Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

Executive Summary

The nested generalized conjugate residual method with shifted Laplace preconditioning for the solution of the finite element discretization of the vector wave equation

Master Thesis for the degree of Master of Science in Applied Mathematics

Problem area
Radar cross section prediction methods are used to analyse the radar signature of military platforms when the radar signature can not be determined experimentally because:

- The platform is in the design, development or procurement phase.
- The platform belongs to a hostile party.

For jet powered fighter aircraft, the radar signature is dominated by the contribution of the jet engine air intake for a large range of forward observation angles. The intake can be regarded as a one-side open large and deep forward facing cavity. Although the contribution of the outer mould shape of the platform can be efficiently and accurately computed using simple scattering models, these can not be used to accurately compute the contribution of the jet engine air intake. Previously, an algorithm was developed to enable so-called full wave analysis of cavity scattering, but the computational work involved prohibits the application for analysis of jet engine air intakes at the relevant excitation frequency band.

Description of work
Full wave cavity scattering is studied by means of a numerical model based on a finite discretization obtained from the Maxwell equations. Non-stationary iterative methods with preconditioning are used to

Report no.
NLR-TR-2007-741

Author(s)
P.B. Hooghiemstra

Classification report
Unclassified

Date
October 2007

Knowledge area(s)
Numerical Mathematics

Descriptor(s)
RCS
Large sparse systems
Iterative solvers
Preconditioning

determine the solution of the system of linear equations. In order to tune the parameters in the chosen method, several numerical experiments are performed.

Results and conclusions
Iterative methods in combination with recently developed technology to solve acoustic problems indicate that it is possible to improve the solution process of X-band scattering of the jet engine air intake of modern fighter aircraft.

Applicability
The method developed during this project can be applied to analyse and improve jet engine air intake geometries of fighter aircrafts in order to decrease the level of observability of the platform.

Nationaal Lucht- en Ruimtevaartlaboratorium, National Aerospace Laboratory NLR

Anthony Fokkerweg 2, 1059 CM Amsterdam,
P.O. Box 90502, 1006 BM Amsterdam, The Netherlands
Telephone +31 20 511 31 13, Fax +31 20 511 32 10, Web site: www.nlr.nl

Nationaal Lucht- en Ruimtevaartlaboratorium

National Aerospace Laboratory NLR

NLR-TR-2007-741

The nested generalized conjugate residual method with shifted Laplace preconditioning for the solution of the finite element discretization of the vector wave equation

Master Thesis for the degree of Master of Science in Applied Mathematics

P.B. Hooghiemstra

Customer	National Aerospace Laboratory NLR
Contract number	—
Owner	National Aerospace Laboratory NLR
Division	Aerospace Vehicles
Distribution	Limited
Classification of title	Unclassified
	October 2007

Approved by:

Author	Reviewer	Managing department
11/10/'07	11/10/07	11/10/2007

Summary

RADAR (**Ra**dio **D**etection **A**nd **R**anging) is technology to detect aircraft and ships by means of electromagnetic waves. These electromagnetic waves impinge on an object and the scattered electric field is received. If the *signal-to-noise* ratio exceeds a minimum hardware specific value, the platform will be detected. During the development phase of a platform, one of the objectives is to minimize the probability of detection of the platform. This can be achieved for example by using special radar absorbing coatings or shape optimization.

A measure of detectability is the *radar cross section* (RCS). From practice it is well known that the contribution of the jet engine air intake of a modern fighter aircraft accounts for the major part of the radar cross section of the total aircraft, if the platform is excited from the front. If a platform is still in the development or procurement phase, the radar cross section can not be measured and in practice it is estimated using prediction techniques. For the contribution of the jet engine air intake this is done by determination of the total electric field on the aperture. The far field scattered field is computed using the electric field on the aperture and from this, the contribution to the radar cross section of the jet engine air intake is computed.

The main part of this thesis concerns the computation of the electric field on the aperture. The *vector wave equation* is derived from the Maxwell equations and it is discretized using the finite element method. A large system of linear equations results and this system is solved with an iterative method. The iterative method of choice is the *Generalized Conjugate Residual* (GCR) method. Since the system matrix is ill-conditioned, a preconditioner is introduced to improve the rate of convergence: A start has been made with the application of the shifted Laplace operator preconditioner (recently developed as a preconditioner for the Helmholtz equation) for the discretized vector wave equation.

Using this preconditioner it is possible to compute the solution of the system for intermediate system size ($N\text{dof} \leq 3.0 \cdot 10^5$.) If the number of degrees of freedom $N\text{dof}$ in the system becomes larger, the memory requirements for storing the Krylov subspace basis vectors are unacceptably high. Therefore, the solution procedure for larger systems uses fast out-of-core storage. If a new basis vector is constructed, the previous computed basis vectors are read in batches into core and the new basis vector is orthogonalized with respect to the basis vectors in the various batches. Since the data transfer between core and the fast hard disk is completed very fast, this method does not increase the total CPU time very much for small problems.

The present method (GCR) is compared to the frontal solver which is a state of the art direct solver. It will be made intuitive that the iterative method using the preconditioner is eventually

faster than the frontal solver, especially when the number of degrees of freedom on the aperture is quite large compared to the total number of degrees of freedom as is the case for an undeep cavity.

Samenvatting

Radar is een algemeen bekend middel om vliegtuigen of schepen te detecteren door middel van elektromagnetische golven. Wanneer deze golven een object raken, wordt het verstrooide veld (deels) opgevangen door de radarontvanger. Als het signaal sterk genoeg is om een bepaalde drempelwaarde te overschrijden, dan wordt het vliegtuig gedetecteerd. Een van de doelen tijdens de ontwikkeling van een platform, is het minimalizeren van de kans op detectie van het platform. Dit kan bijvoorbeeld bereikt worden door een radar absorberende coating te gebruiken of het platform zodanig vorm te geven dat de elektromagnetische golven worden weggekaatst van de bron.

Een maat om de detecteerbaarheid van een platform te specificeren is de zogenaamde *radar cross section* (RCS). Het is algemeen bekend dat de luchtinlaat van de motor van een modern gevechtsvliegtuig een grote bijdrage levert aan de radar cross section van het totale vliegtuig, als het platform van de voorkant wordt aangestraald. Tijdens de ontwikkelingsfase van een platform is het niet mogelijk om de radar cross section te meten en daarom wordt er veel onderzoek gedaan naar methoden om de radar cross section te kunnen voorspellen. Om de bijdrage van de luchtinlaat te berekenen wordt het elektrische veld in de opening van de luchtinlaat bepaald, waarna het verstrooide veld op grote afstand van het platform wordt berekend en hieruit de radar cross section.

Het grootste gedeelte van dit Master project gaat over de bepaling van het elektrische veld op de apertuur van de luchtinlaat. Hiervoor wordt de *vector golf vergelijking* afgeleid van de Maxwell vergelijkingen en gediscretizeerd met de eindige elementen methode. Het resulterende lineaire stelsel is zeer groot en dit wordt opgelost met de *Generalized Conjugate Residual* (GCR) methode. The matrix is slecht geconditioneerd wat over het algemeen resulteert in trage convergentie van Krylov-methoden. Het is daarom noodzakelijk dat een preconditioner wordt geconstrueerd om het convergentie gedrag te verbeteren. Er wordt een start gemaakt met de toepassing van de *shifted Laplace operator* preconditionering (onlangs ontwikkeld als robuuste en efficiente preconditionering voor de Helmholtz vergelijking) voor de gediscretizeerde vector golf vergelijking.

Met behulp van de bovengenoemde preconditionering is het mogelijk om stelsels met een aantal vrijheidsgraden $N\text{dof} \leq 3.0 \cdot 10^5$ op te lossen. Wanneer het aantal vrijheidsgraden verder toeneemt, dan is het niet meer mogelijk om de gehele basis voor de Krylov deelruimte op te slaan in het core geheugen. Daarom wordt er een snelle harde schijf gebruikt om grote problemen op te lossen. Zodra een nieuwe Krylov basisvector wordt geconstrueerd moet deze georthogonaliseerd worden met betrekking tot de vorige basisvectoren. Deze vectoren zijn tijdens vorige

iteraties weggeschreven naar een file op de snelle harde schijf en worden nu in batches ingelezen. De overdracht van data van en naar core en de snelle harde schijf gaat zo snel, dat de CPU tijd ongeveer gelijk blijft voor kleine problemen.

De GCR methode met preconditionering wordt vergeleken met de al bij het NLR aanwezige *frontal solver*. Het zal aannemelijk worden gemaakt dat het mogelijk is om met de nieuwe iteratieve methode sneller een oplossing te bepalen dan met de huidige frontal solver, voor problemen met een groot aantal vrijheidsgraden op de opening in vergelijking met het totaal aantal vrijheidsgraden.

In de toekomst kan de nieuwe GCR methode met preconditionering gebruikt worden om veel grotere problemen op te lossen dan mogelijk is met de huidige frontal solver.

Acknowledgments

This work could not have been finished without the support of a number of people.

I would like to thank the NLR to let me perform my Master thesis there and in particular the manager of the division Flight Physics and Loads, ir. Koen de Cock. I also would like to thank my direct supervisor dr. ir. Duncan van der Heul for leading me in the field of radar technology and pointing out the direction of my survey. His suggestions with respect to the numerical experiments could nearly always be used immediately. Because I had to learn to program in the programming language Fortran90, I also would like to thank dr. Harmen van der Ven who was always there to help me with the programs I wrote. He also taught me how to communicate with the NEC SX-8R computer which I have been using extensively during the last month of the thesis.

For almost six months my roommate was Bert-Jan Steerneman, a fellow internship colleague who I owe a great one, since he was always there to help me with even the smaller things.

Also I would like to thank prof. dr. ir. Kees Vuik for his guidance from Delft and his expertise in numerical methods. His comments on the report were of great value to me. Every time I handed in a new version of the report it came back completely commented.

Then I would like to thank my close friends and family, especially my brother Maarten (aka Tand) who is also my roommate at home. The support from him and the relaxing weekends we had including numerous poker events helped me to relax and do other things than only math. Finally I would like to thank my parents for their (financial) support during al these years and the listening ears in frustrating times. Especially I would like to thank my dad for the conservations we had about math at home that no one of the family understood.

Contents

(93 pages in total)

List of tables

List of figures

Preface

This is the Master thesis for the degree of Master of Science in Applied Mathematics, faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology. The graduation is performed in the unit of Numerical Analysis and has a duration of nine months.

The Master thesis is carried out at the National Aerospace Laboratory (NLR) in Amsterdam. This institute is the key center of knowledge and experience for aerospace technology in the Netherlands. The main part of the research performed in this thesis is about the solution procedure for a large system of linear equations obtained from the finite element discretization of the vector wave equation.

The examination committee for this Master thesis consists of

Prof. dr. ir. C. Vuik	(Delft University of Technology)
Dr. ir. D.R. van der Heul	(National Aerospace Laboratory NLR)
Dr. ir. H.X. Lin	(Delft University of Technology)

Amsterdam, October 2007 Pim Hooghiemstra

1 The radar signature of a typical fighter aircraft

RADAR (**Ra**dio **D**etection **A**nd **R**anging) is technology to detect aircraft and ships by means of electromagnetic waves. Since the early days of radar, a large effort has been put into improving radar to increase the likelihood of detection of airborne platforms. At the same time platform developers have tried to avoid detection of their aircraft. Reduction of the probability of detection of a platform can be achieved by shape optimization and application of special radar absorbing coatings.

The radar signature of an aircraft may be measured but if this is unpractical for reasons of time or costs, e.g. if the platform is still in the development phase or when it belongs to a hostile party, measuring the radar signature is impossible. Therefore, theoretical radar signature prediction techniques have been developed to be able to quantify the detectability of the aircraft by radar in an early stage of the development. A measure for this is the so called *radar cross section* (RCS). In the definition of the RCS it is assumed that the object reflects the electromagnetic wave isotropically which is not the case physically. Theoretically, the RCS is the ratio of the scattered electric field obtained from the target and the incident field send out by the radar installation, normalized with the distance R between the radar installation and the target. In practice, the far-field scattered field is computed from the induced current distributions where after the RCS, proportional to the norm of the far-field squared, is computed. Apart from simple geometrical shapes (e.g. a flat plane or cone) it is in general not possible to compute the RCS exactly and the scattered electric field is determined numerically. The unit of RCS is area ($[m^2]$), but since the RCS of a complex object typically varies several orders of magnitude for different incident angles, it is commonly expressed in ($[dB\, m^2]$).

It is known that the jet engine air intake of a typical fighter aircraft, a forward facing cavity, accounts for the major part of the radar cross section for a large angular region, when excited from the front. This will be made intuitive by the following example. Consider two electric objects: A cylindrical tube and a flat plate shaped exactly like the aperture of the tube as illustrated by Figure 1.1. If the flat plate (left) is excited for various angles ϕ, the corresponding radar cross section has a small peak near $\phi = 0$, but for other angles the radar cross section is nearly zero since the scattered field will not be received by the radar receiver. On the other hand, if the cylindrical tube is excited, the radar cross section is relatively high for a wide ranges of excitation angles (see Figure 1.2). The effect of this difference in radar cross section on the probability of detection is that the flat plate is only visible for a small range of observation angles while the cylindrical tube is visible for a large angular region.

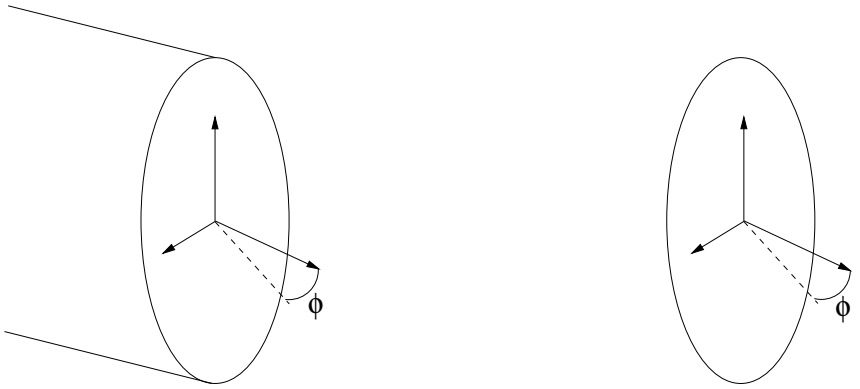

Fig. 1.1 A cylindrical tube (left) and a flat plate (right), where the area of the flat plate is exactly the area of the aperture of the tube.

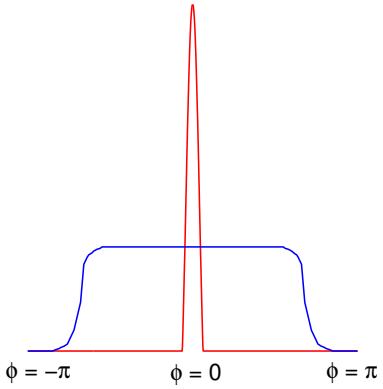

Fig. 1.2 The radar cross section for the flat plate (red) and the cylindrical tube (blue). The horizontal axis represents the excitation angle ϕ, the vertical axis is the radar cross section.

The electric field scattered by the jet engine air intake can be computed by solving the *vector wave equation* obtained from the Maxwell's equations with the appropriate boundary conditions. This equation is discretized using the finite element method, which gives rise to a large system of linear equations. Solving techniques for these kind of matrices can be found in Saad (Ref. 1) and Golub and V. Loan (Ref. 2).

The system matrix is complex valued with a sparsely and a fully populated part. In the present

algorithm to determine the solution, a direct method based on Gaussian elimination is used to solve the system. This approach is accurate, but very computer time and memory consuming.

The purpose of this thesis is to investigate alternative linear solvers to speed up the solution process of this system. If a faster method is derived, it is possible to treat larger linear systems and to obtain more accurate predictions of the radar cross section. Therefore, iterative methods will be introduced. Since the systems are in general ill conditioned several preconditioners are introduced to speed up the convergence rate of the iterative method under consideration.

Chapter 2 considers the mathematical modeling of cavity scattering. This includes the description of the domain of interest, the governing equations with appropriate boundary conditions and the finite element method. The choice of elements to discretize the domain is discussed and basis- and test functions are introduced. The result of the discretization is a large linear system which is characterized in Chapter 3. The main part of this thesis, however, is the solution procedure. In Chapter 4 the most promising iterative method (GCR) as discussed in Hooghiemstra (Ref. 3) to solve the system is described, and two preconditioners are introduced. It will be shown that the nested GCR method using the two preconditioners will result in a CPU time consuming method. Therefore, the two preconditioners will be combined into one. After this, several important parameters are tuned using various numerical experiments. When the method is completely optimized it is applied to find the contribution to the radar signature of a bended jet engine air intake of intermediate size. The iterative method will be compared to the direct method used in the current algorithm. The following chapter is devoted to recommendations. Finally, a conclusion is given based on the theoretical and numerical results in Chapter 8. Appendix A contains the meshes of several model problems used throughout the text.

2 Modeling cavity scattering of a jet engine air intake

2.1 Introduction

In this chapter a comprehensive description of the mathematical model for cavity scattering is given. For further details see Hooghiemstra (Ref. 3), Chapters 3 and 5. The computation of the radar cross section is a two stage process:

1. The computation of the electric field inside the cavity and on the cavity aperture.
2. The actual computation of the RCS: The electric field on the aperture is used to determine the far-field scattered field and eventually the radar cross section.

To start, the domain of interest is given where after the governing equations, (i.e. Maxwell's equations) are introduced. The *vector wave equation* is derived from these equations and made dimensionless afterwords. Appropriate boundary conditions are given and the weak formulation of the problem that forms the basis of the finite element discretization is shortly discussed. The discretization elements (tetrahedra and triangles) are chosen, followed by the test- and basis functions. The order of the basis functions (also called the element order) is discussed and a system of linear equations is derived. When the electric field on the aperture is determined, the actual computation of the RCS is performed using the far-field scattered field.

2.2 Step 1: Determination of the electric field on the cavity aperture

The first step of the computation of the radar cross section is concerned with the determination of the electric field inside an on the aperture of the cavity. First, the domain is described and the governing equations of cavity scattering are given. The weak formulation of the vector wave equation is derived and the finite element discretization is discussed.

2.2.1 Domain description

From this point the jet engine air intake of a modern fighter aircraft is abstracted to a cylindrical, one-side open ended cavity with a depth-to-diameter ratio (L/d) of approximately 10, where L is the length of the cavity and d is the diameter of the (geometrical) cross section of the cavity. The aperture of the cavity lies in an infinite, perfect conducting ground plane[1]. The domain of interest Ω consists of the cavity (Ω_c) and the entire half space $x > 0$ (Ω_h). This is illustrated in Figure 2.1.

2.2.2 Governing equations

The vector wave equation can be derived from the Maxwell equations. Therefore they are stated here for completeness. In differential form for a general domain Ω the Maxwell equations are

[1]This assumption is justified by the fact that the aircraft can be regarded as an electrically large object.

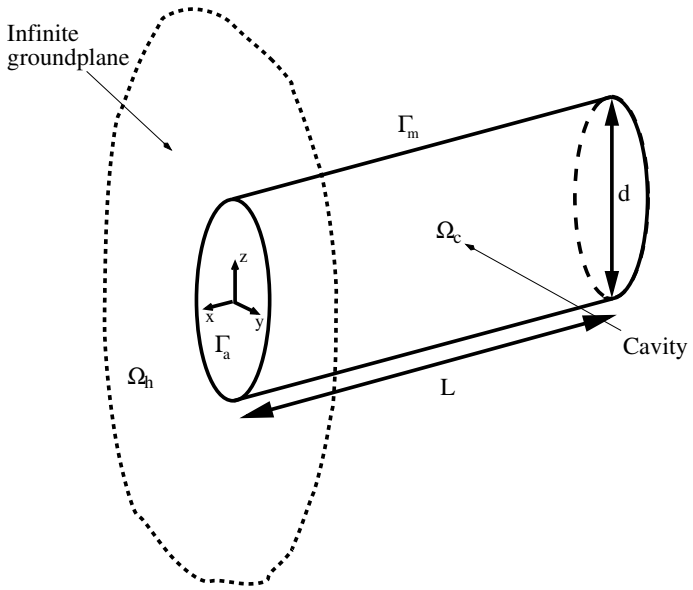

Fig. 2.1 Side view of the cylindrical cavity with length L and cross section diameter d. The surrounding of the aperture is the entire half space $x > 0$.

given by

$$\nabla \times \mathcal{E} = -\frac{\partial \mathcal{B}}{\partial t}, \tag{2.1}$$

$$\nabla \times \mathcal{H} = \frac{\partial \mathcal{D}}{\partial t} + \mathcal{J}, \tag{2.2}$$

$$\nabla \cdot \mathcal{D} = \mathcal{Q}, \tag{2.3}$$

$$\nabla \cdot \mathcal{B} = 0, \tag{2.4}$$

$$\nabla \cdot \mathcal{J} = -\frac{\partial \mathcal{Q}}{\partial t}, \tag{2.5}$$

where the following variables are used:

$\mathcal{E}$ = electric field intensity $[V/m]$,

$\mathcal{D}$ = electric flux density $[C/m^2]$,

$\mathcal{H}$ = magnetic field intensity $[A/m]$,

$\mathcal{B}$ = magnetic flux density $[Wb/m^2]$,

$\mathcal{J}$ = electric current density $[A/m^2]$,

$\mathcal{Q}$ = electric charge density $[C/m^3]$,

t = time $[s]$.

Because the radar installation operates at a single frequency (10GHz), the problem is translated to the frequency domain. This means that all field quantities are considered time-harmonic and equations (2.1), (2.2) and (2.5) become

$$\nabla \times \mathbf{E} = -j\omega\mathbf{B}, \tag{2.6}$$

$$\nabla \times \mathbf{H} = j\omega\mathbf{D} + \mathbf{J}, \tag{2.7}$$

$$\nabla \cdot \mathbf{J} = -j\omega q. \tag{2.8}$$

The quantities $\mathbf{E}, \mathbf{D}, \mathbf{H}, \mathbf{B}, \mathbf{J}$ and q are the *phasor* quantities corresponding to the variables $\mathcal{E}, \mathcal{D}, \mathcal{H}, \mathcal{B}, \mathcal{J}$, and $\mathcal{Q}$ in the Maxwell equations. As an example consider the electric field intensity $\mathcal{E}$. The relation between the phasor $\mathbf{E}$ and the field quantity $\mathcal{E}$ is given by

$$\mathcal{E} = \mathbf{E}(\mathbf{x})e^{j\omega t}. \tag{2.9}$$

Equations (2.6)-(2.8) are completed by the introduction of the macroscopic properties of the medium: the *constitutive relations*. These relations are given by

$$\mathbf{D} = \varepsilon(\mathbf{x})\mathbf{E}, \tag{2.10}$$

$$\mathbf{B} = \mu(\mathbf{x})\mathbf{H}, \tag{2.11}$$

$$\mathbf{J} = \sigma(\mathbf{x})\mathbf{E}, \tag{2.12}$$

where $\varepsilon,\ \mu,\ \sigma$ are the permittivity [farads/meter], the permeability [henrys/meter] and the conductivity [siemens/meter] respectively. The parameters ε and μ are written as the product of the value for vacuum and a relative constant (e.g. $\varepsilon = \varepsilon_0\,\varepsilon_r,\ \ \mu = \mu_0\,\mu_r$). For simple mediums, these parameters are just real valued constants. However, if radar absorbing materials are used, like radar absorbing foam, the permittivity will be complex valued, $\varepsilon \in \mathbb{C}$.

Using the previous relations, the vector wave equation can be derived see Hooghiemstra (Ref. 3), page 23 for details

$$\nabla \times \left(\frac{1}{\mu_r}\nabla \times \mathbf{E}\right) - k_0^2\varepsilon_r\mathbf{E} = -jk_0Z_0\mathbf{J}. \tag{2.13}$$

In this equation the free-space wavenumber is defined by $k_0 = \omega\sqrt{\varepsilon_0\,\mu_0}$ and the free-space impedance is given by $Z_0 = \sqrt{\frac{\mu_0}{\varepsilon_0}}$. If there is no current source $\mathbf{J}$, the right-hand side of equation (2.13) vanishes and the homogeneous vector wave equation is obtained.

2.2.2.1 Boundary conditions

Equation (2.13) has a unique solution if appropriate boundary conditions are provided on the boundary of the domain, Γ. This boundary consists of two parts: The mantle (including the bottom) of the cavity (Γ_m) and the aperture (Γ_a) (Figure 2.1). Since the mantle of the cavity is a perfect conductor, the appropriate boundary condition to apply is a homogeneous Dirichlet condition on the tangential electric field

$$(\hat{n} \times \mathbf{E})_{\Gamma_m} = 0. \tag{2.14}$$

The boundary condition on the aperture Γ_a is derived in Hooghiemstra (Ref. 3), p. 24-26. This condition uses the incoming magnetic field $\bar{H}^{\text{inc}}_{i,j}$ where the subscripts i, j denote the excitation angle $(\phi_i,\, \theta_j)$. For a fixed excitation angle this boundary condition is given by:

$$\hat{n} \times \bar{H}^{\text{inc}}_{i,j} = -4\hat{n} \times \left\{ \frac{\nabla\nabla \cdot \mathbf{N} + k^2\mathbf{N}}{j\omega\mu_0} \right\}_{\Gamma_a}, \tag{2.15}$$

where $\mathbf{N}$ is the vector potential produced by the tangential electric field $\mathbf{K}_2 (= \hat{n} \times \mathbf{E}_2)$ radiating in free space in region 2. From Peterson et al. (Ref. 4) it follows that $\mathbf{N}$ is equal to

$$\mathbf{N} = \mathbf{K}_2 * G, \tag{2.16}$$

where $G(\mathbf{r})$ is the three dimensional Green's function

$$G = \frac{e^{-jk|\mathbf{r}|}}{4\pi|\mathbf{r}|}. \tag{2.17}$$

In the general case the contribution of the jet engine air intake to the radar cross section is determined for a wide angular region. This results eventually in a problem with multiple right-hand sides. For now, it is assumed that only one excitation angle is treated hence, i and j are fixed. See Chapter 7 for more details concerning multiple right-hand sides.

2.2.3 Weak formulation

Before the weak formulation is derived, the vector wave equation is made dimensionless. This is done in order to find the most important parameters for this equation. The dimensionless vector wave equation is given by Hooghiemstra (Ref. 3) p. 27,28

$$\nabla \times \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) - k_0^2 \varepsilon_r \mathbf{E} = -jk_0 Z_0 \mathbf{J}, \tag{2.18}$$

where the same variables are used to denote the dimensionless quantities. From now on this dimensionless equation is used throughout this report. The key parameter of the vector wave equation is the dimensionless wavenumber k_0.

Equation (2.18) is multiplied by a testfield $\mathbf{T}$ in the class of vectorfields Σ satisfying the essential boundary condition

$$\Sigma := \{\mathbf{T} : (\hat{n} \times \mathbf{T})_{\Gamma_m} = 0\}, \tag{2.19}$$

and integrated over the domain Ω. This yields

$$\int_\Omega \mathbf{T} \cdot \left\{ \nabla \times \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) - k_0^2 \varepsilon_r \mathbf{E} \right\} \mathrm{d}\Omega = -jk_0 Z_0 \int_\Omega \mathbf{T} \cdot \mathbf{J} \mathrm{d}\Omega. \tag{2.20}$$

Using the second Green vector theorem (see for example Jin (Ref. 5), appendix) the first rotation operator $(\nabla \times .)$ is moved to the testfunction

$$\int_\Omega (\nabla \times \mathbf{T}) \cdot \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) \mathrm{d}\Omega - \oint_\Gamma \left\{ \mathbf{T} \times \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) \right\} \cdot \hat{n} \, \mathrm{d}\Gamma - \int_\Omega k_0^2 \varepsilon_r \mathbf{T} \cdot \mathbf{E} \, \mathrm{d}\Omega = -jk_0 Z_0 \int_\Omega \mathbf{T} \cdot \mathbf{J} \mathrm{d}\Omega. \tag{2.21}$$

The boundary integral is split in two parts, one for the mantle of the cavity and one for the aperture:

$$\oint_\Gamma \left\{ \mathbf{T} \times \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) \right\} \cdot \hat{n} \, \mathrm{d}\Gamma = \oint_{\Gamma_m} \left\{ \mathbf{T} \times \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) \right\} \cdot \hat{n} \, \mathrm{d}\Gamma + \oint_{\Gamma_a} \left\{ \mathbf{T} \times \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) \right\} \cdot \hat{n} \, \mathrm{d}\Gamma. \tag{2.22}$$

The last term in this equation, corresponding to the aperture, results eventually in a fully populated block in the discretization matrix (see Hooghiemstra (Ref. 3) and Peterson (Ref. 4)). The integral over the cavity mantle is rewritten as (Ref. 3)

$$-\oint_{\Gamma_m} (\mathbf{T} \times \hat{n}) \cdot \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) \mathrm{d}\Gamma = 0, \tag{2.23}$$

where the latter equality is due to the essential boundary condition. Combining all results, the weak formulation of equation (2.18) is

$$\int_\Omega \left\{ (\nabla \times \mathbf{T}) \cdot \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) - k_0^2 \varepsilon_r \mathbf{T} \cdot \mathbf{E} \right\} \mathrm{d}\Omega - \oint_{\Gamma_a} \hat{n} \cdot \left\{ \mathbf{T} \times \left(\frac{1}{\mu_r} \nabla \times \mathbf{E} \right) \right\} \mathrm{d}\Gamma = -jk_0 Z_0 \int_\Omega \mathbf{T} \cdot \mathbf{J} \, \mathrm{d}\Omega \tag{2.24}$$

This weak formulation is used to define the functional $F(\mathbf{E})$ to be minimized (Refs. 3, 5), which is given by

$$F(\mathbf{E}) = \frac{1}{2}\int_{\Omega}\left[\frac{1}{\mu_r}(\nabla\times\mathbf{E})\cdot(\nabla\times\mathbf{E}) - k_0^2\varepsilon_r\,\mathbf{E}\cdot\mathbf{E}\right]dV$$
$$- k_0^2\iint_{\Gamma_a}[\hat{z}\times\mathbf{E}(\mathbf{r})]\cdot\left\{\iint_{\Gamma_a}[\hat{z}\times\mathbf{E}(\mathbf{r}')]G_0(\mathbf{r},\mathbf{r}')\,dS'\right\}dS$$
$$+\iint_{\Gamma_a}\Big[\nabla\cdot[\hat{z}\times\mathbf{E}(\mathbf{r})]\Big]\left\{\iint_{\Gamma_a}G_0(\mathbf{r},\mathbf{r}')\nabla'\cdot[\hat{z}\times\mathbf{E}(\mathbf{r}')]\,dS'\right\}dS$$
$$+ 2jk_0Z_0\iint_{\Gamma_a}[\hat{z}\times\mathbf{E}(\mathbf{r})]\cdot\mathbf{H}^{inc}(\mathbf{r})\,dS. \quad (2.25)$$

2.2.4 The finite element method

The next step is to expand the electric field $\mathbf{E}$ and the testfield $\mathbf{T}$ in a linear combination of basis functions and test functions, respectively. In general identical test- and basis functions are chosen and this strategy is adopted here. The expansions are substituted in the weak formulation (2.24) to obtain a linear system of equations. First, the domain Ω is discretized and suitable basis functions have to be found.

2.2.5 Domain discretization

The interior of the cavity is discretized with rectilinear tetrahedral elements because this allows a much more effective evaluation of the element matrices. As a result of this, the aperture of the cavity is discretized with rectilinear triangles. These elements are chosen because they easily conform to the shape of the cavity and hence, reduce the error which is introduced during the discretization of the domain.

2.2.6 Construction of basis functions

In order to prevent the occurrence of nonphysical solutions (called *spurious* solutions), vector basis functions of order $p \geq 0$ will be introduced. The linear or zeroth order ($p = 0$) basis functions were found by Nedelec (Refs. 6). Higher order ($p \geq 1$) basis functions were introduced by Graglia et al. (Ref. 7).

The basis function ($\mathbf{N}_i$) defined on edge i should satisfy the following conditions:

- $\nabla\cdot\mathbf{N}_i = 0$ (since $\nabla\cdot\mathbf{E} = 0$).
- $\mathbf{N}_i$ has a constant tangential component along the ith edge (a necessary condition for inter-elemental continuity of the expansion of the electric field).

To reach these specified requirements, it is necessary to number the nodes and direct the edges of the tetrahedral elements. This is illustrated in Figure 2.2.

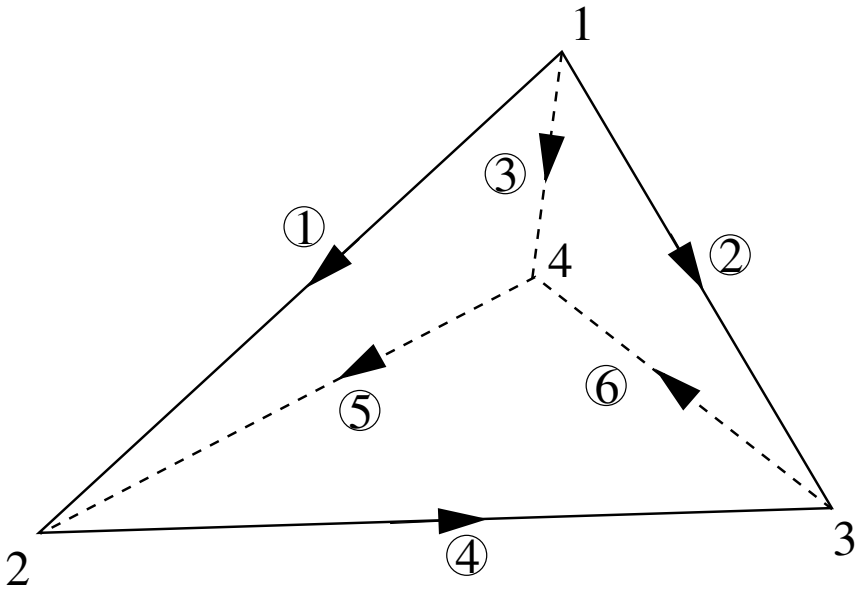

Fig. 2.2 The ordering and direction of the edges and nodes for a tetrahedral element.

As described in Hooghiemstra (Ref. 3), the edge basis functions for a tetrahedral element are constructed using the area coordinates denoted by $(L_1, \ldots, L_4)$. The zeroth basis function $\mathbf{N}_i^0$ (for edge i) is then written as

$$\mathbf{N}_i^0(\mathbf{x}) = l_i \, \mathbf{W}_{i_1,i_2}(\mathbf{x}), \tag{2.26}$$

where l_i is the length of edge i. i_1 and i_2 are the nodes connected by edge i as illustrated in Figure 2.2 and

$$\mathbf{W}_{i_1,i_2}(\mathbf{x}) = L_{i_1}(\mathbf{x}) \nabla L_{i_2}(\mathbf{x}) - L_{i_2}(\mathbf{x}) \nabla L_{i_1}(\mathbf{x}). \tag{2.27}$$

It is observed that these basis functions satisfy the requirements given above (Ref. 3).

2.2.6.1 Higher order basis functions

A basis function of order p for edge i is constructed in the following way

$$\mathbf{N}_i^p(\mathbf{x}) = P_p(\mathbf{x}) \mathbf{N}_i^0(\mathbf{x}), \tag{2.28}$$

where $P_p(\mathbf{x})$ is a complete polynomial of order p consisting of all terms $x_1^k x_2^l x_3^m$ such that $k+l+m = p$. To construct this polynomial, the Lagrange interpolatory polynomials (Ref. 5) combined with (shifted) Silvester polynomials are used (Refs. 3, 8). A pth order basis function for edge i is given by

$$\mathbf{N}_i^p(\mathbf{x}) = \mathbf{N}_{ijkl}^{i_1\, i_2}(\mathbf{x}) = \alpha_{ijkl}^{i_1\, i_2} \frac{(p+2)^2}{\gamma\, \beta} \xi_{i_3} \xi_{i_4} \hat{P}_i^{p+2}(\xi_1) \hat{P}_j^{p+2}(\xi_2) \hat{P}_k^{p+2}(\xi_3) \hat{P}_l^{p+2}(\xi_4) \mathbf{W}_{i_1\, i_2}(\mathbf{x}),$$

(2.29)

where i_3 and i_4 are integers among $(1, 2, 3, 4)$ other than i_1 and i_2 and $\gamma(\beta)$ is taken to be i, j, k or l for $i_3(i_4) = 1, 2, 3$ or 4 respectively.

To construct the higher order vector basis functions for an edge, interpolation points on this edge and on the faces of the tetrahedron that coincide at this edge are used. For each interpolation point on an edge there is one basis function, but for interpolation points on a face three basis functions are defined (since the face has three edges). The tangential electric field on the face is spanned by two independent basis functions, so for a point on the face where three basis functions are defined, one of them must be discarded since it is dependent. For an interior point six basis functions for each interpolation point in the interior are defined. In the same way as for a face there are only three independent basis functions for such a point, so the other three are discarded. However, this must be done with care. To be independent, the three chosen basis functions should not have all the zeroth order basis functions associated with edges bounding the same face (Refs. 3, 9). The number of interpolation points needed increases rapidly as the element order increases. A second order element has 45 interpolation points, a fifth order element even more: 216.

2.2.7 Application of Galerkin's method

When the basis functions are defined and constructed, the electric- and testfield are expanded into a linear combination of these functions per element. Hence,

$$\mathbf{E}^e(\mathbf{x}) = \sum_{i=1}^{n} E_i^e \mathbf{N}_i^e(\mathbf{x}) = \{E^e\}^\top \{\mathbf{N}^e(\mathbf{x})\}. \tag{2.30}$$

Here n is the number of local basis functions per element. The curly brackets $\{.\}$ are used to indicate a vector with components which are vectors themselves. As mentioned before, by subdividing the volume of the cavity into M small volume elements, the aperture is divided into M_s small surface elements. The surface field expansion is given by

$$\hat{z} \times \mathbf{E}^s(\mathbf{x}) = \sum_{i=1}^{n_s} E_i^e \mathbf{S}_i^s(\mathbf{x}) = \{E^e\}^\top \{\mathbf{S}^s(\mathbf{x})\}, \tag{2.31}$$

where n_s is the number of local basis functions for the triangular elements. For a compatible expansion $\mathbf{S}_i^s = \hat{z} \times \mathbf{E}_i^s$. Substituting both the volume and surface expansions into the functional yields

$$F = \frac{1}{2} \sum_{e=1}^{M} \{E^e\}^\top [K^e] E^e + \frac{1}{2} \sum_{s=1}^{M_s} \sum_{t=1}^{M_s} \{E^s\} [P^{st}] \{E^t\} - \sum_{s=1}^{M_s} \{E^s\}^\top \{b^s\}. \tag{2.32}$$

In equation (2.32) the following matrices have been defined:

$$
\begin{aligned}
[K^e] &= \iiint_{V^e} \left[\frac{1}{\mu_r} \{\nabla \times \mathbf{N^e}\} \cdot \{\nabla \times \mathbf{N^e}\} - k_0^2 \varepsilon_r \{\mathbf{N}^e\} \cdot \{\mathbf{N}^e\} \right] dV, \\
\{b^s\} &= -2jk_0 Z_0 \iint_{S^s} \{\mathbf{S^s} \cdot \bar{H}^{\text{inc}}\}\, dS.
\end{aligned}
$$

The matrix $[P^{st}]$ is obtained from the boundary integral and has the form

$$
[P^{st}] = 2 \iint_{S^s} \{\nabla \cdot \mathbf{S^s}\} \left\{ \iint_{S^t} \{\nabla' \cdot \mathbf{S^t}\}^\top G_0 \, dS' \right\} dS - 2k_0^2 \iint_{S^s} \{\mathbf{S^s}\} \cdot \left\{ \iint_{S^t} \{\mathbf{S^t}\}^\top G_0 \, dS' \right\} dS, \quad (2.33)
$$

The first two integrals ($[K^e], \{b^e\}$) are computed numerically using Gauss' Quadrature formulas. For the evaluation of the integral in the matrix $[P^{st}]$ a method described by Rao et al. (Ref. 10) is employed that uses Duffy's method (see Jin (Ref. 5) appendix D) to get rid of the singularity in the Green's function.

2.3 Step 2: The far-field scattered field and the determination of the radar cross section

In the end, when the system is solved, the electric field $\mathbf{E}$ and the magnetic current $\hat{n} \times \mathbf{E}$ are known on the aperture. The latter quantity determined the far-field scattered field $\mathbf{E}^s$ that is used to determine the radar cross section. According to Sommerfeld's radiation condition, the far-field $\mathbf{E}^s$ has no radial component and the angular components E^s_θ and E^s_ϕ are given by the following two equations

$$
\begin{aligned}
E^s_\theta &= (-jk_0 Z_0)(A_\theta + \tfrac{F_\phi}{Z_0}), \\
E^s_\phi &= (-jk_0 Z_0)(A_\phi - \tfrac{F_\theta}{Z_0}).
\end{aligned} \quad (2.34)
$$

The functions A and F used here are potential functions for the electric current and the magnetic current, respectively. Since in this case the electric current is zero by definition, A vanishes and F is given by (Ref. 11)

$$
F(\mathbf{r}) = \frac{e^{-jk_0 \mathbf{r}}}{4\pi\, \mathbf{r}} \int_{\Gamma_a} [\hat{n} \times \mathbf{E}(\mathbf{r}')] e^{jk_0 \mathbf{r}' cos(\psi)} dS'. \quad (2.35)
$$

Using the fact that A is zero, the far-field scattered field components are written as

$$
\begin{aligned}
E^s_\theta &= -jk_0 Z_0\, F_\phi, \\
E^s_\phi &= jk_0 Z_0\, F_\theta.
\end{aligned} \quad (2.36)
$$

The RCS follows as

$$\sigma = \frac{4\pi\, R\, ||E_s||^2}{||E_i||} = 4\pi\, R\, ||E_s||^2, \tag{2.37}$$

where the latter equality follows from the fact that the system is excited by a normalized incident field.

3 Analyzing the linear system obtained from the finite element discretization

3.1 Introduction

In this chapter the focus is on the linear system obtained from minimizing the functional F in equation (2.32). After minimizing this functional a complex valued system matrix is derived with both a sparsely and a fully populated part. The origin of these blocks will be explained and some important properties of this kind of matrix are presented, based on an example matrix corresponding to a small problem.

The size of the matrix corresponding to the target problem depends on the meshwidth used in the discretization. It turns out that a small meshwidth is necessary and it will be explained why. Given the meshwidth an approximation of the number of degrees of freedom in the problem which is equal to the matrix size can be given. It is assumed throughout this chapter that the domain considered here is a rectangular box with dimensions $d \times d \times L$, where $d = 1.5$ and L is altered.

3.2 The sparsity pattern of the finite element discretization matrix

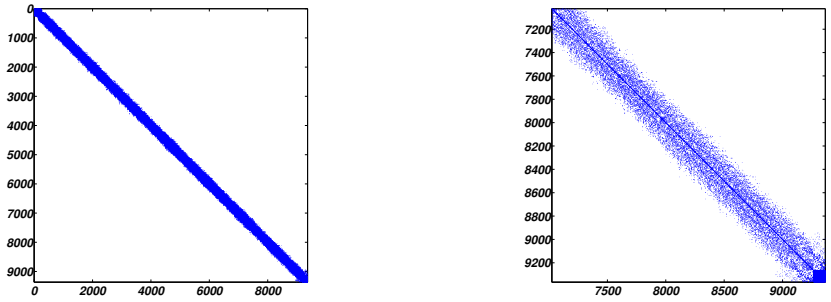

Fig. 3.1 The sparsity pattern of the second order system matrix A for meshwidth $h = 0.25$, and length $L = 8$. The corresponding mesh is illustrated in Figure 3.2.

As can be seen in Figure 3.1, the matrix can be divided into a sparsely populated part and a fully populated part. The fully populated part corresponds to the boundary condition on the aperture (2.15). The sparse part stems from the application of the finite element method inside the cavity. The system matrix is symmetric and ill-conditioned. A thorough investigation of the spectrum of the system matrix has not been performed in Matlab since the matrix corresponding to the target problem is to big to load in Matlab. For a small problem it actually is investigated in Hooghiemstra (Ref. 3) where it is found that the matrix is indefinite. Thereby, it can be shown (Ref. 3) that the eigenvalues are not clustered at all. These last three properties have as a consequence that

iterative methods will converge slowly unless a robust and efficient preconditioner is used. However, these properties are a guideline only. Since the cavity considered here is not a deep cavity, the aperture is much more dominant than is the case for the target problem. This difference is characterized by the ratio $\nu = \frac{\text{full block size}}{\text{sparse block size}}$. For a deep cavity (the target problem considers a deep cavity as mentioned in Section 2.2.1) it holds that

$$\nu_T << \nu_{example}. \tag{3.1}$$

The matrix as illustrated in Figure 3.1, corresponds to a rectangular domain Ω which is depicted in Figure 3.2. A special numbering of the degrees of freedom is applied in order to have all degrees of freedom belonging to the aperture in the lower right corner of the matrix.

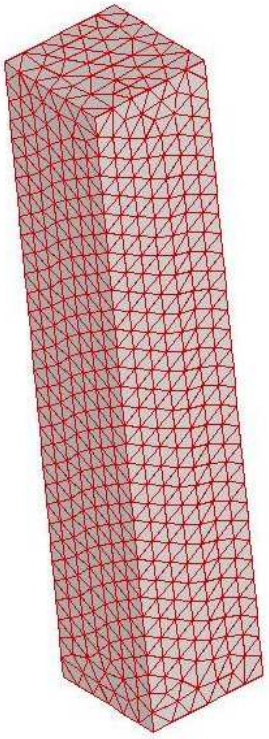

Fig. 3.2 The rectangular domain Ω, discretized with tetrahedra, the boundary is discretized using triangles.

3.3 Estimating the size of the system matrix

In order to calculate the dimensions of the system matrix, it is necessary to know the sizes of the domain (L, d) and the meshwidth (h) used to discretize this domain. The meshwidth is calculated as a function of the total accumulated dispersion error ε, the wavenumber k_0 and the element order p in the next section.

3.3.1 Generating the grid

When the dimensions of the domain are fixed, the grid itself is generated. This must be done with care however, since a cavity of dimensions $1.5 \times 1.5 \times 0.6$ can not be discretized with elements with meshwidth $h = 0.50$ for example. Doing this nonetheless, may cause stretched elements to occur as illustrated in Figure 3.3.

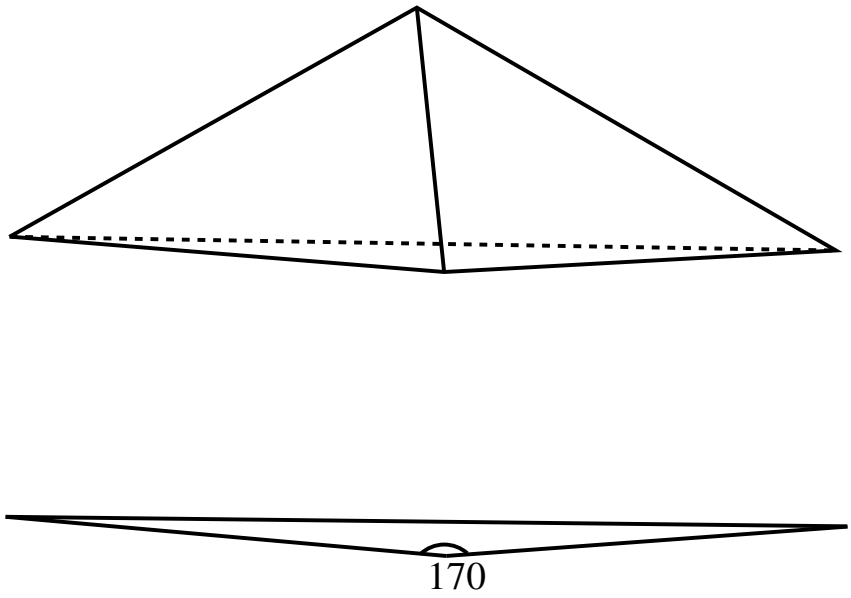

Fig. 3.3 Choosing the meshwidth too big for the domain causes stretched elements. A stretched elements has a nearly 180° *angle on the base triangle.*

These kind of elements, especially those with nearly 180° angles between the edges, increase the condition number of the system matrix. As already mentioned, a high condition number implies an ill-conditioned matrix which influences the convergence behavior of Krylov-subspace based iterative methods. The relation between badly shaped elements and a high condition number has been tested using the grid generator for a small rectangular cavity of dimensions $1.5 \times 1.5 \times 0.6$. Using Matlab, the condition number of the discretization matrix A has been estimated and the number of iterations to reach the prescribed tolerance, was compared for various meshwidths. The results of this test are presented in Table 3.1. From this table it is concluded that stretched elements cause an unacceptably high conditionnumber.

3.3.2 The meshwidth h

Finite accuracy of the discretization introduces a dispersion error ε in the electric field expansion. For deep cavities ($L/d \approx 10$) this error accumulates as the electromagnetic wave propagates a long way through the cavity and becomes significant. It is the dispersion error that affects the phase difference between contributions of different parts of the cavity. To be able to compute

h	$\kappa_{est}(A)$	# iterations	angles $\geq 170^\circ$
0.50	$8.14 \cdot 10^9$	1437	Y
0.40	$1.48 \cdot 10^{10}$	2400	Y
0.35	$2.16 \cdot 10^5$	979	N
0.30	$1.43 \cdot 10^5$	1003	N

Table 3.1 *The presence of badly shaped elements has an unfavourable effect on the condition number of the system matrix and therefore, on the convergence of the iterative methods.*

the RCS accurately, the phase difference must be preserved during the numerical approximation. Hence, minimizing the dispersion error yields an accurate solution. The dispersion error is a function of both the meshwidth as well as the wavenumber and given the dispersion error, it is possible to compute the corresponding meshwidth h by (Ref. 3)

$$h(k_0, p) = C_1 \left(\frac{1}{k_0}\right)^{1+\frac{1}{2(p+\alpha)}}, \tag{3.2}$$

where the constant C_1 depends on the dimensions of the cavity, the total accumulated dispersion error and the element order in the following way

$$C_1 = (p+2)2\pi\, d \left(\frac{\varepsilon\, 2\pi\, d \cos\phi}{2L}\right)^{\frac{1}{2(p+\alpha)}}. \tag{3.3}$$

When the meshwidth is known, the number of elements inside the cavity and on the aperture can be estimated respectively by the equations

$$N_{el}(\text{cavity}) = \frac{V_c}{V_t}, \qquad N_{el}(\text{aperture}) = \frac{A_a}{A_t}, \tag{3.4}$$

where V_c is the volume of the cavity, A_a is the area of the aperture and $V_t,\ A_t$ are respectively the volume and area of a typical tetrahedron or triangle. Assuming that both the typical tetrahedron as the triangle are regular, these quantities are calculated using the meshwidth only. This estimation is not exact however, since the assumption that all tetrahedra have the same volume is made here. In general this is not the case, since the shape and volume of tetrahedra near the boundary is changed due to the fact that the elements have to conform to the boundary. Therefore, the volume of the tetrahedra presented here corresponds to an average, regular element away from the boundary. This same argument holds for the aperture triangles also. The volume and area of such a tetrahedron or triangle is given by:

$$V_t = \frac{\sqrt{2}}{12}h^3, \qquad A_t = \frac{\sqrt{3}}{4}h^2. \tag{3.5}$$

The next step is to calculate the number of degrees of freedom in the mesh. This depends both on the total number of elements N_{el} as on the number of *unique* degrees of freedom per element. In Hooghiemstra (Ref. 3), the number of unique degrees of freedom per element is given (Chapter 7, p.78). This number depends on the element order p in the following manner

$$N\text{dof(tetrahedron)} = (2p+1)(p+1), \qquad N\text{dof(triangle)} = \frac{1}{2}(p+5)(p+1). \tag{3.6}$$

Hence, the number of degrees of freedom[1] inside the cavity and on the aperture may be calculated by

$$N\text{dof} = N_{el}(\text{cavity}) \cdot (2p+1)(p+1), \qquad N\text{dof}_{ap} = N_{el}(\text{aperture}) \cdot \frac{1}{2}(p+5)(p+1). \tag{3.7}$$

For various rectangular domains these numbers are summarized in Table 3.2 for $p = 2$. For completeness, these numbers are also computed using the grid-generator, indicated by an asterisk $(*)$.

(L, h)	Ndof*	Ndof	Ndof$^*_{ap}$	Ndof$_{ap}$	N^*_{el}	N_{el}
(0.60, 0.35)	3969	4008	354	445	230	267
(0.60, 0.25)	11739	10997	741	873	659	733
(0.60, 0.20)	23091	21478	1113	1364	1239	1431
(4.0, 0.35)	28326	26718	354	445	1636	1781
(4.0, 0.25)	80571	73313	741	873	4595	4886
(4.0, 0.20)	154656	143190	1113	1364	8709	9546
(15.0, 0.35)	109872	100190	354	445	6411	6679
(15.0, 0.25)	297945	274920	741	873	16905	18328
(15.0, 0.20)	585519	536960	1113	1364	32988	35797

Table 3.2 *The number of degrees of freedom in the cavity Ndof, on the aperture Ndof$_{ap}$ and the total number of elements in the mesh $N_{el}(cavity)$ for different values of (L, h).*

For the target problem $(L = 6.77m, d = 0.67m)$ the meshwidth and the number of degrees of freedom can be calculated now. These numbers depend heavily on the wavenumber k_0. Therefore, the meshwidth is calculated for two values of the wavenumber: $k_0 = 10$ and $k_0 = 100$. If $k_0 = 10$, the corresponding meshwidth equals 0.24, but if the wavenumber $k_0 = 100$, the meshwidth is only 0.059. The number of degrees of freedom for these two cases are respectively $N\text{dof} = 5.19 \cdot 10^5$ and $N\text{dof} = 2.03 \cdot 10^7$. The dimensionless wavenumber for the target problem is approximately $k_0 \approx 15$.

[1] In Section 4.3, the abbreviations $n = N$dof and $f = N$dof$_{ap}$ are used.

4 Application of the nested preconditioned generalized conjugate residual method to solve the linear system

4.1 Introduction

In this chapter the matrix obtained from the finite element discretization of the vector wave equation as derived in Chapter 2 is solved iteratively. The method adopted here, as described in Chapter 4 of Hooghiemstra (Ref. 3), is the GCR method (**G**eneralized **C**onjugate **R**esidual method). In Chapter 3 it is mentioned that the system is ill-conditioned, hence, a slow convergence of the method is expected. A first, simple preconditioner is proposed to speed up the convergence, but it will be shown that this preconditioner alone is not enough. Therefore a second, nested preconditioner is considered, which is closely related to the shifted Laplace preconditioner for the Helmholtz equation (see Erlangga et al. (Ref. 12) and (Refs. 13, 14, 15)).

Eventually, after doing several numerical experiments which will be described in Chapter 5, it will turn out that a combination of the two preconditioners yields a fast converging method. The algorithm will be adjusted to incorporate this new preconditioner which results in one loop not to be executed anymore.

Due to the preconditioners a nested algorithm is obtained and the total number of iterations to reach a certain tolerance depends on the different number of iterations per preconditioner solve. This number of iterations is investigated along with the total work involved. The last topic of this chapter is *vectorization*. Vectorization of the algorithm is very important for the overall performance of the method.

4.2 The nested, preconditioned GCR method

The GCR method as described in (Ref. 3) has long recurrences, but it has the *minimization property*. This means that in every iteration, the norm of the residual decreases. An explicit orthonormal basis for the Krylov subspace is constructed every iteration. Therefore, storage is required for the orthonormal basis vectors, the number of which increase linearly with the number of iterations. Since this storage becomes unacceptable if many iterations are needed, the method can be truncated or restarted[1]. This means that instead of storing the complete orthonormal Krylov basis, a part of the basis is stored. Of course this will negatively affect the rate of convergence of the method. The template for the GCR method is given in Algorithm 4.1 (Ref. 16), p.85. In the following it is assumed that the full GCR method is used unless stated differently.

[1] Truncated and restarted GCR will be treated in Sections 5.3.1 and 5.3.2.

4.2.1 A block preconditioner

Since the system matrix A is ill-conditioned, the convergence of the GCR method is slow. A preconditioner is included to improve the rate of convergence. A has a special block structure,

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{4.1}$$

where the blocks A_{11}, A_{12} ans A_{21} are sparsely populated and the block A_{22} is fully populated. The first preconditioners are build from the blocks of A. Several block preconditioners have been proposed. The first three preconditioners have two diagonal blocks:

$$M_{\text{I}} = \begin{pmatrix} A_{11} & 0 \\ 0 & I_{22} \end{pmatrix}, \quad M_{\text{II}} = \begin{pmatrix} I_{11} & 0 \\ 0 & A_{22} \end{pmatrix}, \quad M_{\text{III}} = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix}. \tag{4.2}$$

The last two preconditioners use an extra off-diagonal block:

$$M_{\text{IV}} = \begin{pmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{pmatrix}, \quad M_{\text{V}} = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}. \tag{4.3}$$

The effect on the convergence of these preconditioners is illustrated in Figure 4.1 for a model problem, where the corresponding domain is illustrated in Appendix A in Figure A.1. Zeroth order basis functions are used for this problem and the meshwidth is $h = 0.25$. It is immediately clear that the block triangular preconditioners M_{IV} and M_{V} decrease the number of iterations

---

Algorithm 4.1 Generalized Conjugate Residual method

---

in: system matrix A, right-hand side b
out: solution x

Compute initial residual $r^0 = b - Ax^0$ for some initial guess x^0.
for $k = 1, 2...$ **do**
  $s^k = r^{k-1}$
  $v^k = As^k$
  for $j = 1$ to k **do**
    $\alpha = (v^j, v^k)$
    $v^k = v^k - \alpha v^j$
    $s^k = s^k - \alpha s^j$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $x^k = x^{k-1} + (v^k, r^{k-1}) s^k$
  $r^k = r^{k-1} - (v^k, r^{k-1}) v^k$
end for

---

with a factor 6 compared to full unpreconditioned GCR (denoted as $M_0 = I$). However, to produce this figure, the preconditioner system $M_* \, s^k = r^{k-1}$ is solved directly using Matlab. This is done here to investigate the quality of the preconditioners, but in practice it is impossible to solve this system directly. According to Figure 4.1, preconditioner $M_{\rm V}$ is chosen as the block preconditioner since it yields a fast converging method. From now on, this preconditioner is referred to as preconditioner M_1 for simplicity.

$$M_1 := M_{\rm V} = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}. \tag{4.4}$$

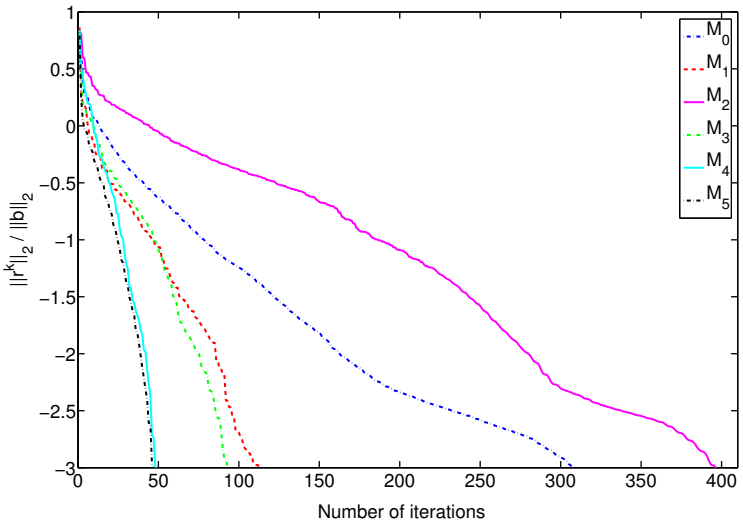

Fig. 4.1 The effect of the preconditioners constructed from the matrix A. This model problem corresponds to a domain with dimensions $1.5 \times 1.5 \times 0.6$ and a meshwidth $h = 0.25$. In this case, the element order $p = 0$.

The template for preconditioned GCR is given in Algorithm 4.2.

In this algorithm, the system $M_1 s^k = r^{k-1}$ has to be solved every iteration. This system will be referred to as *the (first) preconditioner system*. Since M_1 is an upper triangular block matrix, this system is solved by (block) backward substitution. It can be written as

$$M_1 = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix} \begin{pmatrix} s_1^k \\ s_2^k \end{pmatrix} = \begin{pmatrix} r_1^{k-1} \\ r_2^{k-1} \end{pmatrix}. \tag{4.5}$$

First, solve

$$A_{22} s_2^k = r_2^{k-1} \tag{4.6}$$

by applying an LU-decomposition to the matrix A_{22}. This LU-decomposition is cheap to perform because the block A_{22} is very small compared to the other blocks. Besides this, it has to be computed only once. Then, as s_2^k is computed,

$$A_{11}s_1^k + A_{12}s_2^k = r_1^{k-1} \tag{4.7}$$

is solved by considering the system

$$A_{11}s_1^k = r_1^{k-1} - A_{12}s_2^k. \tag{4.8}$$

The latter system has to be solved using a second preconditioned Krylov method. This will be discussed in the next section. Note however, that since the search direction s_1^k is approximated, the number of outer iterations (It_k) will increase. This growth in the number of outer iterations can be bounded if, for example, the system $M_1\, s_1^k = r^{k-1}$ is solved with a high accuracy. Then the total number of iterations would increase drastically. Hence, a careful choice must be made to bound the total number of iterations and the product of the number of middle and inner iterations at the same time. This problem will be revised in Chapter 5.

4.2.2 The shifted Laplace operator preconditioner

The system (4.8) is in general too big to solve directly and an additional iterative method (e.g. Bi-CGSTAB or GCR) is nested inside the outer GCR-iteration. The system matrix here, (A_{11}) corresponds to the discretization of the vector wave equation inside the cavity. This matrix is

Algorithm 4.2 Preconditioned Generalized Conjugate Residual method

in: system matrix A, right-hand side b, preconditioner M_1
out: solution x

Compute initial residual $r^0 = b - Ax^0$ for some initial guess x^0.
for $k = 1, 2...$ **do**
  Solve *preconditioned system* $M_1 s^k = r^{k-1}$
  $v^k = As^k$
  for $j = 1$ to k **do**
    $\alpha = (v^j, v^k)$
    $v^k = v^k - \alpha v^j$
    $s^k = s^k - \alpha s^j$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $x^k = x^{k-1} + (v^k, r^{k-1})s^k$
  $r^k = r^{k-1} - (v^k, r^{k-1})v^k$
end for

sparsely populated and is very similar to the matrix originating from the discretization of the Helmholtz equation obtained by Erlangga et al.(Ref. 12). This is explained below.

The homogeneous (scalar) Helmholtz equation is given by

$$(-\Delta - k^2)u = 0, \tag{4.9}$$

whereas the vector wave equation is given by

$$\nabla \times \left(\frac{1}{\mu_r}\nabla \times \mathbf{E}\right) - k_0^2 \varepsilon_r \mathbf{E} = 0. \tag{4.10}$$

Using the vector identity

$$\nabla \times \nabla \times \mathbf{E} = \nabla\left(\nabla \cdot \mathbf{E}\right) - \nabla^2 \mathbf{E}, \tag{4.11}$$

yields

$$\nabla\left(\frac{1}{\mu_r}\nabla \cdot \mathbf{E}\right) - \frac{1}{\mu_r}\nabla^2 \mathbf{E}. \tag{4.12}$$

Since the electric field is divergence free, $\nabla \cdot \mathbf{E} = 0$ and the vector Helmholtz equation is derived:

$$-\frac{1}{\mu_r}\nabla^2 \mathbf{E} - k_0^2 \varepsilon_r \mathbf{E} = 0. \tag{4.13}$$

It was shown by Erlangga et al. (Ref. 12) that the shifted Laplace operator preconditioner is an efficient preconditioner for the (scalar) Helmholtz equation (4.9), the shifted Laplace operator preconditioner in vector form is used as preconditioner for the vector wave equation. The shifted Laplace operator preconditioner in this case is given by

$$\mathcal{W}_{\beta_1,\beta_2} = -\frac{1}{\mu_r}\nabla^2 \mathbf{E} - \hat{k}_0^2 \varepsilon_r \mathbf{E} \tag{4.14}$$

where $\hat{k}_0$ is the shifted wavenumber

$$\hat{k}_0 = (\beta_1 - j\beta_2)k_0. \tag{4.15}$$

The matrix obtained from the finite element discretization of the operator in equation (4.14), denoted by M_2, is used as preconditioner for the system (4.8). It has been shown in Erlangga et al. (Ref. 12) that the choice of parameters (β_1, β_2) strongly influences the convergence behavior of the solution method applied to the *second preconditioner system* $M_2\, y = z$. Choosing $(\beta_1, \beta_2) = (1, 0.5)$ yields a fast converging iterative method for the Helmholtz equation, if this system is solved using a multigrid method. For the problem at hand here, a multigrid method

would be the method of choice to solve $M_2\, y = z$ also, but the limited time available precludes the implementation of a multigrid algorithm. Therefore, the system will be solved by the GCR method. Turkel et al. (Ref. 17) applied BiCGSTAB to solve the preconditioner system for the the Helmholtz equation. They applied 15 to 20 inner iterations and it yielded a fast converging method. This is also the case for the vector wave equation where 30 to 40 inner iterations are applied but this problem seems to be more involved, since there is also a middle loop which increases the amount of work. To optimize the values of the couple $(\beta_1,\ \beta_2)$, a numerical experiment will be performed in the next chapter.

4.2.3 A solving chart

Using the two preconditioners as described in the last two sections implies that a nested algorithm with three layers is obtained. In the first layer (also called the outer iteration) the approximation x^k of the exact solution x is calculated. The second layer solves the first preconditioner system $A_{11}s_1^k = r_1^{k-1} - A_{12}s_2^k$ and returns an approximation s^j to the search direction s_1^k. In the third layer the solution to the second preconditioner system $M_2\tilde{s}^j = \tilde{r}^{j-1}$ is approximated. Since this nested algorithm is quite complicated, a solve chart is included below (Algorithm 4.3). In this solve chart several calls to the subalgorithm *orthonormalize* are made. This is the algorithm for the construction of an orthonormal basis for the Krylov subspace and stated in Algorithm 4.4

4.2.4 A combination of the triangular- and the shifted Laplace preconditioner

In the last subsection the current nested preconditioned GCR algorithm was given. A triangular preconditioner M_1 is constructed from the blocks of the finite element discretization matrix A. Then a second preconditioner is constructed to improve the convergence of the iterative method to solve the sparse matrix A_{11}. The new idea in this section is to combine the triangular preconditioner with the shifted Laplace operator preconditioner. This preconditioner looks like

$$M_{new} = \begin{pmatrix} M_2 & A_{12} \\ 0 & A_{22} \end{pmatrix}, \tag{4.16}$$

where M_2 is the finite element discretization of the shifted Laplace operator (4.14). The new preconditioner is constructed in such a way that the (former) middle-loop is not executed anymore and a tremendous reduction in CPU time is expected. The preconditioned system to be solved every outer iteration reads

$$M_{new}s^k = r^{k-1}. \tag{4.17}$$

Again (see Section 4.2.1) this is done in two steps.

- Solve $A_{22}s_2^k = r_2^{k-1}$ with a precomputed LU-decomposition of A_{22}.

Algorithm 4.3 Nested Preconditioned Generalized Conjugate Residual method

in: system matrix A, right-hand side b, preconditioner M_2
out: solution x

Start GCR-1 for solving $Ax = b$.
Compute initial residual $r^0 = b - Ax^0$ for some initial guess x^0.
for $k = 1, 2, \ldots, \max k$ **do**
  Solve $A_{22}s_2^k = r_2^{k-1}$ using a precomputed LU-decomposition for A_{22} :
    Solve $L_{22}w = r_2^{k-1}$ (forward substitution)
    Solve $U_{22}s_2^k = w$ (backward substitution)
  Start GCR-2 for solving $A_{11}s_1^k = r_1^{k-1} - A_{12}s_2^k$
  Compute initial residual $\tilde{r}^0 = (r_1^{k-1} - A_{12}s_2^k) - A_{11}\tilde{s}^0$ for initial guess $\tilde{s}^0$.
  for $j = 1, 2, \ldots, \max j$ **do**
    Start GCR-3 for solving $M_2\tilde{s}^j = \tilde{r}^{j-1}$
    Compute initial residual $\bar{r}^0 = \tilde{r}^{j-1} - M_2\bar{s}^0$ for initial guess $\bar{s}^0$.
    for $i = 1, 2, \ldots, \max i$ **do**
      $\bar{s}^i = \bar{r}^{i-1}$
      $\bar{v}^i = M_2\bar{s}^i$
      call $orthonormalize(i, \bar{v}^1, \ldots, \bar{v}^i, \bar{s}^1, \ldots, \bar{s}^i)$
      $w^i = w^{i-1} + (\bar{v}^i, \bar{r}^{i-1})\bar{s}^i$
      $\bar{r}^i = \bar{r}^{i-1} - (\bar{v}^i, \bar{r}^{i-1})\bar{v}^i$
    end for
    $\tilde{s}^j = w^i$
    $\tilde{v}^j = A_{11}\tilde{s}^j$
    call $orthonormalize(j, \tilde{v}^1, \ldots, \tilde{v}^j, \tilde{s}^1, \ldots, \tilde{s}^j)$
    $y^j = y^{j-1} + (\tilde{v}^j, \tilde{r}^{j-1})\tilde{s}^j$
    $\tilde{r}^j = \tilde{r}^{j-1} - (\tilde{v}^j, \tilde{r}^{j-1})\tilde{v}^j$
  end for
  $s_1^k = y^j$
  $s^k = (s_1^k, s_2^k)^\top$
  $v^k = As^k$
  call $orthonormalize(k, v^1, \ldots, v^k, s^1, \ldots, s^k)$
  $x^k = x^{k-1} + (v^k, r^{k-1})s^k$
  $r^k = r^{k-1} - (v^k, r^{k-1})v^k$
end for

- Solve $M_2\, s_1^k = r_1^{k-1} - A_{12}\, s_2^k$ using GCR (perhaps with a simple ILU(0) preconditioner).

The algorithm using this preconditioner is a new version of Algorithm 4.3. It is stated in Algorithm 4.5.

4.2.5 Start vectors and termination criteria

Algorithms 4.3 and 4.5 are nested iterative algorithms. Hence, a number of startvectors has to be defined. The first startvector is the initial guess to start the outer loop. It is the startvector for the

Algorithm 4.4 Orthonormalization process for the Krylov basis

in: $k, v_1, \ldots, v_k, s_1, \ldots, s_k$
out: s_k, v_k, with $||s^k||_2 = 1$, $||v^k||_2 = 1$

for $l = 1, \ldots, k-1$ **do**
  $\alpha = (v^l, v^k)$
  $v^k = v^k - \alpha v^l$
  $s^k = s^k - \alpha s^l$
end for
$s^k = s^k / ||v^k||_2$
$v^k = v^k / ||v^k||_2$
$V(:, k) = v^k$
$S(:, k) = s^k$

Algorithm 4.5 Preconditioned Generalized Conjugate Residual method

in: system matrix A, right-hand side b, preconditioner M_2
out: solution x

Start GCR-1 for solving $Ax = b$.
Compute initial residual $r^0 = b - Ax^0$ for some initial guess x^0.
for $k = 1, 2, \ldots, \max k$ **do**
  Solve $A_{22} s_2^k = r_2^{k-1}$ using a precomputed LU-decomposition for A_{22} :
    Solve $L_{22} w = r_2^{k-1}$ (forward substitution)
    Solve $U_{22} s_2^k = w$ (backward substitution)
  Start GCR-2 for solving $M_2\, s_1^k = r_1^{k-1} - A_{12} s_2^k$
  Compute initial residual $\tilde{r}^0 = (r_1^{k-1} - A_{12} s_2^k) - M_2 \tilde{s}^0$ for initial guess $\tilde{s}^0$.
  for $j = 1, 2, \ldots, \max j$ **do**
    $\tilde{s}^j = \tilde{r}^{j-1}$
    $\tilde{v}^j = A_{11} \tilde{s}^j$
    call $orthonormalize(j, \tilde{v}^1, \ldots, \tilde{v}^j, \tilde{s}^1, \ldots, \tilde{s}^j)$
    $y^j = y^{j-1} + (\tilde{v}^j, \tilde{r}^{j-1}) \tilde{s}^j$
    $\tilde{r}^j = \tilde{r}^{j-1} - (\tilde{v}^j, \tilde{r}^{j-1}) \tilde{v}^j$
  end for
  $s_1^k = y^j$
  $s^k = (s_1^k, s_2^k)^\top$
  $v^k = A s^k$
  call $orthonormalize(k, v^1, \ldots, v^k, s^1, \ldots, s^k)$
  $x^k = x^{k-1} + (v^k, r^{k-1}) s^k$
  $r^k = r^{k-1} - (v^k, r^{k-1}) v^k$
end for

system $Ax = b$. The right-hand side has a special structure given by

$$b = \begin{pmatrix} 0 \\ b_2 \end{pmatrix}, \tag{4.18}$$

which enables one to set the first component x_1 of x to be zero also. Since there is no obvious choice for the second component, this one is also chosen to be zero, hence, $x^0 = 0$.

The second startvector to be defined (for Algorithm 4.3 only) is the initial guess to solve the system

$$A_{11}\, s_1^k = r_1^{k-1} - A_{12}\, s_2^k, \tag{4.19}$$

iteratively. Since the right-hand side changes every outer iteration, it is not useful to use the approximation to s_1^{k-1} as an initial guess for solving the system (4.19). Therefore, the zero vector is chosen as startvector.

Finally, the third startvector is needed as an initial guess to solve the system

$$M_2\, \tilde{s}_j = \tilde{r}^{j-1}. \tag{4.20}$$

The same argument about a changing right-hand side is applicable here. Hence, this startvector is chosen to be zero also.

After the introduction of several startvectors, the termination criteria are considered. The outer iteration is terminated as soon as the relative residual reaches the prescribed tolerance level, that is, the method terminates when

$$\frac{||r^k||_2}{||b||_2} \leq \varepsilon_o, \tag{4.21}$$

where $\varepsilon_o = 10^{-4}$. This value of ε_o is chosen to limit the total work in the method but to ensure the necessary accuracy of the solution. The middle and inner loops have the same criterion for termination, but in addition, these two loops also have a maximum number of iterations that are allowed to be performed. Hence, such a loop is terminated whenever

$$\frac{||\tilde{r}^j||_2}{||rhs||_2} \leq \varepsilon_j \quad || \quad j = \text{max}_{it}, \tag{4.22}$$

where rhs is the right-hand side corresponding to the residual $\tilde{r}^j$, ε_j is the tolerance for the j-loop and max_{it} is the maximum number of iterations for this loop. The prescribed tolerance in the innerloops is 10^{-1} only, otherwise the total work would increase too drastically.

For the preconditioned GCR method using the new preconditioner, the innerloop tolerance remains $\varepsilon_j = 10^{-1}$. Solving this system more accurate will not decrease the number of outer iterations since the shifted Laplace preconditioner is solved and the corresponding search direction is only an approximation of the search direction for the sparse part A_{11} of the matrix A.

4.3 Work comparison for the unpreconditioned and nested preconditioned GCR method

The goal using an iterative method instead of a direct method is to have a faster method which requires less computer memory to compute the solution of the system $Ax = b$. Since in this case the matrix A is ill-conditioned, convergence of the iterative method is expected to be slow and two preconditioners were introduced to improve the convergence behavior. That is, the number of outer iterations is supposed to decrease when the preconditioners are used. But including these preconditioners yields a nested iterative method, so the question is whether the total work involved in applying the iterative method is really decreased compared to the unpreconditioned GCR method applied to the system $Ax = b$. Therefore, the total work involved is computed for both the unpreconditioned case, the case with both the block and the shifted Laplace preconditioner and finally, the total work involved if the new preconditioner, as introduced in Section 4.2.4, is used.

4.3.1 Total work for unpreconditioned GCR

If the unpreconditioned GCR method is applied to solve $Ax = b$, the total work involved in this method ($W_z(k)$) equals the number of iterations k multiplied by the amount of work per iteration $W(\text{iteration})$. Then the total work to orthogonalize the basis vectors of the Krylov subspace in these k iterations is computed and added. The amount of work per iteration is dominated by the matrix vector product Ay. Since the matrix vector product Ay can be separated in a product with the sparsely and fully populated part, the total work per iteration is computed as follows:

$$W(\text{iteration}) = W(A_{11}\, y_1) + W(A_{22}\, y_2). \tag{4.23}$$

Since the block A_{11} is sparse, the work to compute $A_{11}\, y_1$ equals $6\tilde{c}(n - f)$ where n is the total number of degrees of freedom $N\text{dof}$, f is the number of degrees of freedom on the aperture $N\text{dof}_{ap}$ and $\tilde{c}$ is the average number of nonzeros per row for the sparse block A_{11} (see Section 3.3 and Appendix B). Since the block A_{22} is complex valued, the work to compute $A_{22}\, y_2$ equals $8f^2$. This number is justified by considering that per row, f products of two complex valued numbers have to be taken as well as $f - 1$ complex additions. Hence, the total work involved in one iteration equals

$$6\tilde{c}(n - f) + 8f^2. \tag{4.24}$$

Hence, the total work performed after k iterations is

$$W_z(k) = k[6\tilde{c}(n - f) + 8f^2]. \tag{4.25}$$

In the end the work involved in the orthogonalization during this k iterations is computed:

$$W(\text{orthogonalize}, k) = \sum_{j=1}^{k-1} j \Big[W(\text{complex i.p.}) + 2W(\text{complex scalar} \times \text{ vector}) + 2W(\text{vector update}) \Big] \quad (4.26)$$

The work for these three operations is summarized below

$$\begin{aligned} W(\text{complex i.p.}) &= n \text{ complex valued products} + (n-1) \text{ additions} \approx 8n, \\ W(\text{complex scalar} \times \text{ vector}) &= n \text{ complex valued products} = 7n, \\ W(\text{vector update}) &= n \text{ additions} = n, \end{aligned}$$

and this yields

$$W(\text{orthogonalize}, k) = \sum_{j=1}^{k-1} j \Big[8n + 2(7n) + 2n \Big] = 24n \sum_{j=1}^{k-1} j \approx 12k^2 n. \quad (4.27)$$

The total work is therefore,

$$W_z(k) = k[6\tilde{c}(n-f) + 8f^2] + 12k^2 n. \quad (4.28)$$

The value of $\tilde{c}$ was already calculated in Hooghiemstra (Ref. 3) (section 6.2.4) and is given as a function of the element order

$$\tilde{c} = (19 + 18p)(p+1). \quad (4.29)$$

Hence, the total work involved in the unpreconditioned GCR method is equal to

$$W_z(k) = k[6(19 + 18p)(p+1)(n-f) + 8f^2] + 12k^2 n. \quad (4.30)$$

4.3.2 Total work for nested GCR

In case of the application of the nested GCR method, the computation is more involved. Now, the total work ($W_m(i,j,k)$) is calculated as the product of the total number of (outer) iterations k with the sum of the work per outer iteration (C_2) and the work involved in the preconditioner solves ($m_1(i,j)$), whereafter the orthogonalization costs are added again. Hence

$$W_m(i,j,k) = k[C_2 + m_1(i,j)] + W(\text{orthogonalize}, k). \quad (4.31)$$

The work per outer iteration exists of a forward and backward substitution with the fully populated block (the LU-solve) and one matrix vector product Ay. Since the forward and backward substitution cost $4f^2$ each (see Appendix B for details)

$$C_2 = 6\tilde{c}(n-f) + 8f^2 + 8f^2. \tag{4.32}$$

The preconditioner solves exist of solving the system $A_{11}\, s_1 = r_1 - A_{12}\, s_2$ iteratively. The work involved in this operation is

$$m_1(i,j) = j[W(A_{11}y) + m_2(i)] + W(\text{orthogonalize}, j), \tag{4.33}$$

where $m_2(i)$ accounts for the work in the most inner loop. Hence the costs for performing $m_1(i,j)$ are

$$m_1(i,j) = j[6\tilde{c}(n-f) + m_2(i)] + 12j^2(n-f). \tag{4.34}$$

The work involved in the most innerloop $m_2(i)$ equals

$$m_2(i) = i\, W(M_2 y) + W(\text{orthogonalize}, i). \tag{4.35}$$

Since the matrix M_2 is complex valued, due to the complex shift (see Section 4.2.2), the work to compute the matrix vector product equals (see Appendix B for details)

$$W(M_2y) = 10\tilde{c}(n-f) \quad \Longrightarrow \quad m_2(i) = i\,10\tilde{c}(n-f) + 12i^2(n-f)]. \tag{4.36}$$

Hence, the total work for the nested GCR method equals

$$W_m(i,j,k) = k\Big[6\tilde{c}(n-f) + 16f^2 + j\Big\{6\tilde{c}(n-f) + 10i\tilde{c}(n-f) + 12i^2(n-f)\Big\} + 12j^2(n-f)\Big] + 12k^2\, n. \tag{4.37}$$

4.3.3 Total work using the new preconditioner

In exactly the same way as in the last section, the amount of work for the GCR method using the new preconditioner is computed. The j-loop vanishes and hence, the amount of work equals

$$W_{new}(i,k) = k\big[6\tilde{c}(n-f) + 16f^2 + i\,10\tilde{c}(n-f) + 12i^2(n-f)\big] + 12k^2\, n. \tag{4.38}$$

4.3.4 Amount of work: conclusion

From equations (4.30), (4.37) and (4.38), it is clear that the amount of work involved in the nested preconditioned GCR method with two preconditioners is more than for the preconditioned GCR method with the adjusted preconditioner M_{new}. The latter method performs fewer work since it does not need to execute the middle-loop anymore. A reduction in work of at least a factor 10 may be expected as will be shown in Chapter 5.

The comparison between the preconditioned GCR method and the unpreconditioned GCR method is more involved since the number of outer iterations k and k_z respectively, are not known on before hand. A strong reduction in the number of outer iterations is required for the preconditioned GCR method to obtain a method that requires less work than the unpreconditioned GCR method. The memory requirements however, are nearly always in favor of the preconditioned GCR method.

4.4 Vectorization of the implementation

The NLR uses a NEC-SX-8R, 8 processor shared memory vector machine as their main computing platform. Therefore, the implementation of the iterative algorithm has to be tailored for vectorization of the operations. In the subsequent sections, several adjustments to the algorithm are made for this purpose.

4.4.1 Optimization step 1 for the product $M_2\bar{s}^i$

The product $M_2\bar{s}^i$ is performed *matrix free*, which means that the matrix is not assembled from the element matrices, whereafter the product with the vector $\bar{s}^i$ is taken, but that this multipication is done using the element matrices and the mapping which maps the local degrees of freedom to the corresponding global ones. In general there are two benefits of this approach:

- The complete matrix needs not to be stored. For large sparse matrices this means a huge saving in memory.
- Direct access to the element matrices which are fully populated preserves multiplication by zeros.

The algorithm for performing the matrix free multiplication is given in algorithm C.1 in Appendix C.

4.4.2 Speed of vectorization

The vectorization speed of a problem depends heavily on the lengths of the vectors. For the SX-8R computer used here at the NLR, the optimal vector length is 256. Roughly speaking, if all vectors have this length, vectorization is optimal and the calculations are performed very fast. But when even one vector has a smaller size, much of the vectorization speed is lost and the

performance will be much lower. In the present implementation of the matrix free matrix vector product (Algorithm C.1) the vectorization runs over vectors with length *nlocbas*. This number depends on the element order p. For $p = 0$ and $p = 1$, the vectorlength is too small, in which case it is not favourable to work on a vectorcomputer. But even for second order elements, $nlocbas = 57$ only, not even close to the ideal vector length 256. This is immediately observed if a close look on the performance output is taken. The performance using an average vectorlength of 60 is only 2.0 Gflops, which is only 10 percent of the peak performance of a single processor.

4.4.3 Optimization step 2 for the product $M_2\bar{s}^i$

To overcome the problem of short vectors, the loops within the algorithm are switched. The innerloop will be a loop over the elements instead of the entries of the mapping. For large problems this will give a vector length that is at least 256. If there are more than 256 elements, the calculations are performed in batches of exactly 256.

However, switching these loops may cause the same problems mentioned before: The simultaneous updating of one vector position with two values. This problem only occurs if two elements sharing an edge are treated at the same time. This problem is solved using a so-called *coloring* algorithm for the elements. In this algorithm (which can be found in Appendix C in Algorithm C.2 and C.3), the elements are divided in colors in such a way that elements in one color do not share a single edge with each other. The algorithm to divide these elements runs through the elements twice. In the first *swap* the elements are divided in colors, but then not every color has the same amount of elements. The second swap divides the elements in a fair way over all colors. Assuming that every color has an ideal number of elements (=256), the matrix free product is taken per color. The performance of the method using longer vectors results in a computation speed of nearly 8.0 Gflops, which is over half the peak performance of a single processor.

5 Optimization of the nested, preconditioned GCR method: Fine tuning the parameters

5.1 Introduction

In this chapter the performance of the nested, preconditioned GCR method is investigated. To find relations between for example, the amount of work involved in the method and the number of outer iterations, various numerical experiments are performed. In first instance the baseline method as introduced in Section 4.2,is used to analyse the relation between the number of degrees of freedom (Ndof) and the number of outer iterations (It_k) and CPU time. The baseline method is the nested preconditioned GCR method where all important parameters such as the number of middle (It_j) and inner iterations (It_i) and the couple ($\beta_1,\ \beta_2$) are fixed to a standard value. The shift-parameters ($\beta_1,\ \beta_2$) of the shifted Laplace operator preconditioner are chosen equal to the ideal couple found by Erlangga et al. (Ref. 12). Hence, ($\beta_1,\ \beta_2$) $= (1.0, 0.5)$. The number of middle and inner iterations is fixed on the value $(It_j, It_i) = (10, 5)$.

Secondly, the nested, preconditioned baseline GCR method will be compared to the unpreconditioned GCR method. Therefore an experiment is done to obtain a better insight in the quality of the preconditioners. The theory concerning the amount of work in both methods as described in Section 4.3 is applied to a model problem to see the differences in work (and memory requirements) between the two methods.

Thereafter a comparison between the frontal solver and the baseline GCR method is made. A third numerical experiment is performed to show that the amount of work and CPU time for the frontal solver depends both on the size of the aperture and the total number of unknowns, whereas the CPU time of full GCR depends solely on the total number of degrees of freedom. Therefore, a series of rectangular cavities with increasing aperture area and constant cavity volume will be considered in this experiment.

Using the baseline method, the memory requirements for the method are very high because all the Krylov subspace basisvectors have to be stored. Various techniques to decrease the amount of memory needed in the iterative solver will be discussed, amongst others, truncated GCR and restarted GCR.

A third method to relax the memory requirements is to increase the number of middle and inner iterations (It_j, It_i). The sensitivity of the method with respect to (It_j, It_i) is tested next. To optimize the shift-parameters ($\beta_1,\ \beta_2$) heuristically, since there is not enough time to do a thorough mathematical analysis, several experiments are done with different values of these parameters.

Once all important parameters $(\beta_1,\ \beta_2,\ It_j,\ It_i)$ are set to their heuristic values, the effect on the CPU time and the number of outer iterations is investigated for the optimized GCR method. Hence, two more tests are performed. The first test considers the relation between Ndof and It_k The second test considers the relation between Ndof and the corresponding CPU time. Based on these two trends a prediction will be given for the CPU time for the target problem.

Finally, the preconditioned GCR method using the coupled preconditioner as described in Section 4.2.4 is considered. A trend in the CPU time as a function of the number of degrees of freedom is being sought and the performance of this method is compared to the frontal solver.

5.2 Nested preconditioned GCR: the baseline method

The performance of the baseline method with fixed parameters is tested for a series of rectangular cavities with increasing depth L. The parameters in the nested preconditioned GCR method are given by $(It_j, It_i) = (10, 5)$ and $(\beta_1,\ \beta_2) = (1.0, 0.5)$. The number of outer iterations and the corresponding CPU time are given in Table 5.1.

L	L/d	N_{el}	Ndof	It_k	CPU (hh:mm:ss)	memory (Gb)
1	0.67	1111	19614	2298	00:27:27	1.61
2	1.33	2268	39840	3293	01:08:31	4.78
4	2.67	4580	80283	5001	04:13:08	13.49
8	5.33	9175	160599	10001	18:10:14	52.67

Table 5.1 *The number of elements N_{el}, the number of degrees of freedom Ndof, the number of (outer) iterations performed to reach the tolerance $\varepsilon = 10^{-4}$ and the CPU time for cavities with various (increasing) depth L discretized with a meshwidth $h = 0.25$. All these experiments have been performed for the baseline method: $(It_j, It_i) = (10, 5)$ and $(\beta_1,\ \beta_2) = (1.0, 0.5)$.*

According to Table 5.1, the trend in the number of outer iterations as a function of Ndof seems to be linear. This is illustrated in Figure 5.1.

Referring to Table 5.1 once more, the CPU time increases very fast if the number of degrees of freedom is doubled. This increment in CPU time has two reasons.

- Every iteration the Krylov subspace vectors are orthogonalized. For an increasing number of outer iterations, the amount of work per iteration increases significantly.
- Since the number of degrees of freedom doubles, the time to perform a matrix free matrix vector product doubles as well.

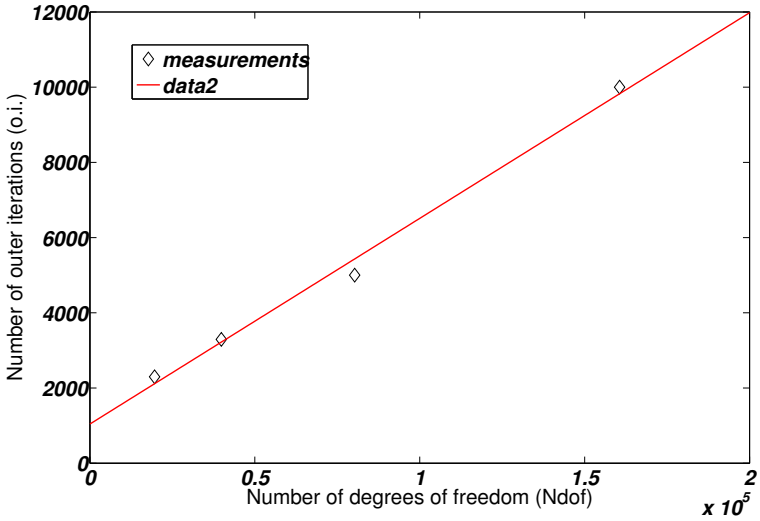

Fig. 5.1 An extrapolation of the data given in Table 5.1 for the number of outer iterations of the nested, preconditioned GCR method as a function of the number of degrees of freedom (Ndof) in the system. The corresponding domains are rectangular cavities with dimensions $1.5 \times 1.5 \times L$, *where* $L \in \{1, 2, 4, 8\}$. *The number of inner iterations is fixed:* $(It_j, It_i) = (10, 5)$ *and* $(\beta_1,\ \beta_2) = (1.0, 0.5)$.

These two reasons result roughly in a quadratic CPU time trend (Figure 5.2). If the number of degrees of freedom is doubled, the CPU time doubles twice. Hence

$$\text{CPU time (min)} \propto c_1\, N\text{dof}^2, \tag{5.1}$$

where c_1 is independent of Ndof.

5.2.1 Nested preconditioned GCR vs. unpreconditioned GCR

Since the introduction of the two preconditioners there is not very much attention given to the unpreconditioned GCR method. According to the theory in Section 4.3, the amount of iterations performed for a model problem using the unpreconditioned GCR, denoted by k_z, has to be much greater than the corresponding number of outer iterations using the nested, preconditioned GCR method, denoted by k, in order to have a smaller CPU time for the nested preconditioned GCR method than for unpreconditioned GCR. In the following this will be analysed for a model problem.

The linear system to be solved for this model problem is the matrix obtained from the finite element discretization of the vector wave equation on a rectangular domain with dimensions $1.5 \times 1.5 \times 2.0$ discretized using a meshwidth $h = 0.30$. The mesh is illustrated in Figure

A.7 in Appendix A. The parameter for the nested preconditioned GCR method are chosen to be the same as in the previous experiments, that is, $(It_j, It_i, \beta_1, \beta_2) = (10, 5, 1.0, 0.5)$. The convergence behavior of both methods is illustrated in Figure 5.3. This figure suggests that the quality of the preconditioners is very good, but the CPU time is roughly 3 times longer for the nested preconditioned GCR than it is for unpreconditioned GCR.

5.2.2 Nested preconditioned GCR vs. the frontal solver

As already mentioned in the introduction of this chapter, it has been shown (see van der Heul et al. (Ref. 9) and Jin et al. (Ref. 18)) that the computational time for the frontal solver depends strongly on the number of degrees of freedom on the aperture $N\text{dof}_{ap}$, while the convergence of the nested, preconditioned GCR depends on the total number of degrees of freedom $N\text{dof}$. Hence, if a series of cavities is analysed with increasing aperture area and constant volume, the CPU time for the frontal solver is expected to increase rapidly, but the CPU time for the nested preconditioned GCR method is expected to be approximately constant. Hence, a numerical experiment is performed to investigate if this is really the case. Table 5.2 summarizes the result. The meshes corresponding to the problems in these experiments are shown in Appendix A, Figures A.3 - A.6.

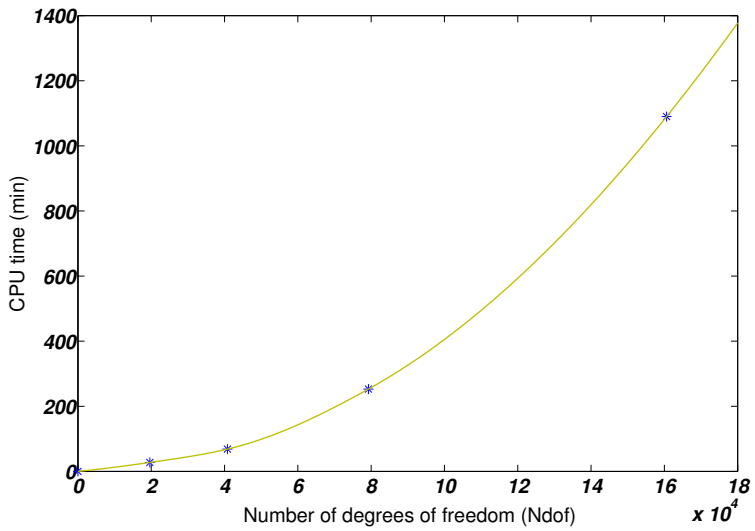

Fig. 5.2 An extrapolation of the data given in Table 5.1 for the CPU time (min) of the nested, preconditioned GCR method as a function of the number of degrees of freedom (Ndof) in the system. The corresponding domains are rectangular cavities with dimensions $1.5 \times 1.5 \times L$, *where* $L \in \{1, 2, 4, 8\}$. *The number of inner iterations is fixed:* $(It_j, It_i) = (10, 5)$ *and* $(\beta_1, \beta_2) = (1.0, 0.5)$.

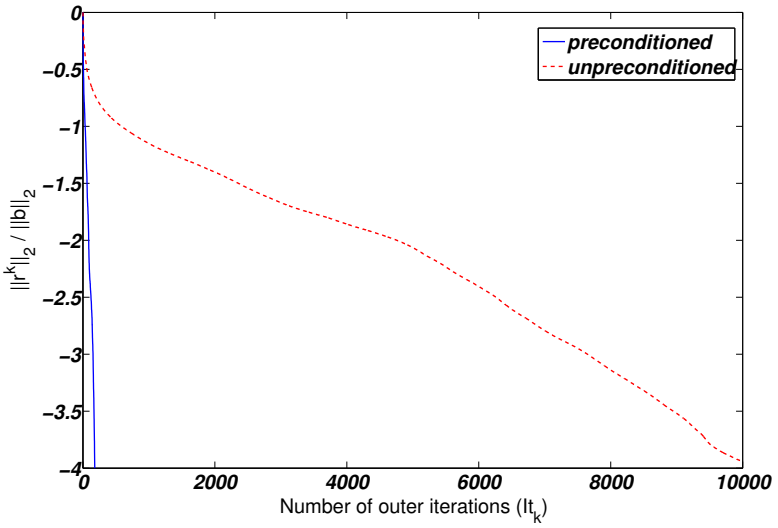

Fig. 5.3 The preconditioned GCR method versus the unpreconditioned GCR method for a model problem of dimensions $1.5 \times 1.5 \times 2.0$, $h = 0.30$. *The number of degrees of freedom* $N\mathrm{dof} = 22407$.

size cavity	Ndof	$N\mathrm{dof}_{ap}$	CPU time GCR	CPU time Frontal
$1.0 \times 1.0 \times 8.0$	66948	312	02:52:13	00:00:32
$2.0 \times 1.0 \times 4.0$	69750	636	03:10:12	00:01:33
$4.0 \times 1.0 \times 2.0$	70140	1284	03:37:44	00:04:36
$8.0 \times 1.0 \times 1.0$	69789	2580	03:31:25	00:16:48

Table 5.2 *The number of degrees of freedom on both the aperture as inside the cavity and the corresponding CPU times (hh:mm:ss) for the iterative method and the frontal solver. The corresponding meshes are depicted in Appendix A in Figures A.3 - A.6.*

The results shown in Table 5.2 are clearly in favour of the frontal solver[1]. But if an experiment is performed with a larger aperture area, the computation time for the frontal solver explodes as illustrated by the trend in Figure 5.4. This will be explained below.

Although the frontal solver is a state-of-the-art direct solver, this method is still based on the Gaussian elimination process. The amount of work to be done with Gaussian elimination scales with the cube of the matrix size (see Vuik et al. (Ref. 16), p.27). For the frontal solver, the amount of work seems to scale with the cube of the number of degrees of freedom on the aperture $N\mathrm{dof}_{ap}$. This means that for very large linear systems corresponding to a rectangular cavity with many unknowns on the aperture ($N\mathrm{dof}_{ap} \geq 25000$) the nested preconditioned GCR method is pre-

[1]This experiment will be done in Section 5.5.1 also, where the preconditioner M_{new} will be used.

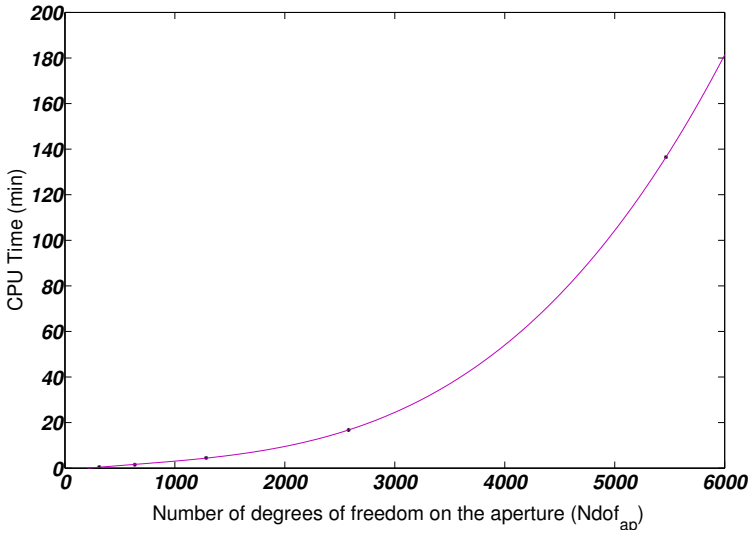

Fig. 5.4 A cubic extrapolation of the data given in Table 5.2, for the CPU time (min) of the frontal solver as a function of the number of degrees of freedom on the aperture ($N\text{dof}_{ap}$). The corresponding domains have a constant number of degrees of freedom $N\text{dof}$, but $N\text{dof}_{ap}$ increases.

ferred over the frontal solver.

Thereby, an exact solution as computed by the frontal solver is redundant, because of the presence of the discretization error introduced during the finite element discretization. Therefore, the iterative method might perform a whole lot better if the tolerance is adjusted from $\varepsilon_o = 10^{-4}$ to $\varepsilon_o = 10^{-2}$ for example.

5.2.3 The baseline algorithm revised

The last sections showed the performance of the baseline nested preconditioned GCR method for a series of problems. Several trends concerning CPU time, and number of outer iterations have been pointed out and from this it seems that the baseline nested preconditioned GCR is a promising method. But, the amount of memory needed for the storage of the Krylov basisvectors increases linearly with the number of outer iterations. This means that the memory requirements for this method increase drastically if the number of outer iterations is not bounded *a priori*.

Suppose a problem with $N\text{dof} = 2.0 \cdot 10^5$ is considered and the maximum number of iterations $\max_{it} = 10000$ is set[2]. Storing the orthonormal vectors would cost

$$2 \cdot N\text{dof} \cdot \max_{it} \cdot 16\text{b} \approx 64\,\text{Gb}. \tag{5.2}$$

[2]This number is justified by considering Table 5.1 once more.

If the mesh is refined which implies a greater number of degrees of freedom, this storage becomes unacceptably expensive, since the SX-8R computer has *only* 128 Gb.

Therefore, it is necessary that some adjustments are made to the algorithm in order to relax the memory requirements. In the following section several methods to do this are described and tested in order to investigate their quality. In general it is possible to reduce this storage demand in three ways.

1. Apply a so-called *truncation* method. Instead of storing the complete orthonormal basis, only several (e.g. 100 or 500) basis vectors are stored. Then the memory requirements are relaxed, but information of previous basis vectors is lost. Hence, the influence of truncation on the convergence behavior of the method should be investigated. Three different truncation methods will be described and applied to a small model problem in the next subsection.
2. Apply a restart. A restart parameter *res* is defined, and after every *res* outer iterations, the approximation x^{res} is used as input for the next *res* iterations. Again, the influence on the rate of convergence of the nested, preconditioned GCR method has to be investigated.
3. Apply far more inner iterations, for example, choose $(It_j, i.i) = (15, 30)$ instead of $(10, 5)$. Although the number of outer iterations will decrease, the total CPU time per iteration increases.

5.3 Adjustments to the nested preconditioned GCR method

In this section a number of methods is described concerning the problem of the high memory requirements which were obtained using the baseline method. For example, truncation, restarting and increasing the number of inner iterations will be treated. Although the number of inner iterations can be reduced greatly by applying a multigrid method to solve the second preconditioned system, this is not treated here by lack of time. Therefore, the shift-paramaters (β_1, β_2) are altered and the effect on the average number of inner iterations, the number of outer iterations and CPU time will be investigated.

5.3.1 Truncation

A truncation method uses a truncation parameter *trunc*. Instead of storing all Krylov basis vectors, only *trunc* basis vectors are stored. A huge saving in storage requirements can be reached if the variable *trunc* is chosen small enough. For example, if $trunc = 500$, the storage requirements decrease with a factor 20 compared to the former example where $\max_{it} = 10000$. The three truncation methods to be described next differ from each other in the choice of basis vectors to be stored.

The first truncation method is very simple, since only the last *trunc* basis vectors are stored. This means that the Krylov basisvectors are completely renewed every *trunc* iterations. If the first *trunc* basisvectors yield an approximation of a Ritz value corresponding to a convergence disturbing (bad) eigenvalue and these basisvectors span the eigenvalue corresponding to this eigenvalue, superlinear convergence is established. If in the following iterations some of these basisvectors are lost, the remaining basisvectors do not span this eigenvector completely anymore losing the superlinear convergence behavior.

A second drawback of this method is that the same *bad* eigenvalues are found repeatedly, hence some of the bad eigenvalues will never be found leading to a slow convergence or even stagnation of the method (see also (Ref. 19)). The orthogonalization loop as given in Algorithm 4.4 will change slightly into Algorithm C.4 in Appendix C.

The second truncation method uses the first $trunc - 1$ basis vectors and the last one every iteration. This is done in order to hold the first several Ritzvalues (approximations of the *bad* eigenvalues) in memory to maintain superlinear convergence. The last vector is added to ensure that the new vector is orthogonal. This truncation method also has a drawback since later *bad* eigenvalues are neglected. Algorithm C.5 in Appendix C gives the algorithm for this truncation method.

The last truncation method is called the *minalfa* method. The basic idea is to remove the basis vector for which

$$\alpha = (v^i, v^j) \tag{5.3}$$

is minimal. Since the new vector is updated using this α, the update does not make much sense for small α and is omitted. Algorithm C.6 in Appendix C gives the algorithm for this truncation method.

The three truncation methods described above have been implemented in Matlab and have been tested for a model problem. The corresponding grid is depicted in Appendix A in Figure A.2. For this model problem second order basis functions are used and a meshwidth $h = 0.30$. This yields a discretization matrix A with $N\text{dof} = 5976$. Figure 5.5 illustates the convergence behavior for all three truncation methods. From this figure it is immediately clear that the convergence of the nested, preconditioned GCR method is very poor if truncation is applied. For this figure, only 137 basis vectors have been neglected resulting in slow convergence or even stagnation of the process. Clearly, there are many *bad* eigenvalues and it seems not a good idea to use truncation for a problem involving a matrix with more degrees of freedom.

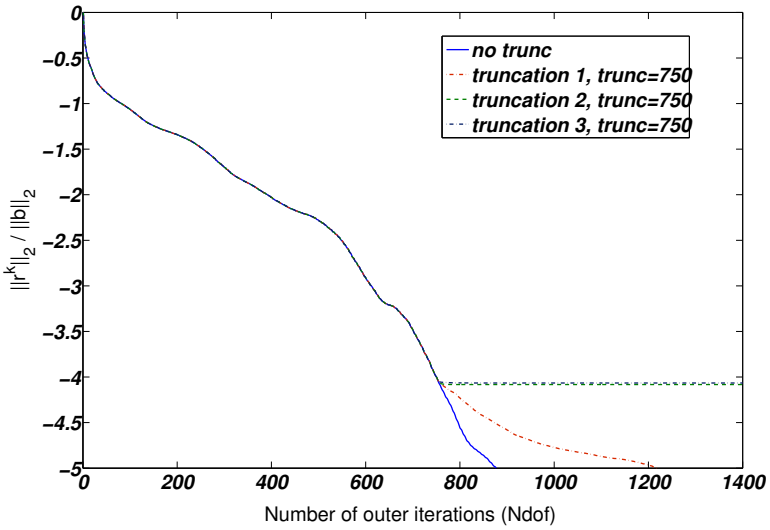

Fig. 5.5 The convergence behavior of the nested, preconditioned GCR method using the three truncation methods. If no truncation is applied the number of outer iterations equals 887. *The variable* $trunc = 750$. *The couple* $(\beta_1, \beta_2) = (1.0, 0.5)$ *and* $(It_j, It_i) = (10, 5)$. *The number of degrees of freedom is* $N\text{dof} = 5976$ *for a cavity of dimensions* $1.5 \times 1.5 \times 0.6$.

5.3.2 Restarted GCR

Another method to decrease the amount of memory is so-called *restarted GCR*. In this method a restart parameter res is defined and every res iterations the method starts over again using the approximation x^{res} as initial guess for the following run. Choosing the right restart parameter is quite difficult since choosing it too big may not lead to a decrease in memory, while choosing the restart parameter too small may not lead to a converging iterative method. Restarted GCR is tested for the same model problem as used for the truncation tests above. The restart value also equals the truncation parameter *trunc*, hence $res = 750$. Figure 5.6 illustrates the convergence behavior for this test. This figure shows, that even restarting with a restart parameter so close to the number of iterations without restart, yields a very slow converging method. It turns out that it takes a large number of iterations before the superlinear convergence is established (at $It_k \approx 500$ here) and restarting destroys this nice behavior.

5.3.3 More inner iterations

In the last two sections it is made clear that although truncation and restarting the GCR method yields only a small amount of memory in theory, the nice convergence behavior is completely lost. Therefore, the third method to decrease the amount of memory is analysed.

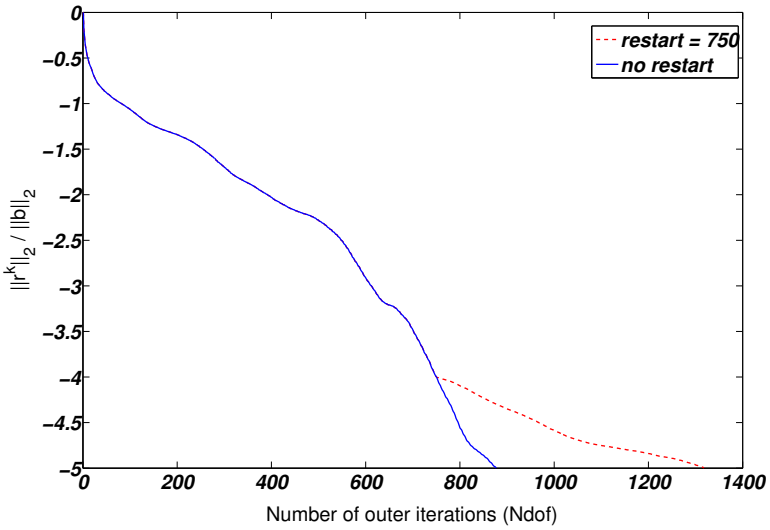

Fig. 5.6 The convergence behavior of the nested, preconditioned GCR method with restart. The restart variable is $res = 750$. *The couple* $(\beta_1, \beta_2) = (1.0, 0.5)$ *and* $(It_j, It_i) = (10, 5)$. *The number of degrees of freedom is* $Ndof = 5976$ *for a cavity of dimensions* $1.5 \times 1.5 \times 0.6$.

Until now only a small amount of inner iterations were applied in the experiments. For the measurements in Figure 5.2 and Table 5.1, for example, the couple $(It_j, It_i) = (10, 5)$ was used. Using more inner iterations cause the iterative method to converge faster since the search direction s_1^k is approximated with a higher accuracy. Hence, the number of outer iterations (It_k) decreases. This seems to be an excellent method to decrease the amount of memory needed to run the iterative method because of the small number of outer iterations, but it also has a drawback. If the number of inner iterations is increased, the CPU time per outer iteration will increase also. So on one hand the memory requirements are relaxed using more inner iterations, but on the other hand the CPU time will increase. Hence, a careful choice must be made to obtain an optimal method in both memory requirements as well as CPU time. Of course this also depends heavily on the computer used for the experiments and the available time.

For various couples of middle and inner iterations some experiments have been performed to analyse the sensitivity of the method. Table 5.3 summarizes the result for (again) the same model problem as in the truncation and restarted GCR experiments. This table suggests that the couple $(It_j, It_i) = (10, 10)$ reduces the number of outer iterations with a factor 2.

As already mentioned before, increasing the number of inner iterations cause an increment in the CPU time per outer iteration. The results in Table 5.3 do not give any information about the CPU

It_j	$It_i = 5$	$It_i = 10$	$It_i = 20$	$It_i = 50$
5	1296	928	579	372
10	1001	574	418	253
20	662	414	300	251
50	404	266	261	247

Table 5.3 *The effect on the number of outer iterations It_k of various couples of middle and inner iterations (It_j, It_i), for a problem with Ndof $= 5976$. The corresponding mesh is illustrated in Figure A.2 in Appendix A.*

time involved. Therefore, it is interesting to find out how this CPU time is affected for the various choices of (It_j, It_i). Instead of using the same model problem again, a second model problem with more degrees of freedom is used to analyse the CPU time, since in the end the nested, preconditioned GCR method is supposed to be applied on much larger systems. The domain for this case (which is depicted in Figure A.7 in Appendix A) has dimensions $1.5 \times 1.5 \times 2.0$, where the same meshwidth $h = 0.30$ is used. The number of elements is $N_{el} = 1287$ and the number of degrees of freedom is $N\text{dof} = 22407$. Table 5.4 summarizes the number of outer iterations (It_k) and the CPU time. From this table it seems that the choice $(It_j, It_i) = (20, 10)$ is the most promising in order to decrease the memory by a factor 2 with respect to the old couple $(It_j, It_i) = (10, 5)$ and to have a CPU time increment of nearly a factor 1.5. However, for model problems involving much larger linear systems, the reduction in memory of *only* a factor 2 might not be enough to ensure a bounded memory requirement for the nested, preconditioned GCR method.

	$i.i = 5$		$i.i = 10$		$i.i = 20$		$i.i = 50$	
It_j	It_k	CPU/memory	It_k	CPU/memory	It_k	CPU/memory	It_k	CPU/memory
5	3424	26/2.68	2287	26/1.97	1556	34/1.44	935	52/0.93
10	2302	29/1.97	1547	40.5/1.44	1098	45/1.05	644	69/0.75
20	1591	38/1.44	993	42/0.91	715	57/0.74	434	96/0.58
50	858	49/0.90	553	63/0.75	454	94/0.58	260	140/0.46

Table 5.4 *The effect on the number of outer iterations It_k, CPU time (min) and the amount of memory (Gb) of various couples of middle and inner iterations (It_j, It_i), for a problem with Ndof $= 22407$. The corresponding mesh is illustrated in Figure A.7 in Appendix A.*

5.3.4 A heuristic optimization of the shift-parameters (β_1, β_2)

All the numerical experiments that have been performed in this chapter so far use the same couple of shift-parameters $(\beta_1, \beta_2) = (1.0, 0.5)$. This choice is justified by considering the analogy between the scalar Helmholtz equation and the vector wave equation and the fact that Erlangga et al. (Ref. 12) performed several numerical experiments to show that this choice of the parameters yields an efficient and robust preconditioner for the Helmholtz equation. The idea to use the values $(\beta_1, \beta_2) = (1.0, 0.5)$ is that the finite element discretization of the shifted Laplace preconditioner is very similar to the original finite element discretization matrix. Hence, using this choice of the parameters, the number of outer iterations is expected to be low.

An interesting experiment would be to obtain a relation between the number of outer iterations at one side, and the average number of inner iterations to reach a prescribed (inner) tolerance $\varepsilon_i = 10^{-2}$ in the innerloop for various values of (β_1, β_2), on the other side. Table 5.5 summarizes the number of outer iterations It_k to reach tolerance $\varepsilon_o = 10^{-5}$, given various values of (β_1, β_2) and the corresponding average number of inner iteration $\overline{It_i}$. The used CPU time is also denoted in this table. The model problem that is used here is the same as in the last subsection. Hence, the dimensions of the rectangular cavity are $1.5 \times 1.5 \times 2.0$ and the discretization meshwidth is $h = 0.30$. The number of degrees of freedom equals 22407. The mesh is depicted in Figure A.7 in Appendix A.

According to Table 5.5 the average number of inner iterations $\overline{It_i}$ depends heavily on the complex-shift parameter β_2. While $\overline{It_i}$ increases rapidly as $\beta_2 \rightarrow 0$, the corresponding number of outer iterations It_k decreases slowly. As can be seen in the CPU time column, the amount of work and CPU time performed for a minimum number of outer iterations is very high compared to the amount of work and CPU time for the minimum number of $\overline{It_i}$ (see also Figure 5.7). This is easily explained since for every outer iteration It_k there are approximately $\overline{It_i}$ inner iterations to be performed. The total amount of work in these experiments can roughly be estimated by

$$\text{Work} = \overline{It_i} \cdot It_k \tag{5.4}$$

Using this estimation it is clear that the couple $(\beta_1, \beta_2) = (1.0, 2.50)$ (Work = 23484) results in less work than for example the couple $(\beta_1, \beta_2) = (1.0, 0.50)$ (Work = 49224), see Figure 5.7 again.

Considering Table 5.5 again it is observed that a much larger value for the complex-shift parameter β_2 is obtained here in comparison with the value used by Erlangga[3] et al. (Ref. 12). The

[3]The choice $(\beta_1, \beta_2) = (1.0, 0.5)$ is also strongly affected by the use of a one-cycle multigrid method that is used in their work.

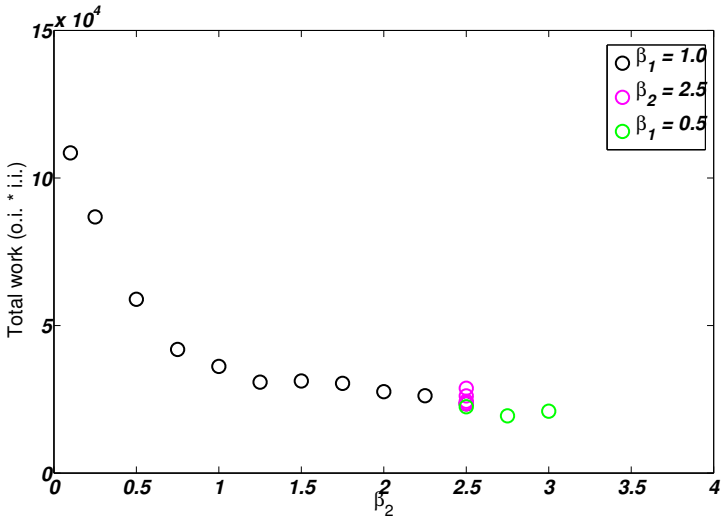

Fig. 5.7 The total work (computed as $It_k \cdot \overline{It_i}$) as a function of the complex-shift parameter β_2 where the real-shift parameter β_1 is fixed. The pink circles correspond to a fixed value for the complex-shift parameter β_2 and β_1 is altered. The model problem here has dimensions $1.5 \times 1.5 \times 2.0$ and $N\,dof = 22407$.

influence of the real-shift parameter β_1 is negligible compared to β_2 but it is still worth the effort to investigate it. It turns out that the optimal value for this parameter is $\beta_1 = 0.5$. The optimal couple $(\beta_1,\ \beta_2)$ according to Table 5.5 is $(\beta_1,\ \beta_2) = (0.5, 3.0)$.

5.3.5 Conclusion

The last four subsections have shown that there are many ways to decrease the amount of memory for the nested preconditioned GCR method. One thing they all have in common is the increase in CPU time compared to the high memory baseline method discussed in Section 5.2. Truncating and restarting the GCR method is not our method of choice since the 'nice' superlinear convergence is immediately lost after truncating or restarting the method and it takes a long time before it establishes again.

Increasing the nummer of inner iterations (It_j, It_i) is our method of choice, since the number of outer iterations can be reduced significantly and hence so do the memory requirements. But the CPU time increases also for this method. In order to limit the CPU time per iteration, the shift parameters $(\beta_1,\ \beta_2)$ have been optimized heuristically and the optimal value of this couple is $(\beta_1,\ \beta_2) = (0.5, 3.0)$. The relation between the number of outer iterations and the number of degrees of freedom as well as the relation between the number of outer iterations and the CPU time will be investigated in the next section, where the optimized GCR method will be used.

$\beta_1,\ \beta_2$	$\overline{It_i}$	It_k	CPU time	$\beta_1,\ \beta_2$	$\overline{It_i}$	It_k	CPU time
$(1.0, 2.50)$	57	412	51	$(-1.0, 2.50)$	57	459	50
$(1.0, 2.25)$	63	416	60	$(-0.5, 2.50)$	53	459	53
$(1.0, 2.00)$	70	394	62	$(-0.5, 2.50)$	53	459	53
$(1.0, 1.75)$	80	380	59.5	$(0.0, 2.50)$	52	553	61
$(1.0, 1.50)$	91	343	63	$(0.75, 2.50)$	55	432	51
$(1.0, 1.25)$	107	288	66	$(1.0, 2.50)$	57	412	52
$(1.0, 1.00)$	139	260	83.5	$(0.5, 2.50)$	53	424	49
$(1.0, 0.75)$	187	224	107	$(0.5, 2.75)$	48	404	45
$(1.0, 0.50)$	293	201	168	$(0.5, 3.00)$	45	466	43
$(1.0, 0.25)$	496	175	226				
$(1.0, 0.10)$	≥ 500	217	304				

Table 5.5 *The number of outer iterations It_k, the average number of inner iterations $\overline{It_i}$ and the CPU time (min) for various values of $(\beta_1,\ \beta_2)$. The first experiments consider the complex shift β_2 only and keep β_1 fixed. For the most promising complex shift, the real shift β_1 is altered. the last few experiments are performed for some special values of both shift parameters.*

5.4 The low memory nested, preconditioned GCR method.

The results in Section 5.3.3 show that a reduction in the number of outer iterations and hence, in the amount of memory requirered for the nested, preconditioned GCR method can be obtained if more inner iterations are used. The results from Section 5.3.4 imply a great reduction in CPU time if the shift parameters $(\beta_1,\ \beta_2)$ are changed to the ideal couple found just before. This means that it is possible with this method to solve larger systems of linear equations. The total CPU time to solve these systems will increase, since the CPU time per outer iteration increases more rapidly than the number of outer iterations decreases.

The numerical experiments that led to the trend as described in Section 5.2 are repeated in this section. The difference is that the tuned parameters[4] $(\beta_1,\ \beta_2) = (1.0, 2.5)$ are used and the number of inner iterations $It_i = 500$, hence, the innerloop always converges to the prescribed tolerance $\varepsilon_i = 10^{-1}$. The number of middle iterations is fixed $It_j = 10$. Table 5.6 summarizes the results, whereas Figure 5.8 illustrates the relation between the number of degrees of freedom and the number of outer iterations to reach tolerance ε_o. In Table 5.6 an extra column is used for the

[4]The experiments should have been performed with the ideal shift-parameter couple obtained from Table 5.5, $(\beta_1,\ \beta_2) = (0.5, 3.0)$. Unfortunately these tests were alreay performed before these values, to decrease the CPU time even more, were known.

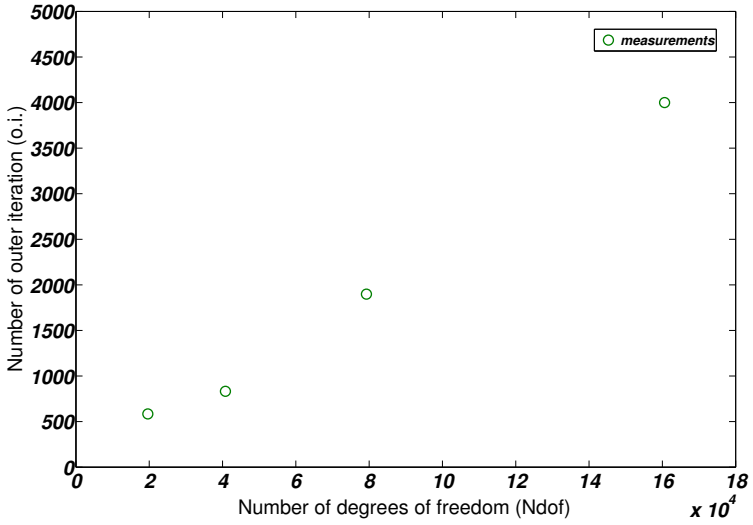

Fig. 5.8 The number of outer iterations as a function of the number of degrees of freedom. A rectangular cavity of dimensions $1.5 \times 1.5 \times L$, $L \in \{1, 2, 4, 8\}$ *is considered here, where* $h = 0.25$, $(\beta_1, \beta_2) = (1.0, 2.5)$.

amount of *necessary* memory to perform the iterative method. This is not the amount of memory *reserved* on the computer since the average number of inner iterations to reach ε_i and the number of outer iterations are unknown on beforehand.

dimensions	N_{el}	Ndof	$It_k, It_j, \overline{It_i}$	CPU	mem$_n$ (Gb)	mem$_r$ (Gb)
$1.5 \times 1.5 \times 1.0$	1111	19614	584,10,34	00:34:47	0.56	2.0
$1.5 \times 1.5 \times 2.0$	2268	39840	832,10,35	01:47:33	1.47	4.0
$1.5 \times 1.5 \times 4.0$	4580	80283	1898,10,36	08:02:18	5.54	8.2
$1.5 \times 1.5 \times 8.0$	9175	160599	4000,10,44	82:00:05	22.04	29

Table 5.6 $N dof_{ap} = 741$ *fixed. The shift parameters are fixed* $(\beta_1, \beta_2) = (1.0, 2.5)$.

To illustrate Table 5.6, the norm of the relative residual as a function of the number of outer iterations is illustrated in Figure 5.9 for the model problem with Ndof $\approx$ 160000. The number of middle iterations is fixed on $It_j = 20$ and $(\beta_1, \beta_2) = (0.5, 2.5)$. The maximum number of inner iterations is chosen large enough to ensure convergence of the inner-loop to the inner-loop tolerance $\varepsilon_i = 10^{-1}$. From Figure 5.9 it is concluded that the number of middle iterations is too small. In the first 500 outer iterations, $It_j = 20$ in enough to reach the prescribed tolerance $\varepsilon_j = 10^{-1}$ in the middle loop. But after this, every outer iteration uses exactly 20 middle iterations and the tolerance ε_j is not reached after these 20 middle iterations. This results in a

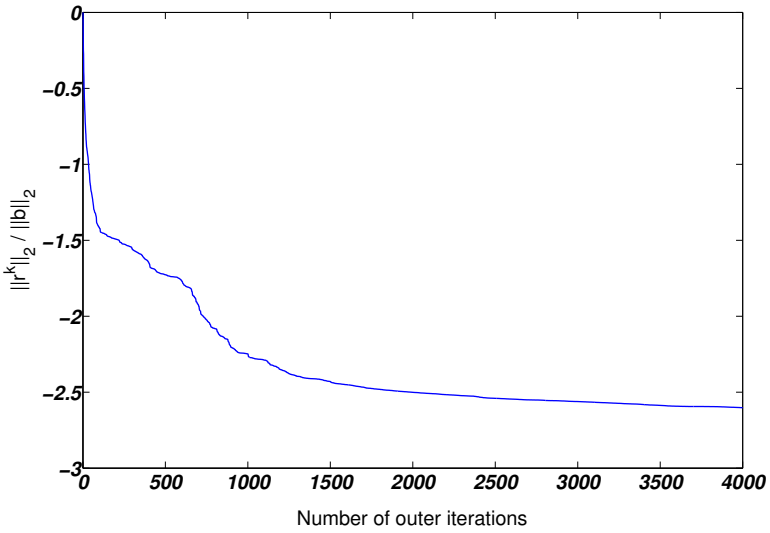

Fig. 5.9 The relative residual as a function of the number of outer iterations for the model problem with dimensions $1.5 \times 1.5 \times 8.0$, $h = 0.25$ *and* $Ndof = 160599$.

bad approximation of the search direction s_1^k. An improvement in the rate of convergence can therefore be expected if the number of middle iteration is enlarged to for example $It_j = 100$. Of course this is very CPU time intensive and another method to improve the rate of convergence is explained in the next section. The basic idea is the construction of a new preconditioner, combined from the triangular- and shifted Laplace preconditioner.

5.5 A combination of the triangular- and the shifted Laplace preconditioner

Finally, the new preconditioned GCR method as described in Section 4.2.4 is used to determine a relation between the number of degrees of freedom and the corresponding CPU time. Again, exactly the same experiment as before is performed. The results of this new experiment are summarized in Table 5.7

N_{el}	Ndof	It_k to $\varepsilon_o = 10^{-4}$	$\overline{It_i}$	CPU time
2319	40785	938	28.33	00:11:37
4525	79266	1500	28.52	00:28:16
9175	160599	1812	68.17	03:02:16
18209	321066	3500	75.46	13:41:22

Table 5.7 *The performance of the preconditioned GCR method with the new preconditioner. The inner-loop tolerance is fixed,* $\varepsilon_j = 10^{-1}$ *for the first two experiments. The latter two use an inner-loop tolerance* $\varepsilon_j = 10^{-2}$.

From this table it is observed immediately that there is a huge CPU time reduction compared to the former nested preconditioned GCR method using both preconditioners (see Table 5.6). The question remains though what the effect of the inner-loop tolerance ε_j is on the number of outer iterations. The expectation is that, since the preconditioner M_2 used to compute a new search direction is an approximation of A_{11} only, it is not necessary to choose ε_j very small. To check this heuristic explanation, the following numerical experiment has been performed. Using a model problem, the inner-loop tolerance ε_j is decreased from 10^{-1} to 10^{-8} and the corresponding number of outer iterations, average number of inner iterations and the CPU time are depicted in Table 5.8. The table is illustrated in Figure 5.10, where the relative residual as a function of the number of outer iterations is showed for the various choices of ε_j. It is clear from both Table and Figure that the number of outer iterations does not decrease very much if the inner-loop is solved with a higher accuracy.

ε_j	CPU	$\|\|r^k\|\|_2/\|\|b\|\|_2$	$\overline{It_i}$	It_k
10^{-1}	00:22:18	$10^{-2.51}$	28.18	> 1000
10^{-2}	00:41:20	$10^{-3.96}$	59.81	> 1000
10^{-3}	01:09:07	$10^{-4.0}$	97.18	896
10^{-4}	01:43:50	$10^{-4.0}$	137.31	891
10^{-8}	04:27:20	$10^{-4.0}$	298.73	880

Table 5.8 *The influence of a high accuracy in the inner-loop on the number of outer iterations. Note that for these experiments the maximum number of outer iterations equals* $It_k = 1000$. *The first two are not converged to the tolerance* $\varepsilon_o = 10^{-4}$ *after this number of outer iterations.*

5.5.1 Preconditioned GCR vs. the frontal solver

As already mentioned in Section 5.2.2, the experiment to compare the iterative method with the frontal solver will be done in this section too for the improved GCR method. Table 5.7 shows that a system consisting of approximately 80000 degrees of freedom is solved in about half an hour. Since the model problems concerned in Section 5.2.2 have roughly the same number of degrees of freedom, the CPU time for both the preconditioned GCR method as well as the frontal solver is illustrated in Figure 5.11 as a function of the number of degrees of freedom on the aperture. This Figure seems to indicate that the preconditioned GCR method will be faster if $N\text{dof}_{ap} \geq 3500$. In fact this is only the case if the number of degrees of freedom $N\text{dof}$ is not changed.

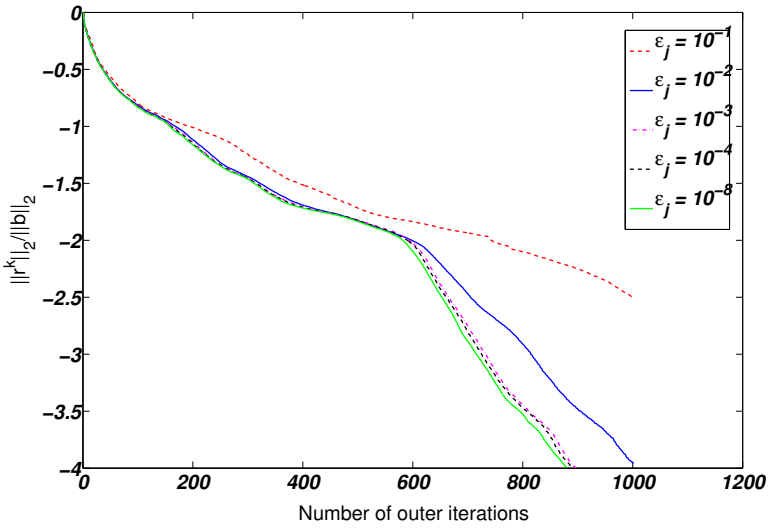

Fig. 5.10 The relative residual as a function of the number of outer iterations for various values of the inner-loop tolerance ε_j. The model problem has dimensions $1.5 \times 1.5 \times 4.0$ and $Ndof = 79266$.

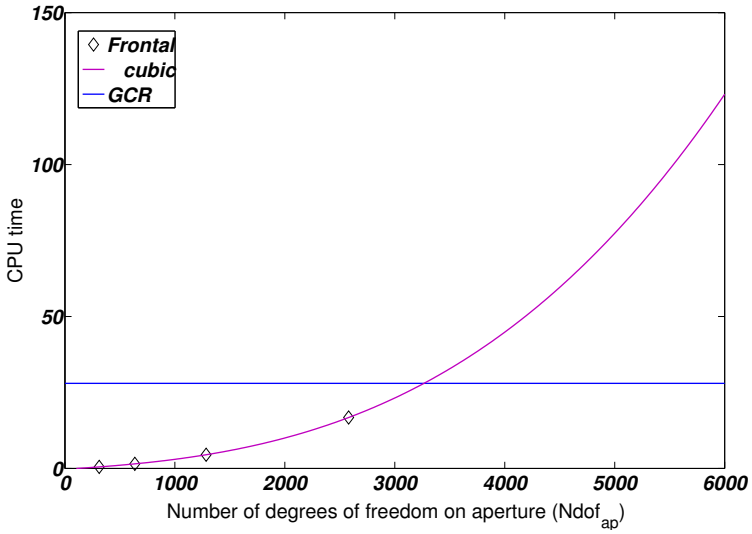

Fig. 5.11 The CPU time as a function of the number of degrees of freedom on the aperture for the preconditioned GCR method and the frontal solver for a model problem with dimensions $1.5 \times 1.5 \times 4.0$ and $Ndof = 79266$.

5.5.2 Memory requirements for the new preconditioner

It has been observed that using the new preconditioner M_{new}, the CPU time is greatly reduced compared to the older version of the nested preconditioned GCR method with two preconditioners. But how much memory is required for the new version of the algorithm? This will be

investigated below.

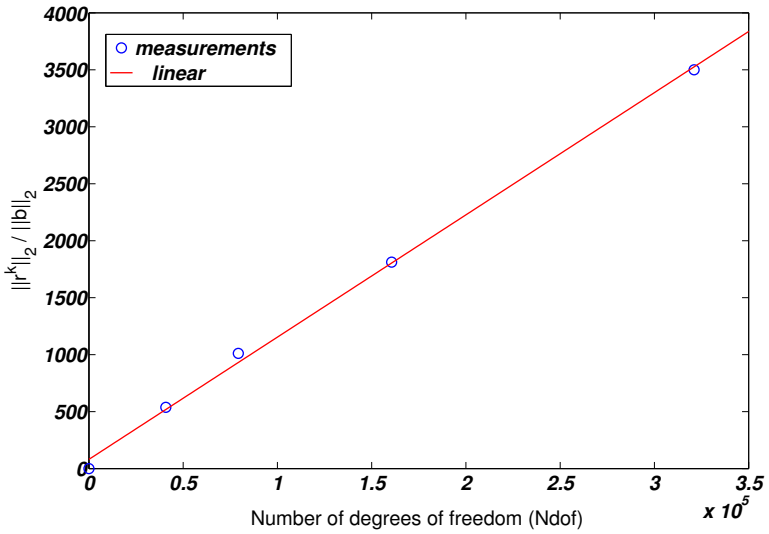

Fig. 5.12 The number of outer iterations It_k as a function of the number of degrees of freedom Ndof for a series of test problems with dimensions $1.5 \times 1.5 \times L$, $L \in \{1, 2, 4, 8\}$.

The main part of the memory requirements are for the storage of the two Krylov subspace bases. If an upper bound for the number of outer iterations can be given, the amount of memory to store the basisvectors equals

$$2 \cdot N\text{dof} \cdot max_{it} \cdot 16\,\text{b}. \tag{5.5}$$

For a model problem with approximately $3.0 \cdot 10^5$ degrees of freedom and maximal 3500 outer iterations this is approximately 33 Gb. According to Figure 5.12, the number of outer iterations roughly doubles if the number of degrees of freedom doubles. Hence, a model problem where $N\text{dof} = 6.0 \cdot 10^5$ with 7000 outer iterations would require already 134 Gb. From this calculation it is clear that with the current algorithm the memory requirements exceed the amount of memory in core of the SX-8R computer.

5.5.3 Transferring data between CORE and a fast hard disk

The SX-8R vector machine in use at the NLR contains a so-called *fast hard disk*. This means theoretically that transferring data from core to this fast disk will not take a large amount of time. Hence, a way to decrease the amount of memory needed in core of the supercomputer, is to write the Krylov subspace basisvectors in batches to a file. Every iteration the new vector under construction has to be orthogonalized with respect to all previous basis vectors. These basis vectors are read into core in batches such that the amount of memory needed per batch is controllable. In

batchsize	CPU time
100	00:30:47
500	00:29:52
no-batch	00:28:16

Table 5.9 *The influence of the batchsize on the CPU time for a model problem with Ndof ≈ 80000. The row with* no-batch *corresponds to an experiment where no transferring of data is performed.*

this way the basis vectors from the batches are read in successively and the new vector is orthogonalized with respect to the vectors in the batch.

The question remains how fast this data transfer really is. Therefore, an experiment with a variable batchsize is performed for the model problem where Ndof ≈ 80000. In Table 5.9, the CPU times are compared for various batchsizes. This table shows that the batchsize has no influence on the CPU time for this example. The approach of data transfer will also be used for the system of linear equations in the next chapter.

6 Application of the preconditioned GCR method to a bended cavity

The final iterative method derived in Section 4.2.4 with the data transfer as described in Section 5.5.3 will be used in this chapter to solve a system of linear equations obtained from the finite element discretization of the vector wave equation in a cylindrical bended inlet (see Figure 6.1).

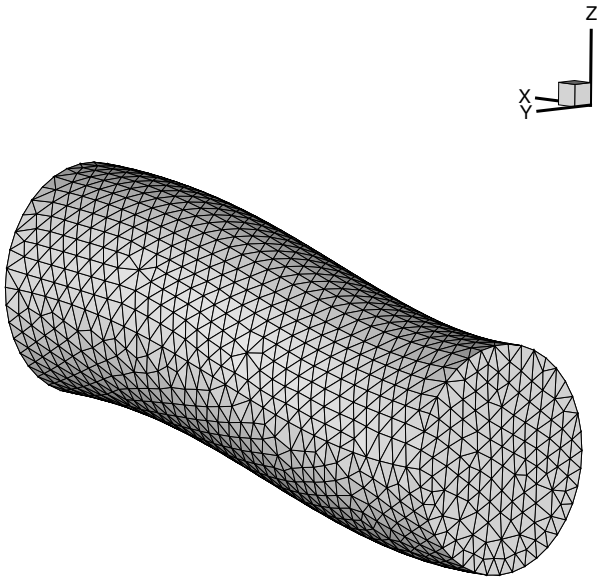

Fig. 6.1 A bended inlet of intermediate length discretized with tetrahedra (interior) and triangles (aperture). This inlet has a length-depth ratio of 2.5. The total number of degrees of freedom $N\mathit{dof} = 553965$, $N\mathit{dof}_{ap} = 2556$.

For this application, the number of elements in the mesh is $N_{el} = 30707$ yielding a total number of degrees of freedom $N\text{dof} = 553965$ and 2556 degrees of freedom on the aperture. According to Figure 5.12, the maximum number of outer iterations is fixed on $max_{it} = 10000$ in order to reach the prescribed tolerance $\varepsilon_o = 10^{-4}$. As described in Section 5.5.2 it is not possible to store the complete Krylov subspace bases, hence the basisvectors have to be separated in batches. Roughly 30 to 40 Gb is reserved for the element matrices, mappings and vectors used by the method. Since the iterative method runs on a single processor, maximal 50 Gb can be reserved. This yields a batchsize of 750 (the basisvectors use 14 Gb for storage in this case).

6.1 Performance of the preconditioned GCR method

According to Table 5.7, the CPU time for this application is approximately 40 a 50 hours. Since this experiment could not be finished in the given time, only the performance for the first part of this experiment is given below.

- The number of outer iterations is $It_k = 2280$
- The average number of inner iteration is $\overline{It_j} = 40.23$
- The CPU time is approximately 140 hours
- The amount of memory needed in core is 50.75 Gb

The preconditioned GCR method using the preconditioner M_{new} is converged to the tolerance $10^{-1.65}$ using 2280 outer iterations. The convergence behavior is illustrated in Figure 6.2. The reason why the actual CPU time is much larger then expected is the relative slow data transfer. The fast hard disk has a performance of 400 Mb per second. Hence, if 120 Gb data has to be transfered this already takes 5 minutes. Hence, the overall performance of the preconditioned GCR method is strongly influenced when using the fast hard disk for out-of-core storage of the Krylov basis vectors.

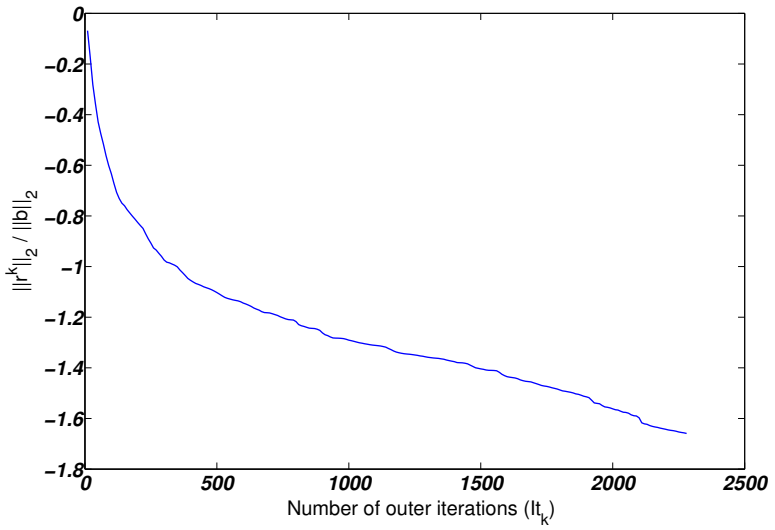

Fig. 6.2 The convergence behavior of the preconditioned GCR method for the bended inlet.

The linear system is also solved using the frontal solver. Since the number of degrees of freedom on the aperture is 2556 only, it is expected that the CPU time for the frontal solver is about 20 minutes according to Table 5.2 in Section 5.2.2. However, the domains in the corresponding problems to this table are chosen in such a way that the total number of degrees of freedom is constant. This means that the cavity is undeep (depth to diameter ratio is 0.35). The target prob-

lem as illustrated in Figure 6.1 considers a cavity with a depth to diameter ratio of 2.5. Therefore, the actual CPU time is expected to be far more than 20 minutes. The frontal solver has computed the solution in $04 : 30 : 18$.

7 Recommendations

During this thesis a start has been made with the application of a shifted Laplace operator preconditioner for the solution of the discretized vector wave equation. To further improve the algorithm the following recommendations for future research are made:

- Inclusion of a multigrid method in the innerloop.
- Use of higher order basis functions for the electric field.
- Use of short recurrence Krylov method for the outer iteration.
- Parallelization of the algorithm.
- Investigation of the tolerance to be reached in the outer loop.
- Theoretical optimization of the parameters $(\beta_1,\, \beta_2)$ by spectral analysis.
- Efficient (simultaneous) solution for multiple right-hand sides.

Multigrid method

The number of iterations in the innerloop, where the shifted Laplace operator preconditioner is used is relatively high $(It_i \approx 30)$ resulting in large amount of CPU time for an accuracy of only 0.10. Therefore, it is strongly recommended to use a fast *multigrid* method to solve the system $M_2\, y = z$. According to Erlangga et al. (Ref. 12), it is possible to reach the accuracy $\varepsilon = 0.10$ with only one multigrid cycle. This would be a decrease in the amount of work and CPU time of a factor 30. However, the efficient and robust shifted Laplace operator with multigrid has only been succesfull applied for the Helmholtz equation so far, the scalar form of the vector wave equation. The efficiency of the multigrid algorithm for the discretized vector wave equation has to be investigated.

Higher order basis functions

In Chapter 3 and in Hooghiemstra (Ref. 3), the necessity to minimize the dispersion error is explained. In the present implementation of the finite element discretization, the elements are of second order and the resulting linear system has approximately $20 \cdot 10^6$ degrees of freedom. If even higher order basis functions (e.g. $p = 4$ or $p = 5$) are used , a higher accuracy is obtained using far less degrees of freedom. In order to use these higher order basis functions, new polynomial coefficients have to be computed. This number increases very fast since the number of local basis functions per element is a function of the element order in the following way

$$nlocbas(p) = (p+1)[6 + 4p + \frac{p(p-1)}{2}]. \tag{7.1}$$

For fourth order basis functions this yields 140 local basis functions and for fifth order elements the number of local basis functions increases to 216.

Use of a short recurrence Krylov method

The first numerical experiments in Chapter 5 showed that in order to have a reasonably fast converging iterative method, all the basis functions of the Krylov subspace had to be stored. The storage cost increases linearly with the number of iterations resulting in memory requirements that can not be met for very large problems ($N\text{dof} \geq 5.0 \cdot 10^5$). Therefore, it might be worth the effort to choose another iterative method with short recurrences, although the monotone decreasing residual property will be lost.

Parallelization of the algorithm

The present algorithm is not parallelizable and hence it runs on a single processor at the moment. If a domain decomposition method would be applied, it is possible to parallelize the algorithm.

The outer-loop tolerance

As already mentioned in Section 5.2.2, the frontal solver solves the system of linear equations exactly, while a solution within the range of the dispersion error ε is accurate enough. In the nested preconditioned GCR method it is possible to set the desired tolerance. But the effect of this tolerance ε_o to the actual approximation of the solution should be investigated.

Theoretical optimization of the parameters $(\beta_1,\, \beta_2)$

The shift parameters $(\beta_1,\, \beta_2)$ have been optimized heuristically in the current method. To take full advantage of the shifted Laplace operator preconditioner it is necessary to investigate the influence of these parameters on the convergence if a multigrid method is used to solve the preconditioner system $M_2\, y \,=\, z$. This can be achieved by analysing the spectral properties of the preconditioner system.

Simultaneous solution for multiple right-hand sides

As indicated in Section 2.2.2.1, a typical analysis requires computation of the electric field for a large number of different excitations. In the current implementation of the nested preconditioned GCR method, the number of right-hand sides to be treated at once is 1. The frontal solver on the other hand has the property to be able to solve the linear system for numerous right-hand sides with only a little extra effort and CPU time. It should be investigated whether the iterative

method can be improved to solve the system for multiple right-hand sides, e.g. by means of a dedicated Krylov method like Block-GMRES.

8 Conclusion

It has been demonstrated that the shifted Laplace preconditioner can be used effectively to speed up the convergence rate of the GCR method for solution of the linear system resulting from higher order finite element discretisation of the vector wave equation. However, the efficiency of the preconditioned Krylov method strongly depends on the amount of work involved in solution of the preconditioner system. Currently, a basic approach has been implemented that uses GCR to solve the preconditioner system, as opposed to the multigrid method advocated in the original publications of Erlangga et al. (Ref. 12). The current approach already gives satisfactory results for intermediate system sizes up to $N\text{dof} = 3.0 \cdot 10^5$. A number of recommendations have been formulated to make the approach practically applicable for very large systems.

To reduce the storage required by the GCR method, a number of different approaches have been analysed, including truncation, restarting, and out-of-core storage of the Krylov basis vectors. Although truncation and restart methods yield a low memory algorithm, superlinear convergence of the GCR method is lost and a slow converging method is obtained. Out-of-core storage of the Krylov basis vectors is an ideal way to save memory in core, but the data transfer speed is supposed to increase to be able to analyse larger systems of linear equations.

Contrary to the frontal solver the work required by the preconditioned GCR method depends solely on the number of degrees of freedom in the system, with no or only weak dependence on the number of unknowns in the aperture of the cavity.

References

1. Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2000.
2. G. H. Golub ans C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.
3. P. B. Hooghiemstra. Full wave analysis of the contribution to the radar cross section of the jet engine air intake of a generic fighter aircraft. Technical Report NLR-TR-2007-310, National Aerospace Laboratory NLR, 2007.
4. A. F. Peterson, S. L. Ray, and R. Mittra. *Computational Methods for Electromagnetics*. IEEE PRESS, first edition, 1998.
5. J. Jin. *The Finite Element Method in Electromagnetics*. John Wiley and Sons, second edition, 2002.
6. J. C. Nedelec. Mixed finite elements in $\mathbb{R}^3$. *Numer. Mathem.*, 35:315–341, 1980.
7. R. D. Graglia, D. R. Wilton, and A. F. Peterson. Higher order interpolatory vector bases for computational electromagnetics. *IEEE Trans. Magn.*, 45(3):329–342, 1997.
8. P. P. Silvester and R. L. Ferrari. *Finite Elements for Electrical Engineers*. Cambridge Press, University of Cambridge edition, 1990.
9. D. R. van der Heul, H. van der Ven, and J. W. van der Burg. Full wave analysis of the influence of the jet engine air intake on the radar signature of modern fighter aircraft. European Conference on Computational Fluid Dynamics.
10. S. M. Rao, D. R. Wilton, and A. W. Glisson. Electromagnetic scattering by surfaces of arbitrary shape. *IEEE Trans. Magn.*, AP-30(3):409–418, 1982.
11. J. D. Greenwood and J. Jin. A hybrid mom/fem technique for scattering from a complex bor with appendages. *Aces Journal*, 15(1):1–12, 2000.
12. Y. A. Erlangga. *A robust and efficient iterative method for the numerical solution of the Helmholtz equation*. PhD thesis, Delft University of Technology, 2005.
13. Y. A. Erlangga, C. Vuik, and C. W. Oosterlee. On a class of preconditioners for solving the Helmholtz equation. *Appl. Num. Math.*, 50:409–425, 2004.
14. Y. A. Erlangga, C. W. Oosterlee, and C. Vuik. A novel multigrid based preconditioner for heterogeneous Helmholtz problems. *SIAM J. Sci. Comput.*, 27:1471–1492, 2006.
15. M. B. van Gijzen, Y. A. Erlangga, and C. Vuik. Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplacian. *SIAM J. Sci. Comput.*, 29:1942–1958, 2007.
16. C. Vuik and C. W. Oosterlee. *Scientific Computing*. Lecture Notes. TU Delft, 2005.
17. E. Turkel and Y. A. Erlangga. Preconditining a finite element solver of the exterior Helmholtz equation. European Conference on Computational Fluid Dynamics.

18. J. Jin, J. Liu, Z. Lou, and C. S. T. Liang. A fully high-order finite-element simulation of scattering by deep cavitities. *IEEE Trans. Magn.*, 51(9):2420–2429, september 2003.
19. H. A. van der Vorst and C. Vuik. The superlinear convergence behaviour of gmres. *J. Comp. Appl. Math.*, 48(3):327–341, 1993.

Appendix A Grid figures

This appendix contains the grids corresponding to various convergence plots in this report.

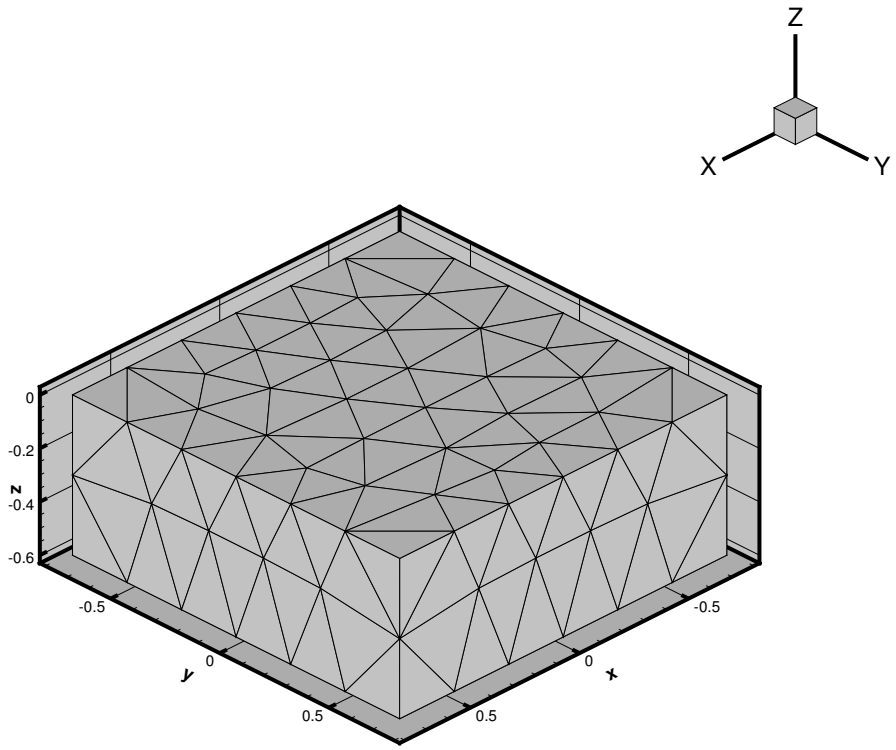

Fig. A.1 The rectangular cavity $(1.5 \times 1.5 \times 0.6)$ *discretized on a fine grid. The meshwidth is* $h = 0.25$, *the number of elements of the grid equals 681.*

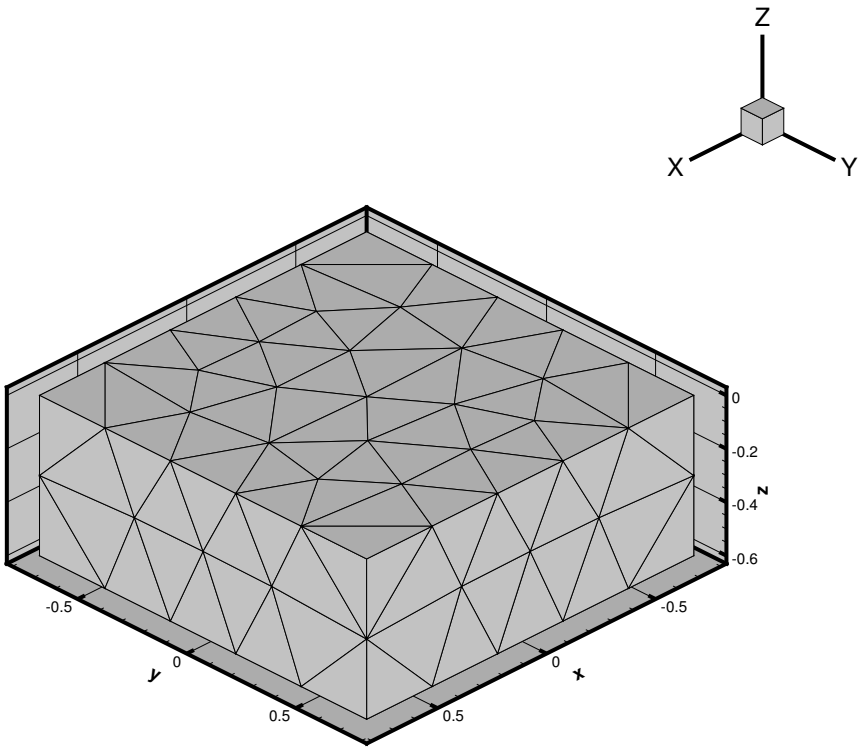

Fig. A.2 The rectangular cavity $(1.5 \times 1.5 \times 0.6)$ *discretized on a fine grid. The meshwidth is* $h = 0.30,$ *the number of elements of the grid equals 343.*

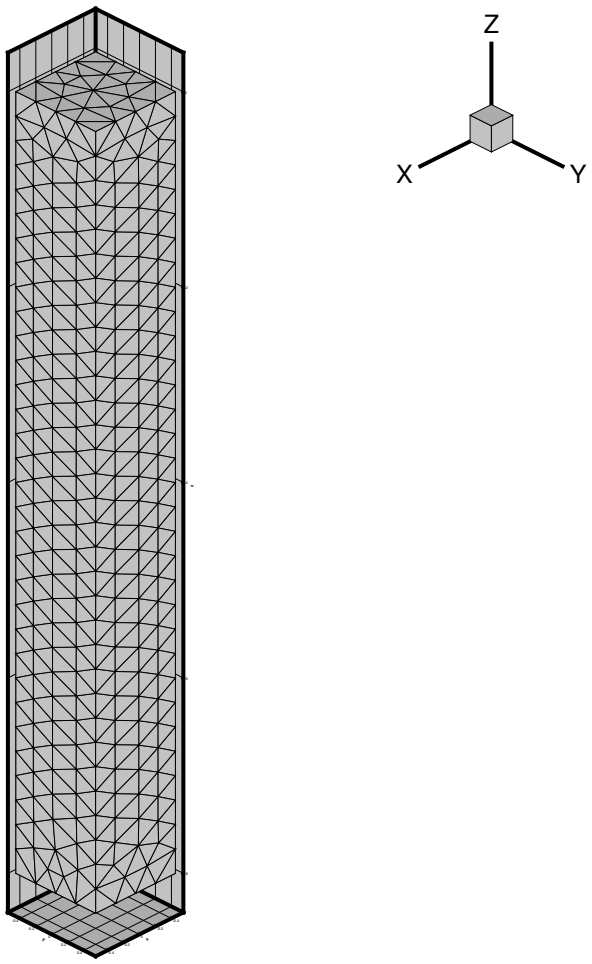

Fig. A.3 The rectangular cavity $(1.0 \times 1.0 \times 8.0)$ *discretized on a fine grid. The meshwidth is* $h = 0.25$, *the number of elements of the grid equals 3904.*

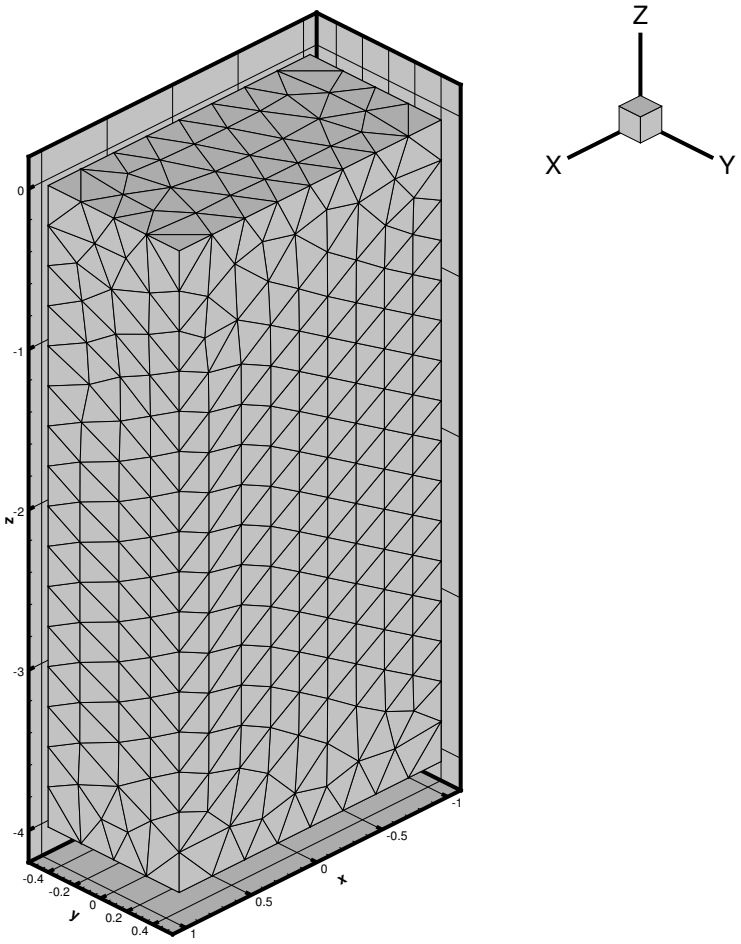

Fig. A.4 The rectangular cavity $(2.0 \times 1.0 \times 4.0)$ *discretized on a fine grid. The meshwidth is* $h = 0.25$, *the number of elements of the grid equals 4035.*

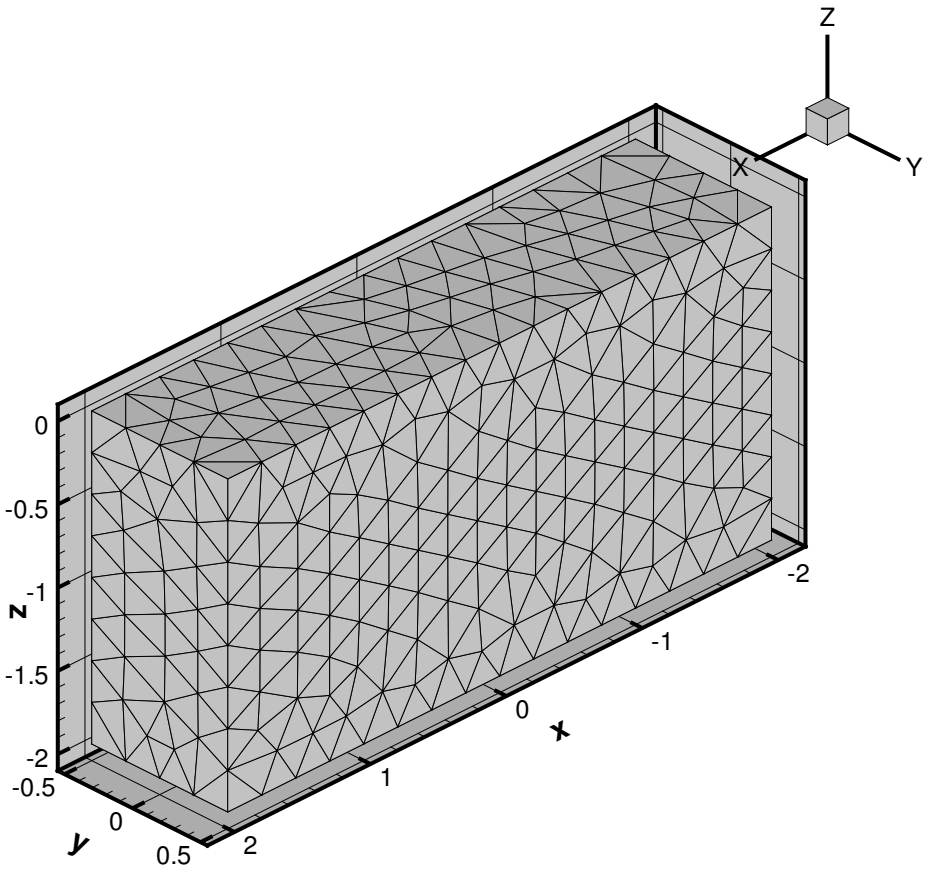

Fig. A.5 The rectangular cavity $(4.0 \times 1.0 \times 2.0)$ *discretized on a fine grid. The meshwidth is* $h = 0.25$, *the number of elements of the grid equals 3995.*

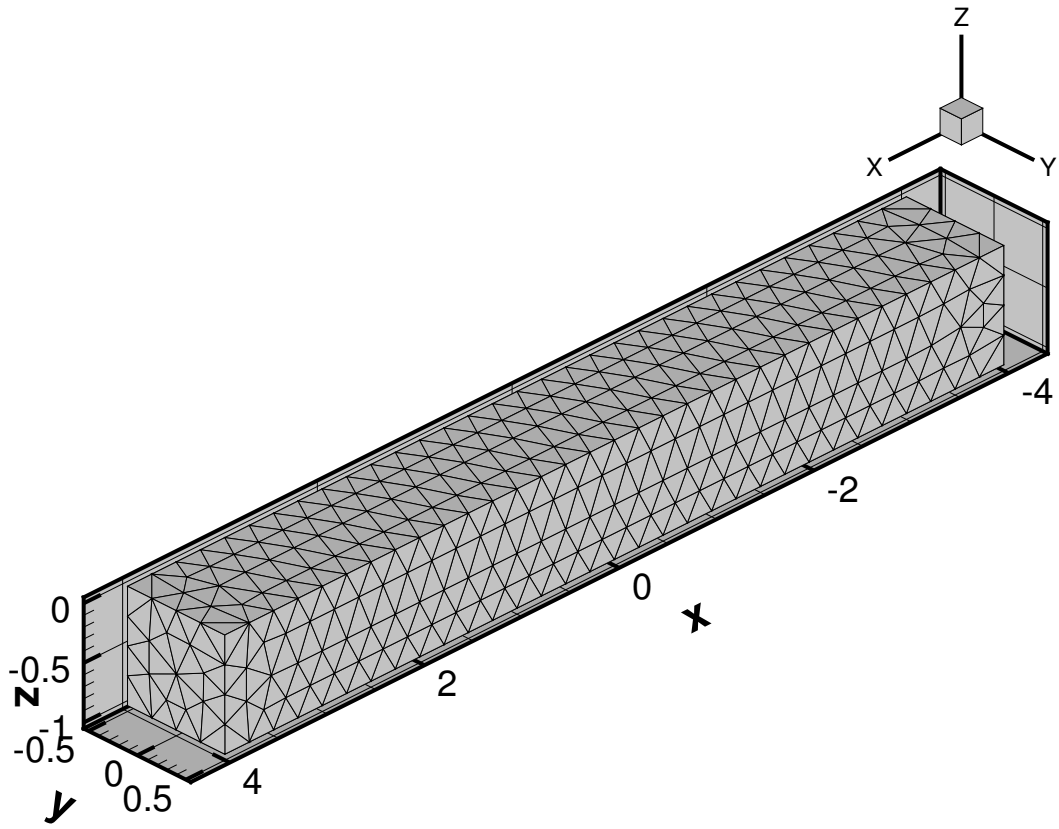

Fig. A.6 The rectangular cavity $(8.0 \times 1.0 \times 1.0)$ *discretized on a fine grid. The meshwidth is* $h = 0.25$, *the number of elements of the grid equals 3874.*

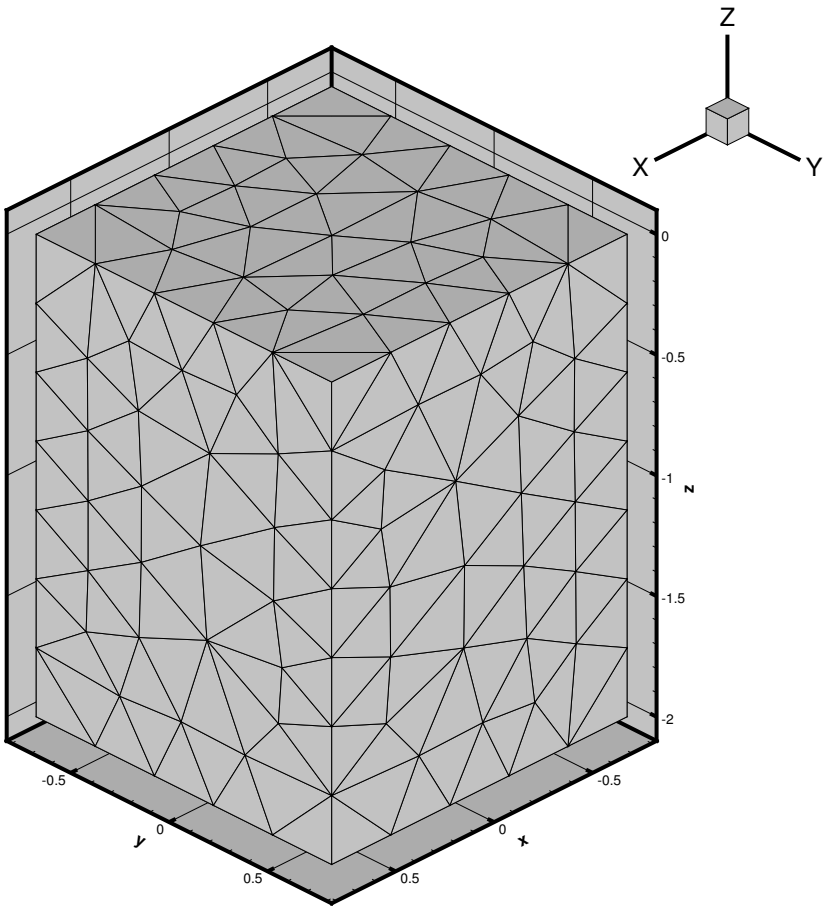

Fig. A.7 The rectangular cavity $(1.5 \times 1.5 \times 2.0)$ *discretized on a fine grid. The meshwidth is* $h = 0.30$, *the number of elements of the grid equals 1287.*

Appendix B Work

In Chapter 4, some details concerning the amount of work to be performed in the iterative methods were not treated. Therefore, this section explains the work in more detail. The following four operations are analysed:

- Addition of two complex valued numbers.
- Multiplication of two complex valued numbers.
- The matrix vector product $A_{11}\, y$.
- The matrix vector product $M_2\, y$.
- The forward substitution algorithm to solve $L_{22} w = r_2^{k-1}$

Starting with the first one it is immediately seen that the addition of two complex numbers $(\alpha + i\beta) + (\gamma + i\delta)$ consists of 3 real valued additions. Hence, a complex addition costs 3 flops. The product of two complex valued numbers is more involved. This product consists of the following operations

$$(\alpha + i\beta) * (\gamma + i\delta) = \alpha * \gamma + i\alpha * \delta + i\beta * \gamma + i^2\beta * \delta. \tag{B.1}$$

Altogether this yields 4 real valued multiplications and 3 additions. Hence, such a product costs 7 flops.

The matrix product $A_{11}\, y$ is the next operation. Note that the matrix A_{11} is real valued, but the product vector y is complex valued. Since there are $\tilde{c}$ nonzeros per row in A_{11}, the work per row is equal to

$$\tilde{c} \text{ products of real} * \text{complex} + (\tilde{c} - 1) \text{ complex additions.} \tag{B.2}$$

Such a product costs 3 flops and an addition also yielding $3\tilde{c} + 3(\tilde{c} - 1) \approx 6\tilde{c}$ flops per row. Therefore,

$$W(A_{11}\, y) = 6\tilde{c}(n - f). \tag{B.3}$$

In exactly the same way the product $M_2\, y$ is treated. The only difference with the former product is the matrix, since M_2 is complex valued. Hence, the work per row consists of

$$\tilde{c} \text{ complex products} + (\tilde{c} - 1) \text{ complex additions.} \tag{B.4}$$

Using the results for complex products, the work involved in this matrix vector product is

$$W(A_{11}\, y) = 10\tilde{c}(n - f). \tag{B.5}$$

Finally, the forward substitution algorithm is treated. This is given below in Algorithm B.1 (Ref. 16):

Algorithm B.1 Forward substitution algorithm

in: L_{22}, r_2^{k-1}
out: w

for $i = 1, \ldots, f$ **do**
  $w(i) := r_2^{k-1}(i)$
  for $j = 1, \ldots, i-1$ **do**
    $w(i) := w(i) - L_{22}(i,j)w(j)$
  end for
  $w(i) := w(i)/L_{22}(i,i)$
end for

In the j-loop one complex product and an addition are performed yielding 8 flops. This is done $i-1$ times. Hence $8(i-1)$ flops are performed. Then there is one complex division, which is assumed to be as expensive as a complex product yielding $8(i-1)+7$ flops. Adding the i-loop one obtains the following sum

$$\sum_{i=1}^{f} 8(i-1)+7 = \sum_{i=1}^{f} 8i - 1 = -f + 8\sum_{i=1}^{f} i = -f + \frac{8}{2}f(f+1) \approx 4f^2. \tag{B.6}$$

Appendix C Algorithms

This appendix contains several algorithms referenced to from the text in Chapters 4 and 5. The first two algorithms describe the matrix free matrix vector product and the coloring algorithm. The last three algorithms are slight adjustments of the orthonormalization algorithm (Algorithm 4.4.)

Algorithm C.1 Matrix free product

in: $E[N_{el}, nlocbas^2]$, $map[N_{el}, nlocbas]$, $x[N\text{dof} + nlocbas]$.
out: solution $z[N\text{dof}]$

for $n = 1, \ldots, N_{el}$ **do**
  for $j = 1, \ldots, nlocbas$ **do**
    for $k = 1, \ldots, nlocbas$ **do**
      $z(map(n, nlocbas(j-1)+k)) = z(map(n, nlocbas(j-1)+k))) +$
        $E(n, nlocbas(j-1)+k)) * x(map(n, nlocbas(j-1)+k))).$
    end for
  end for
end for

In the algorithm given above, E is an array containing the element matrices, map is an array containing the corresponding mappings and x is the product vector. At first glance, this algorithm looks perfectly vectorisable, but in fact it is not. This is due to the following problem. Degrees of freedom which are placed on the Dirichlet boundary $\hat{n} \times \mathbf{E} = 0$, have a zero value in the mapping, since the corresponding global degree of freedom is zero. This may cause the occurence of *out of bound errors*.

In first instance this has been circumvented by giving these zeros a dummy value $N\text{dof} + 1$, where $N\text{dof}$ is the number of degrees of freedom in the system. The product vector has an extra dummy element on position $N\text{dof} + 1$, which has value zero. Doing this, no extra if statement is necessary to tell whether the mapping entry is zero. But on the other hand this solution also has a severe drawback. The routine is not vectorisable if several elements of one mapping are the same, since simultaneous updating of one vector position with two values is impossible. Still, this might be the case for an element with one face attached to the boundary.

A solution to this problem, which makes the routine perfectly vectorisable is to extend the number of dummy variables. Let the dimension of the mapping vector and the corresponding element matrix be $nlocbas$ (**n**umber of **loc**al **bas**is functions). Then there can be at most $nlocbas$ zeros in the mapping. If all these zeros are given a unique dummy value starting at $N\text{dof} + 1$, yields a mapping where no equal degrees of freedom will be present.

Algorithm C.2 Element coloring (**Part 1**)

in: tet-to-edge, Nedges, $veclen$
out: color, ipcolor

```
icolorsw = 1
for i = 1, ..., N_el do
  remaining(i) = i
end for
for istart < Nel do
  nelems = 0
  nelemr = 0
  for i = istart + 1, ..., N_el do
    tet = remaining(i)
    for nelems < veclen do
      tel = 0
      for k=1:6 do
        if icolorsw(tet - edge(tet, k)) == 1 then
          tel = tel + 1
        end if
      end for
      if tel == 6 then
        nelems = nelems + 1
        color(ipcolor(icol) + nelems) = tet
        icolorsw(tet - edge(tet, 1 : 6)) = 0
      else
        nelemr = nelemr + 1
        elemr(nelemr) = tet
      end if
    end for
  end for
  iend = i - 1
  icol = icol + 1
  ipcolor(icol) = ipcolor(icol - 1) + nelems
  icolorsw = 1
  for i = iend, iend - nelemr - 1, -1 do
    remaining(i) = elemr(i - iend + nelemr)
  end for
  istart = iend - nelemr
end for
ncol = icol
```

Algorithm C.3 Element coloring (**Part 2**)

in: tet-to-edge, Nedges, $veclen$
out: color, ipcolor

$icolorsw = 1$
for $i = 1, \ldots, ipcolor(ncol)$ **do**
  $remaining(i) = color(i)$
end for
$jbuff = \frac{Nel}{veclen} + 1$
$ntetc = \frac{Nel}{jbuff} + 1$
$j = 0$
for $istart > 0$ **do**
  $j = j + 1$
  $nelems = 0$
  $nelemr = 0$
  for $i = istart, 1, -1$ **do**
    for $nelems < veclen$ **do**
      $tet = remaining(i)$
      $tel = 0$
      for k=1:6 **do**
        if $icolorsw(tet - edge(tet, k)) == 1$ **then**
          $tel = tel + 1$
        end if
      end for
      if $tel == 6$ **then**
        $nelems = nelems + 1$
        $color(ipcolor(icol) + nelems) = tet$
        $icolorsw(tet - edge(tet, 1:6)) = 0$
      else
        $nelemr = nelemr + 1$
        $elemr(nelemr) = tet$
      end if
    end for
  end for
  $iend = i + 1$
  $icol = icol + 1$
  $ipcolor(icol) = ipcolor(icol - 1) + nelems$
  $icolorsw = 1$
  for $i = iend, iend + nelemr - 1$ **do**
    $remaining(i) = elemr(i - iend + 1)$
  end for
  $istart = iend + nelemr - 1$
end for
$ncol = icol$

Algorithm C.4 Orthonormalization process with truncation *last*.

in: $k, v_1, \ldots, v_k, s_1, \ldots, s_k, trunc$
out: s_k, v_k, with $||s^k||_2 = 1$, $||v^k||_2 = 1$

if $k <= trunc$ **then**
  for $l = 1, \ldots, k-1$ **do**
    $\alpha = (v^l, v^k)$
    $v^k = v^k - \alpha v^l$
    $s^k = s^k - \alpha s^l$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $V(:, k) = v^k$
  $S(:, k) = s^k$
else
  $it = \text{mod}(k, trunc)$
  if $it == 0$ **then**
    $it = trunc$
  end if
  for $l = 1, \ldots, trunc$ **do**
    $\alpha = (v^l, v^k)$
    $v^k = v^k - \alpha v^l$
    $s^k = s^k - \alpha s^l$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $V(:, it) = v^k$
  $S(:, it) = s^k$
end if

Algorithm C.5 Orthonormalization process with truncation *first*.

in: $k, v_1, \ldots, v_k, s_1, \ldots, s_k, trunc$
out: s_k, v_k, with $||s^k||_2 = 1$, $||v^k||_2 = 1$

if $k <= trunc$ **then**
  for $l = 1, \ldots, k-1$ **do**
    $\alpha = (v^l, v^k)$
    $v^k = v^k - \alpha v^l$
    $s^k = s^k - \alpha s^l$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $V(:, k) = v^k$
  $S(:, k) = s^k$
else
  for $l = 1, \ldots, trunc$ **do**
    $\alpha = (v^l, v^k)$
    $v^k = v^k - \alpha v^l$
    $s^k = s^k - \alpha s^l$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $V(:, trunc) = v^k$
  $S(:, trunc) = s^k$
end if

Algorithm C.6 Orthonormalization process with truncation *min alfa*.

in: $k, v_1, \ldots, v_k, s_1, \ldots, s_k, trunc$
out: s_k, v_k, with $||s^k||_2 = 1$, $||v^k||_2 = 1$

if $k <= trunc$ **then**
  for $l = 1, \ldots, k-1$ **do**
    $\alpha = (v^l, v^k)$
    $v^k = v^k - \alpha v^l$
    $s^k = s^k - \alpha s^l$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $V(:, k) = v^k$
  $S(:, k) = s^k$
else
  for $l = 1, \ldots, trunc$ **do**
    $\alpha = (v^l, v^k)$
    $b(j) = \alpha$
    $v^k = v^k - \alpha v^l$
    $s^k = s^k - \alpha s^l$
  end for
  $s^k = s^k / ||v^k||_2$
  $v^k = v^k / ||v^k||_2$
  $[val, pos] = \min b$
  $V(:, pos) = v^k$
  $S(:, pos) = s^k$
end if