dimensional shear wall problem. After restarting, the costs do not increase as sharply as for the 2-dimensional problem. This can be attributed to the larger bandwidth of the stiffness matrix. Since the reinforced slab is a 3-dimensional problem, the bandwidth is significantly larger than for a 2-dimensional problem. As a result, the back- and forward substitutions are relatively cheap compared to the matrix factorisation.

In Figure 8.9 the same problem is solved as in Figure 8.8 only now using multi-threading.

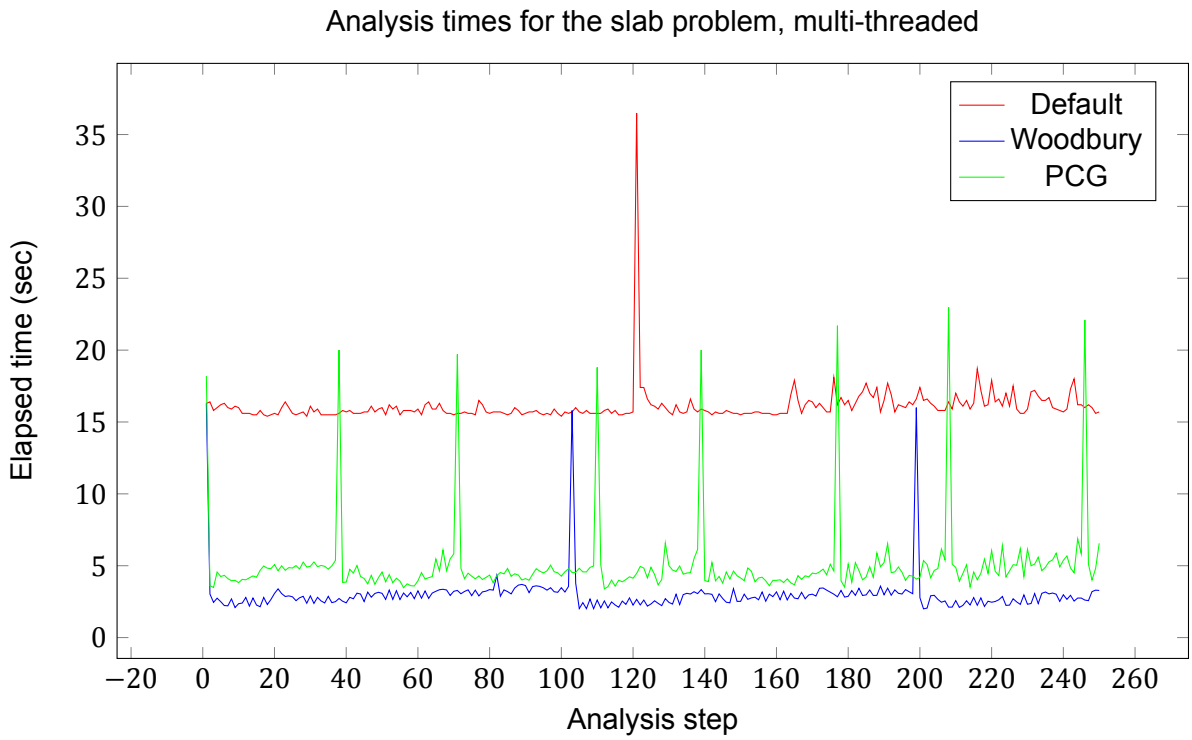


Figure 8.9: Benchmark reinforced concrete slab, multi-threaded

Note first the large peak in the cost of the direct solution method. This is typically the result of the occupancy of the machine the problem was solved on. Again, the performance of Woodbury's identity is similar as in Figures 8.6, 8.7 and 8.8. However, the performance of *PCG* is now significantly better than in Figure 8.8. Due to the multi-threading, the factorisation and repeated back- and forward substitutions are cheaper such that the performance of *PCG* is close to that of Woodbury's identity.

Before summarising the results of the benchmark problems, a remark is made on the restarting of Woodbury's identity. In Figures 8.8 and 8.9 it can be seen that the costs have not significantly increased before the method is restarted. To justify these restarts, the shear wall problem is solved using both an earlier and later restarting point. The first 125 analysis steps are shown in Figure 8.10.

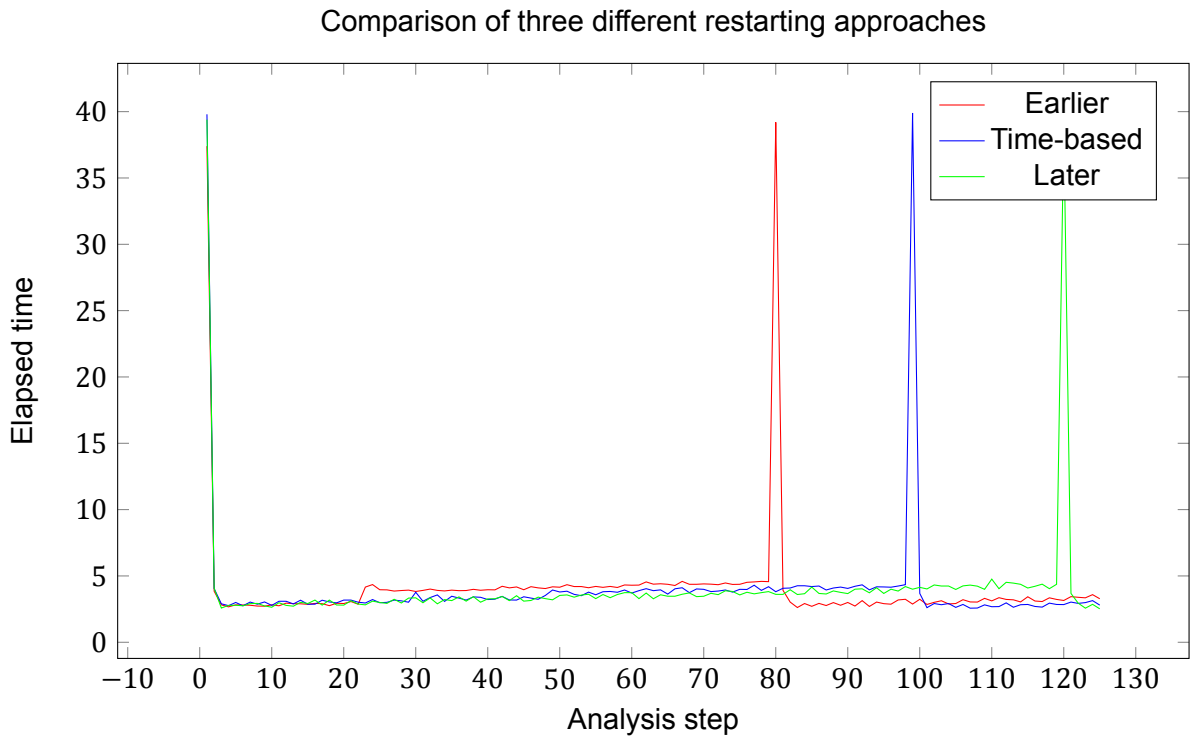


Figure 8.10: Different restarting points single threaded slab.

The corresponding analysis times are given in Table 8.3.

Table 8.3: Total analysis times for different restarting strategies.

Restart point	**Earlier**	**Time-based**	**Later**
Total analysis time	1:26:40	1:24:32	1:26:13

Comparing the total analysis times of the earlier and later restarting to the automated time-based restarting, it can be concluded that the proposed time-based restarting results in the best performance.

To summarise the results of the solution methods on the benchmark problesm, the total analysis times corresponding to Figures 8.6 - 8.9 are shown in Table 8.4 along with a percentual difference with respect to the direct solution method.

Table 8.4: Overview of improvements to analysis times on benchmark problems.

	Shear wall		**Slab**	
	Single-thread	Multi-thread (4)	Single-thread	Multi-thread (4)
Base	2:44:16	2:24:48	4:13:11	2:19:07
Woodbury	1:24:35 -48.5%	1:30:42 -37.4%	1:24:32 -66.6%	1:22:42 -40.5%
PCG	2:30:38 -8.3%	2:14:38 -7.0%	2:55:11 -30.8%	1:45:41 -24.0%

Indeed, both Woodbury's identity and *PCG* perform significantly better than the default direct solution method. It can be seen that the performance increase is smaller when the problem is run on multiple threads. The reason for this is that *PARDISO* is highly optimized to run parallel, as can also be seen by comparing the single- and multi-threaded times of the direct solution method. Furthermore, *PCG* performs considerably worse than Woodbury's identity on the shear wall problem. This can be attributed to the fact that the shear wall is a 2-dimensional problem. As a result, the back- and forward-substitutions are relatively expensive compared to the factorisation and since these substitutions have to be repeatedly performed as preconditioner step, the performance degrades.

At the start of this section, the dominance of the solver in *SLA* was motivated in Table 8.2. This table shows the contributions of the dominant building blocks on the total analysis time of the direct solution method. To analyse the influence of the building blocks on the performance of the new solution methods, Table 8.5 shows the old contributions along with the building block contributions using the two new solution methods.

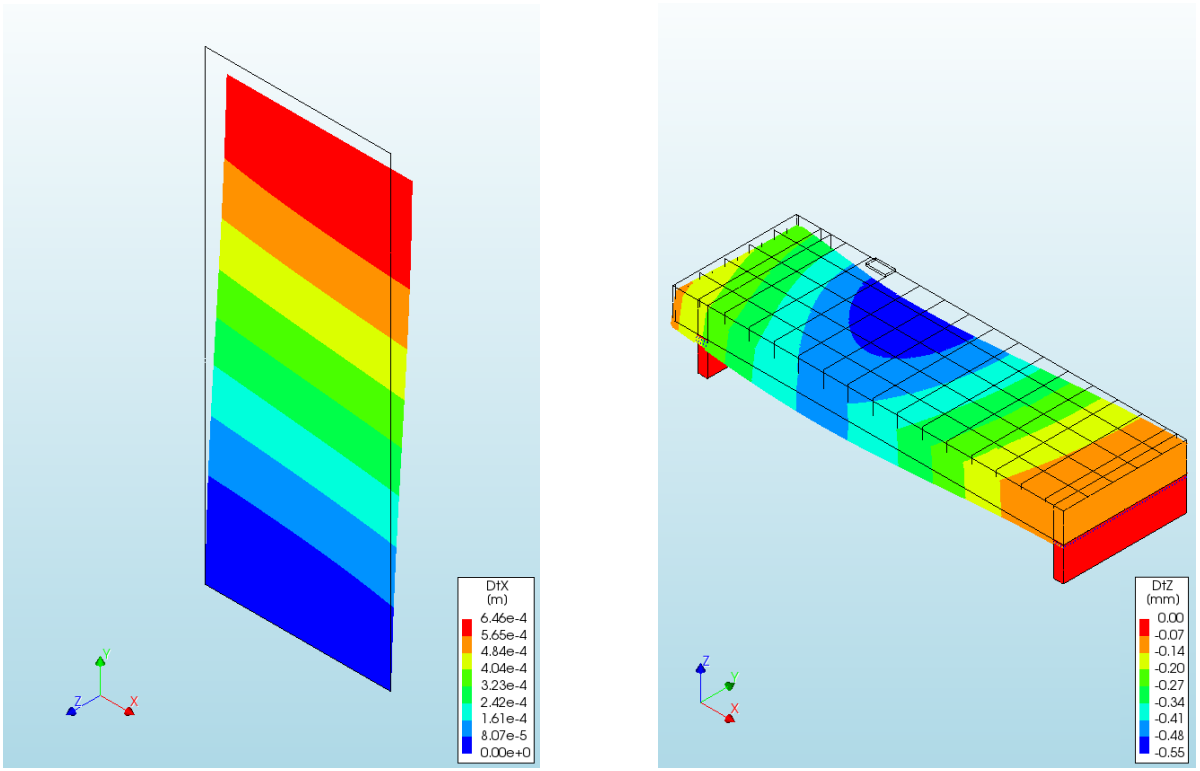
Table 8.5: Contribution of the dominant building blocks to the total analysis time.

		Shear wall		**Slab**	
		Single-thread	Multi-thread (4)	Single-thread	Multi-thread (4)
	SOLVE	47.9%	41.4%	70.6%	48.8%
	STREAC	10.0%	10.8%	11.8%	20.1%
	SLSCAL	40.3%	45.6%	15.6%	27.8%
Woodbury	**SOLVE**	16.7%	13.1%	20.0%	16.0%
Woodbury	**STREAC**	19.3%	19.7%	30.9%	32.3%
Woodbury	**SLSCAL**	60.6%	63.9%	43.9%	46.3%
PCG	**SOLVE**	39.1%	26.0%	29.2%	20.7%
PCG	**STREAC**	9.8%	12.8%	34.8%	30.1%
PCG	**SLSCAL**	49.2%	59.1%	30.8%	44.1%

Comparing the building block contributions of both new solution methods to the contributions of the default direct solution method, it can be concluded that the effect of the solver on the performance has been significantly reduced. In particular Woodbury's identity performs good such that the solver is no longer the dominant building block in *SLA*. In particular, the determining the scaling factor for the linear analysis and scaling the linear analysis becomes dominant

as the impact of the solver is reduced. When further improving the computational performance of *SLA*, the biggest improvements can be achieved when improving this building block.

Finally, to motivate that Woodbury's identity produces physically meaningful results the model displacements of both benchmark problems are shown in Figure 8.11.



(a) Shear wall model

(b) Reinforced concrete slab model

Figure 8.11: Visualisation of benchmark problem results.

The displacements shown in Figure 8.11 are in line with expectations given the loads as were shown in Figure 8.1.

The results presented in this section only consider a single problem size. In order to investigate how the proposed solution methods scale with the problem size, the next section will use the same problems with different mesh sizes in order to assess the scalability of the solution methods.

8.3. Scaling of results

This section will consider four different mesh sizes for each of the benchmark problems of Table 8.1 to analyse the scalability of the proposed solution methods. The different mesh sizes of the two benchmark problems are given in Table 8.6.

Table 8.6: Different mesh sizes of the benchmark problems.

	Degrees of freedom			
Shear wall	24240	96480	150600	338400
Slab	2403	46337	134082	335331

Similarly to Section 8.2, the scalability of the solution methods will be assessed both using single- and multi-threading. Figure 8.12 shows the scaling on the 2-dimensional shear wall problem using single-threading.

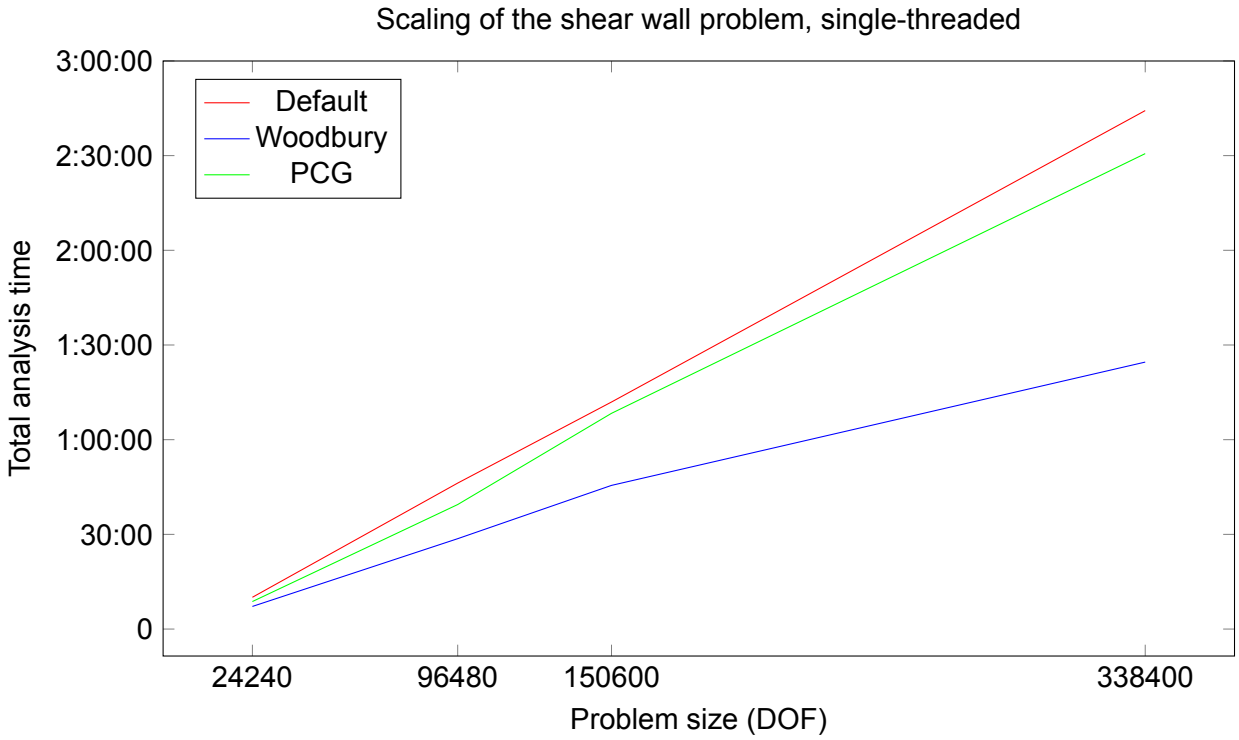


Figure 8.12: Scaling of the solution methods on the shear wall problem, single-threaded.

From Figure 8.12 it is immediately clear that both Woodbury's identity and *PCG* perform equal to the direct solution method for small problem. For these problems, the costs of factorisation and back- and forward substitutions are low. On the other hand, the set-up costs for Woodbury's identity and *PCG* for these problem sizes are relatively expensive. As a result, most of the performance that is gained with the solution methods is lost with this necessary overhead. The set-up costs include allocating memory, creating arrays and initialising the *PARDISO* interface for the back- and forward substitutions. The cost of these operations are largely independent of the problem size. Therefore, when increasing the problem size the relative influence of the set-up on the overall performance will decrease.

For increasing problem sizes, it is observed that the performance of *PCG* remains largely equal to that of the direct solution method. In Section 8.2 it was discussed that this can be attributed to the fact that for 2-dimensional problems the back- and forward substitutions are relatively expensive. Note furthermore the linear scaling of the direct solution method. The calculation

of a matrix factorisation scales cubically with the problem size ($\mathcal{O}\left(n^3\right)$) and the back- and forward substitutions quadratically with the problem size ($\mathcal{O}\left(n^2\right)$). Therefore, a similar scaling of the direct solution method would seem logical. The fact that a linear scaling is observed can be attributed to a combination of two reasons. Firstly, the bandwidth of the stiffness matrix increases relatively slowly with increasing problem sizes for 2-dimensional problems. Therefore, the number of non-zeros (that occur within the bandwidth) increases slowly correspondingly. Secondly, the mentioned cubic and quadratic scaling does not consider the sparsity pattern of the matrix. *PARDISO* is highly-optimised and exploits the sparsity pattern of the stiffness matrix to minimise the required computations. As a result of a combination of the mentioned reasons, the method scales linearly for 2-dimensional problems.

From Figure 8.12 it can also be seen that the scaling of Woodbury's identity is significantly better than that of either *PCG* or the direct solution method. This is promising, especially when there is a desire for solving even larger 2-dimensional problems.

In Figure 8.13 the same problems are solved only now using multi-threading.

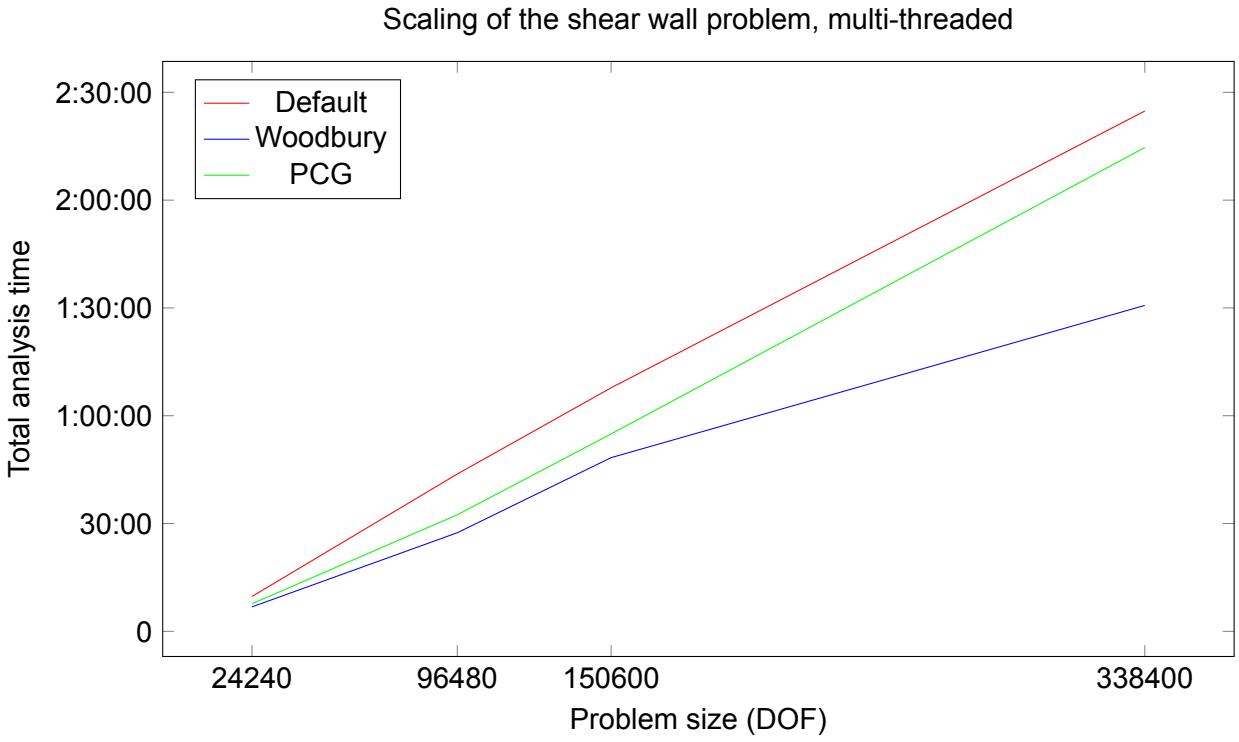


Figure 8.13: Scaling of the solution methods on the shear wall problem, multi-threaded.

Using multi-threading, it can be seen that *PCG* performs slightly better compared to the direct solution method than when using single-threading. This is the result of the relatively expensive back- and forward substitutions being able to performed in parallel, reducing the impact on performance. Furthermore, the linear scaling of the direct solution method is also observed when using multi-threading.

Comparing Figures 8.6 and 8.7, it can be seen that Woodbury's identity scales less favourable when using multi-threading than for single-threading. The reason for this is that the numerous matrix and vector manipulations use less optimised linear algebra libraries, some of which

do not allow for parallel computing. Hence, the implementation of Woodbury's identity scales less favourable on multiple threads. However, even though the scaling is less favourable than on a single thread, the performance is still significantly better than the direct solution method especially for large problems.

The previous two figures showed the scaling of the results on a 2-dimensional problem. Figure 8.8 shows the scaling of the solution methods on the reinforced slab problem, a 3-dimensional problem.

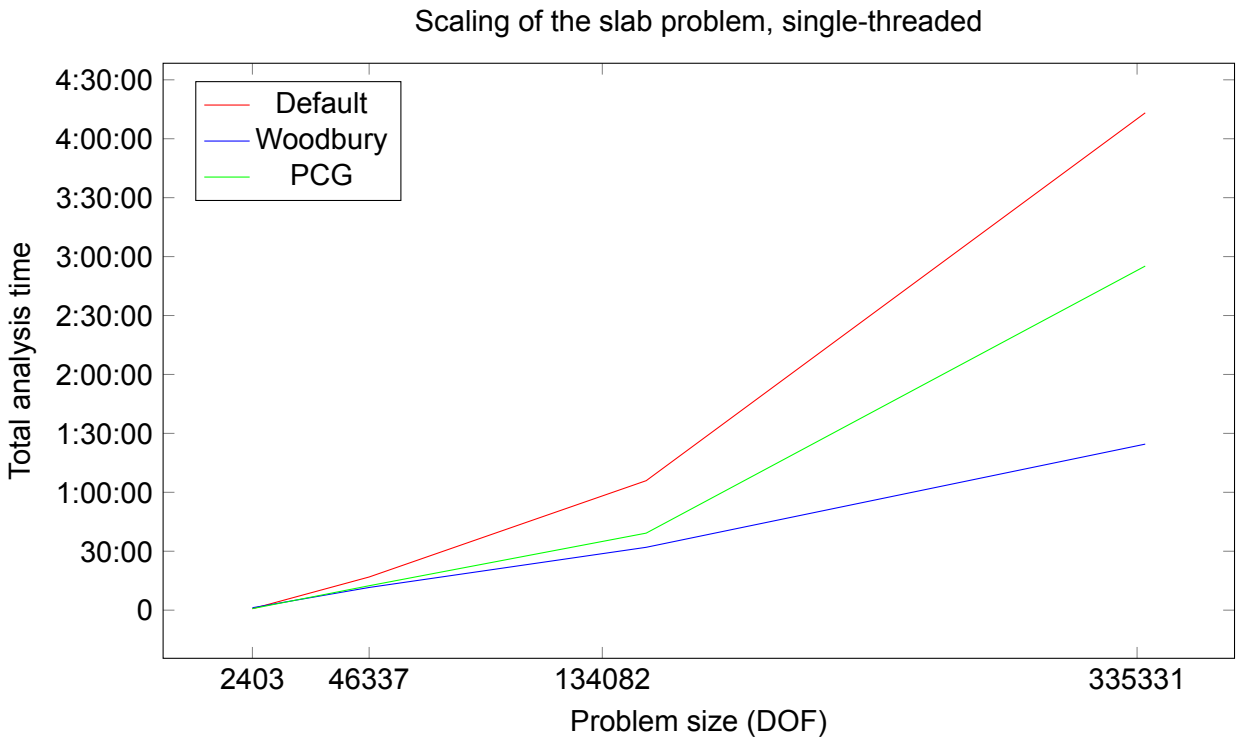


Figure 8.14: Scaling of the solution methods on the reinforced slab problem, single-threaded.

The linear scaling of the direct solution method observed in Figures 8.6 and 8.7 no longer holds for the 3-dimensional problem. The linear scaling in the mentioned figures was attributed to the small growth of the bandwidth due to the problem being 2-dimensional. The reinforced slab problem is 3-dimensional, as a result of which the bandwidth increases significantly faster. The number of non-zeros in the bandwidth increases faster correspondingly, and more calculations have to be performed on non-zero elements.

Similarly to the 2-dimensional problem, Woodbury's identity scales significantly better than the direct solution method and results in a significant performance increase for large problem sizes. Since the typical problems of interest of *SLA* are 3-dimensional this is a promising result.

In Figure 8.9 the scaling of the solution methods is shown on 3-dimensional problems using multi-threading.

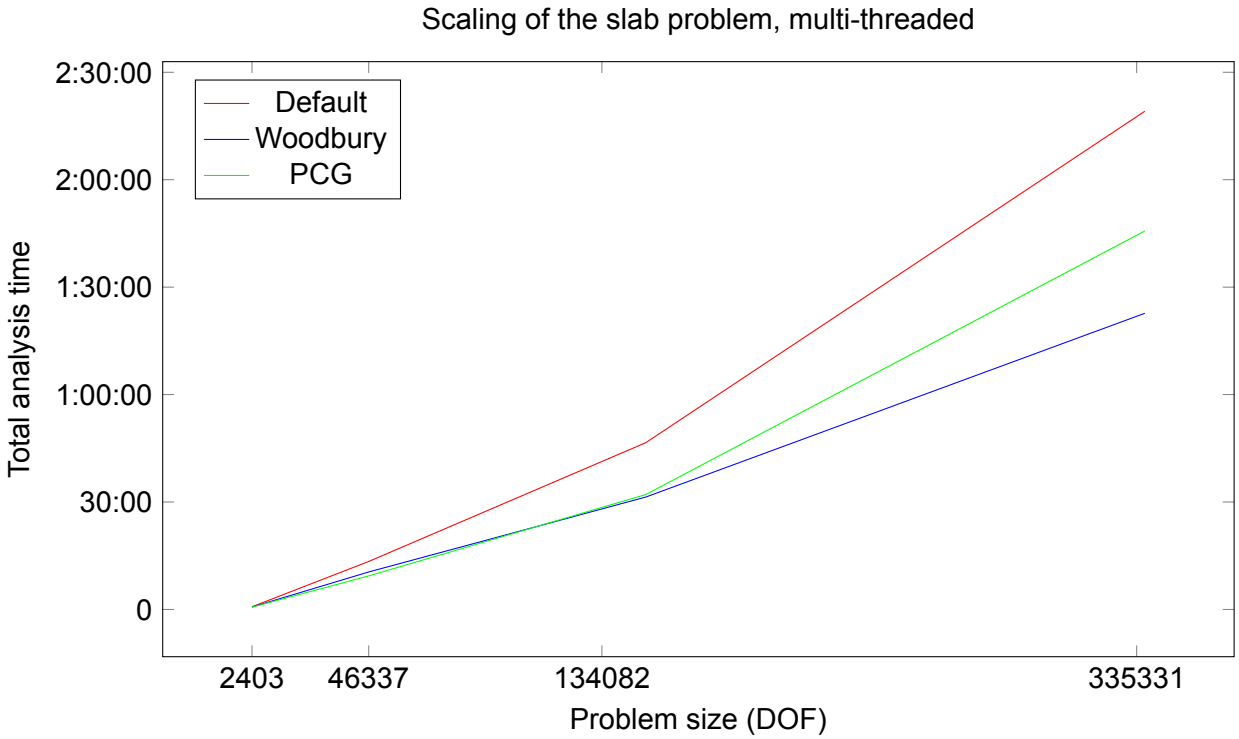


Figure 8.15: Scaling of the solution methods on the reinforced slab problem, multi-threaded.

The behaviour of the solution methods in Figure 8.9 is similar to those in Figure 8.8. The main difference is again the less favourable scaling of Woodbury's identity. Even though Woodbury's identity's performance does not scale as well when using multi-threading, the increase in performance is still significant. For the largest problem considered, the analysis time is almost halved. From Figure 8.15 it can be derived that for even larger problems, the performance gain will be even more significant.

The aim of this thesis is to increase the size of problems that could be solved within an acceptable time frame using *SLA*. To this extend, consider 24 hours as an acceptable total analysis time. The typical problems to be solved with *SLA* are 3-dimensional. Furthermore, any realistic problem problem will not be solved single-threaded especially considering the number of cores of modern computers. Therefore, to analyse what new problem sizes can be solved with Woodbury's identity within the given time, the reinforced slab problem will be considered using multi-threading (on 4 threads). From Section 8.2 it follows that using the direct solution method, it is possible to approximately do 2500 analysis steps in 24 hours. Since cracking constitutes of multiple elements of the model failing, increasing the problem size leads to more analysis steps having to be performed in order to obtain the same structural response. Taking the above in considertion, it follows from numerical experiments that the problem size for Woodbury's identity can be increased by 15.2% such that the analysis still takes approximately 24 hours. The results are summarised in Table 8.7.

Table 8.7: Increase in problem size using Woodbury's identity.

	Default	**Woodbury**
Problem size	335331	386448 (+15.2%)
Analysis steps	2500	2881 (+15.2%)
Analysis time	23:11:10	23:31:55

From Table 8.7 it can be seen that both the problem size and analysis steps have increased with 15.2% while the analysis time has large remained constant. One of the main issues of *SLA* was the poor numerical performance. The implementation of Woodbury's identity has lead to a big step in this regard such that considerably larger problems can be solved in time frames in which prior to this thesis only smaller problems could be solved.

9

Conclusion & discussion

9.1. Conclusion

One of the main issues of *SLA* is the poor numerical performance. Due to the event-by-event strategy, typically many linear analyses have to be solved, each requiring an expensive matrix factorisation. In this thesis, two solution methods have been developed which exploit the event-by-event strategy of *SLA*.

Firstly, a new preconditioned was developed for *CG*. Instead of using an *ILU* preconditioner, the complete *LU* factorisation of the stiffness matrix is used as preconditioner for *CG*. Due to the quality of this preconditioner, few iterations are required every analysis step. From numerical experiments it followed that the performance of this preconditioner was poor on 2-dimensional problems. This was attributed to the fact that the application of the preconditioner on the stiffness matrix for these problems is relatively expensive compared to the matrix factorisation. For 3-dimensional problems, the performance was of the complete factorisation preconditioner was significantly better.

Secondly, a new direct solution was developed. To this extend, Woodbury's matrix identity has been applied which allows for the numerically efficient computation of the inverse of a low-rank corrected matrix. This solution method reuses the old matrix factorisation along with several matrix and vector manipulations to efficiently calculate the solution to a system of linear equations. An optimal restarting strategy was derived to determine the point at which a new factorisation should be calculated. The method performs significantly better than the direct solution method, especially for large problems.

Prior to this thesis, the solver was the main bottleneck of *SLA* in terms of performance. Using Woodbury's identity, the impact of the solver on the analysis time has been significantly reduced up to the point at which the solver is no longer the dominant factor in *SLA*. Furthermore, using Woodbury's identity it is possible to solve significantly larger problems with analysis times that previously only smaller problems were solved in. From numerical experiments it followed that problems up to 15% larger could be solved in a similar time frame as smaller problems used to be solved in. As a result, the applicability of *SLA* on real-life problems has increased.

9.2. Discussion

From numerical experiments it followed that Woodbury's identity does not scale well with parallel computing. Many of the linear algebraic libraries that are used in the implementation of Woodbury's identity do not enable parallel computing. As a result, the performance increase is small when going from single- to multi-threaded. An obvious recommendation is therefore to investigate the use of libraries which allow parallel computing in order to investigate effect on Woodbury's identity.

A related issue is the performance of the other building blocks in *SLA*. Using Woodbury's identity, the contribution of the solver on the analysis time has been greatly reduced. As a result, the building blocks STREAC (calculating stresses and strains) and, especially, SLSCAL (determining scaling factor) have become dominant. These building blocks perform their calculations on integration point level, meaning that they have to loop over all these points. With the current implementation, these calculations are performed sequential, not using any parallel processing. Considering that these building blocks now form the bottleneck in terms of the performance of *SLA*, a large leap forward can be achieved by introducing parallel processing in these building blocks.

When assessing the performance of the implemented solution methods, *proportional loading* was assumed which means that all loads increase and decrease simultaneously at the same rate. Since all loads were proportionate, the critical scaling factor for the linear analysis could easily be found by taking the minimum of a set of values. However, when assuming *non-proportional loading*, the loads are not applied proportionally anymore. As a result, the calculation of the critical scaling factor is no longer possible using the same approach. Instead, a quadratic (for 2-dimensional problems) or cubic (for 3-dimensional problems) equation needs to be solved. Therefore, moving from proportional- to non-proportional loading results in a significant increase in the required computations in the SLSCAL building block. It is therefore worthwhile to investigate methods for decreasing the impact of this building block, especially for non-proportional loading.

Bibliography

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

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

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

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

[5] B. N. Datta. *Numerical Linear Algebra and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 2010.

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

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

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

[9] N.J. Higham. Matrix condition numbers, 1983.

[10] T.B. Jönsthövel. *The Deflated Preconditioned Conjugate Gradient Method Applied to Composite Materials*. PhD thesis, TU Delft, 2012.

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

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

[13] N. Li, Y. Saad, and E. Chow. Crout versions of ilu for general sparse matrices. *SIAM Journal of Scientific Computing*, pages 716–728, 2002.

[14] 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.

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

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

[17] 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.

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

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