

DELFT UNIVERSITY OF TECHNOLOGY

REPORT 08-19

DEVELOPMENT OF A SOLVER FOR LARGE ALGEBRAIC SYSTEM FROM  
STRUCTURAL MECHANICS.

T.B. JÖNSTHÖVEL

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2008

Copyright © 2008 by Department of Applied Mathematical Analysis, Delft,  
The Netherlands.

No part of the Journal may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission from Department of Applied Mathematical Analysis, Delft University of Technology, The Netherlands.

# Contents

<b>Notation</b>	<b>iii</b>
<b>Introduction</b>	<b>v</b>
<b>1 Basics structural mechanics</b>	<b>1</b>
1.1 Continuum model . . . . .	1
1.1.1 Strain . . . . .	2
1.1.2 Stress . . . . .	3
1.2 Equilibrium equation: balance of forces . . . . .	4
1.3 Balancing forces . . . . .	6
1.4 Material response . . . . .	7
1.4.1 Elasticity . . . . .	8
1.4.2 Plasticity . . . . .	9
1.4.3 Viscosity . . . . .	10
1.5 Implementation of material response . . . . .	11
1.5.1 Dissipation of energy . . . . .	12
1.5.2 Multiplicative decomposition . . . . .	12
1.5.3 Generalized model local dissipation . . . . .	14
1.5.4 Plastic response . . . . .	15
1.5.5 Viscoelastic response . . . . .	17
<b>2 Discretization</b>	<b>21</b>
2.1 Finite elements . . . . .	21
2.1.1 Element and shape functions . . . . .	21
2.1.2 Gauss points and numerical integration . . . . .	23
2.2 Discretization balancing of forces . . . . .	24
2.2.1 Static mechanics . . . . .	24
2.2.2 Dynamic mechanics . . . . .	27
2.3 Non-linear material properties . . . . .	28
<b>3 Problems arising in structural mechanics</b>	<b>31</b>
3.1 Problem sizes . . . . .	31
3.2 Improvements algorithm . . . . .	32
3.3 Development issues . . . . .	32
3.4 Singularity of the stiffness matrix . . . . .	32
3.5 Misbalancing due to plasticity and viscosity . . . . .	34
3.5.1 Plastic response surface . . . . .	34
3.5.2 Viscosity . . . . .	35
3.6 Domain decomposition and multiple grids . . . . .	35

---

<b>4</b>	<b>Basics numerical computations and analysis</b>	<b>37</b>
4.1	Direct solution methods . . . . .	38
4.1.1	LU decomposition . . . . .	38
4.1.2	Cholesky factorization . . . . .	39
4.2	Basic iterative methods . . . . .	39
4.2.1	Splitting of matrix . . . . .	40
4.2.2	Gauss-Seidel . . . . .	41
4.3	Preconditioning . . . . .	42
4.3.1	Basic iterative methods . . . . .	43
4.3.2	Incomplete factorization . . . . .	44
4.4	Krylov subspace methods . . . . .	44
4.4.1	Conjugated gradient method . . . . .	45
4.4.2	Preconditioned conjugated gradient method . . . . .	46
4.5	Multigrid . . . . .	47
4.5.1	Basics multigrid (two-grid) . . . . .	48
4.5.2	Multigrid Components . . . . .	50
4.6	Domain decomposition . . . . .	53
4.6.1	Block-Gaussian elimination . . . . .	54
4.6.2	Multiplicative Schwarz . . . . .	56
4.6.3	Additive Schwarz . . . . .	58
4.7	Deflation . . . . .	58
4.7.1	Deflation definitions . . . . .	59
4.7.2	Deflated CG method . . . . .	61
4.7.3	Deflated preconditioned CG method . . . . .	61
4.7.4	Deflation vectors . . . . .	62
4.7.5	Example of solving deflated virtual work equation . . . . .	63
<b>5</b>	<b>Parallel direct solver</b>	<b>67</b>
5.1	Parallel computing . . . . .	67
5.2	MUMPS . . . . .	68
5.3	Test results . . . . .	69
<b>6</b>	<b>Summary</b>	<b>71</b>
	<b>Bibliography</b>	<b>81</b>

# Notation

---

In the thesis tensors are utilized for the governing equations as it is common practice in structural mechanics. There are many different variations on the original tensor notation. The notation introduced in the book of dr. A. Scarpas [10] will be the notation utilized here. Vectors and matrices can be seen as 1st and 2nd order tensors respectively. Sometimes classic linear algebraic notation will be used, e.g. the transpose of a second order tensor  $(\mathbf{A}_{ij})^T = \mathbf{A}_{ji}$ .

## Symbols

Tensors with index notation.

- 1st order tensor:  $A_i$  (vector)
- 2nd order tensor:  $A_{ij}$  (matrix)
- 3rd order tensor:  $A_{ijk}$
- 4th order tensor:  $A_{ijkl}$

All tensors without index notation are written bold and with capitals, e.g. fourth order elasticity tensor,  $\mathbb{C}$ .

Operation	Index notation	Tensor notation
multiply	$A_{ij}B_{ij} = c$	$\mathbf{A} : \mathbf{B} = c$
	$A_{ik}B_{kj} = C_{ij}$	$\mathbf{A} \cdot \mathbf{B} = \mathbb{C}$
	$A_{ij}B_j = C_i$	$\mathbf{A} \cdot \mathbf{B} = \mathbb{C}$
	$a_i b_i = c$	$\mathbf{a} \cdot \mathbf{b} = c$
	$A_{ij}B_{ik} = C_{jk}$	$\mathbf{A}^T \cdot \mathbf{B} = \mathbb{C}$
	$A_{ij}B_{kl} = C_{ijkl}$	$\mathbf{A} \otimes \mathbf{B} = \mathbb{C}$
derivative	$\frac{\partial x_i}{\partial X_j} = A_{ij}$	$\frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \mathbf{A}$
	$\frac{\partial X_i}{\partial X_j} = \delta_{ij}$	$\frac{\partial \mathbf{X}}{\partial \mathbf{X}} = \mathbf{I}$
	$\frac{\partial X_{ij}}{\partial X_{kl}} = \delta_{ik}\delta_{jl} = C_{ikjl}$	$\mathbf{X} \otimes \frac{\partial}{\partial \mathbf{X}} = \mathbb{C}$
kronicker-delta	$\delta_{kl}\delta_{lj} = \delta_{kj}$	$\mathbf{I} \cdot \mathbf{I} = \mathbf{I}$
	$\delta_{ij}\delta_{kl} = \delta_{ijkl}$	$\mathbf{I} \otimes \mathbf{I}$
	$\delta_{ik}\delta_{jl}$	$\mathbf{I}$

Table 1: Tensor and index notation

$$\begin{aligned}
\mathbf{A} : \mathbf{B} &= \mathbf{B} : \mathbf{A} \\
\mathbf{A} : \mathbf{B} &= \mathbf{A}^T : \mathbf{B}^T \\
(\mathbf{A} \cdot \mathbf{B})^T &= \mathbf{B}^T \cdot \mathbf{A}^T \\
(\mathbf{A} \cdot \mathbf{B}) : \mathbb{C} &= (\mathbb{C}^T \cdot \mathbf{A}) : \mathbf{B}^T \\
(\mathbf{A} \cdot \mathbf{B}) : \mathbb{C} &= \mathbf{B} : (\mathbf{A}^T \cdot \mathbb{C}) \\
\mathbf{a} \otimes \mathbf{b} &= (\mathbf{b} \otimes \mathbf{a})^T \\
\mathbf{a} \otimes \mathbf{b} &= \mathbf{a} \cdot (\mathbf{I} \otimes \mathbf{b}) \\
\mathbf{A} : (\mathbf{b} \otimes \mathbb{C}) &= \mathbf{b} \cdot (\mathbf{A} : (\mathbf{I} \otimes \mathbb{C}))
\end{aligned}$$

Table 2: Basic tensor computations.

# Introduction

---

Interest in mechanics based approaches for pavement engineering design has recently grown considerably, both nationally and internationally. A typical example of this change in design philosophy is the recent Empirical-Mechanistic Design Guide on pavement design, in which the finite element method in combination with advanced material constitutive models for all layers and characterization techniques constitute the backbone of the whole design process.

By accounting for the idiosyncrasies of pavement material response and by enabling the visualization of the internal distributions of stresses and strains in the body of a pavement, the finite element method constitutes a valuable tool in understanding the mechanisms and the processes leading to pavement deterioration. In addition, the method enables the quantification of the interaction between the material and the geometric characteristics of a pavement.

Unfortunately, because of the dependence of pavement engineering materials on the state of stress, on the rate of loading and on the temperature, they constitute some of the most difficult and computational intensive materials for finite element simulation. In addition, in pavement engineering, very few, if any, realistic situations can be encountered which are truly three-dimensional.

Unfortunately the use of three-dimensional Finite Element models is both, time and resource consuming. Especially so if the non-linear nature of the materials and the processes concerned is considered. Nevertheless, when such models become available, utilization of the method can result to significant time and financial savings in laboratory and field-testing.

Since the early 90s, the group of Mechanics of Structural Systems of the Faculty of Civil Engineering and Geosciences of TU Delft has been closely cooperating with other national and international teams towards the development of tools and procedures capable of addressing realistically the response characteristics of a wide range of pavement engineering materials.

In the framework of this cooperation, CAPA-3D has been developed as a finite element based platform to serve the computational needs of the pavement research community. Over the years, it has evolved into a fully fledged finite element system for static or dynamic analysis of very large scale three dimensional pavement and soil engineering models. It consists of a sophisticated user interface, a powerful band-optimizing mesh generator, high quality user controlled graphical output, several material and element types, and a variety of specialized algorithms for the more efficient analysis of pavement constructions. Among others, these include a moving load simulation algorithm and a contact algorithm.

Ever since its inception in the early 90s, the system has been under continuous update and development. Invariably developments were dictated by the research

needs of the progressively growing groups of researchers/users. In the development of the CAPA-3D system the need for powerful, albeit expensive, computational facilities has been addressed temporarily by segmenting the system into subsystems. However, at the core of the CAPA-3D systems lies the algorithm for updating the external and internal forces acting on and within materials respectively. For simulations requiring very large numbers of degrees of freedom and loading cycles more sophisticated algorithms will have to be developed as memory size and CPU power limits the scale of the simulations that can be handled.

Modern iterative and direct solvers are capable of handling large systems of equations resulting from 3D models which was unthinkable ten years ago. Still, even with the power of CPUs increasing every year and the introduction of multiple computing cores on one CPU, the demand for efficient, parallel computing algorithms is higher. Well proven techniques like the Krylov subspace methods, multigrid and direct solution methods are mashed up in all forms of hybrid (iterative) solvers. Due to the overwhelming amount of open source software, solvers have become available for every engineer with a modern desktop computer. Unfortunately, as mixing different medicine to cure an unknown disease isn't likely to work, mashing up numerical methods is also not a guarantee for an efficient, fast and moreover robust solver. The particularities of each numerical method and the way they influence one another is often not well understood. Extensive analysis of the underlying physical phenomena, the discretization and meshing techniques lie at the base of a succesful solving algorithm. A good recipe can only be written if it is understood what, and especially, how to cure.



# Basics structural mechanics

---

What is structural mechanics all about? Most of all, structural mechanics is about deformation. How does a body of material, or like in most cases, a combination of different materials respond to an externally applied load. Natural responses are compression, dilatation and also cracking. However, the rate of deformation is probably less important than the reason of deformation. Why is the material showing cracks after a period of constant loading? Where has the material to be reinforced to avoid cracking or too much compression.

It will become clear that the underlying physical processes of these models are all based on the balance of forces. When an external force is applied, the material will respond with an internal stress evidently. An important material property is stiffness, the relation between strain and stress. Stiffness can be seen as the resistance of a material to compression or stretching. For instance, steel is a very stiff material. Bending a plate of steel of significant thickness at room temperature requires an enormous load (force). In contrary to the compression of a soft rubber ball, which needs hardly any force at all.

In this chapter a mathematical framework will be introduced to describe the quantities stress and strain. The balance of forces and accompanying mechanical laws will be introduced and a brief overview on the most important material properties, elasticity, plasticity and viscosity is provided.

## 1.1 Continuum model

Imagine a body of material,  $V$  and from a certain moment in time a distributed force  $F_{ext}$  is applied at the top of the material.

Compare a current configuration of the system at a certain point in time with the reference configuration of the system at time zero. The following quantities are introduced:

$$\begin{aligned} \mathbf{X} &, \text{ position vector in reference configuration} \\ \mathbf{x} &, \text{ position vector in current configuration} \end{aligned} \tag{1.1}$$

They relate as:

$$d\mathbf{x} = \mathbf{F}d\mathbf{X} \tag{1.2}$$

in which  $F$  is known as the deformation gradient tensor and is defined by

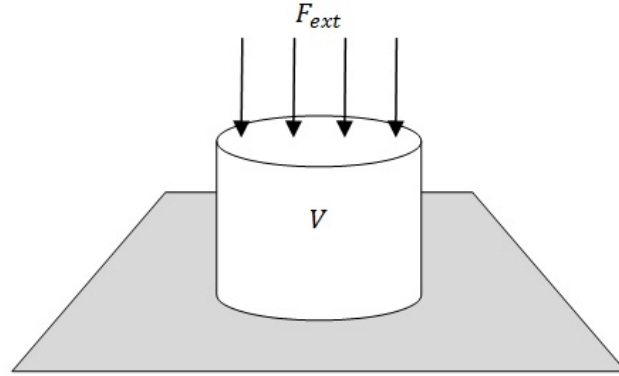


Figure 1.1: Schematic representation of test situation

$$\begin{aligned} \mathbf{F} &= \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \\ &= \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} & \frac{\partial x_3}{\partial X_3} \end{bmatrix} \end{aligned} \quad (1.3)$$

The displacement between position vectors  $\mathbf{x}$  and  $\mathbf{X}$  can be expressed as  $\mathbf{u}$ :

$$\mathbf{x} = \mathbf{X} + \mathbf{u} \quad (1.4)$$

$$d\mathbf{x} = d\mathbf{X} + d\mathbf{u} \quad (1.5)$$

### 1.1.1 Strain

Write the deformation gradient tensor as

$$\mathbf{F} = \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}} \quad (1.6)$$

or in index notation

$$F_{ij} = \delta_{ij} + \frac{\partial u_i}{\partial X_j} \quad (1.7)$$

Hence the right Cauchy-Green deformation tensor can be introduced.

$$\begin{aligned} \mathbf{C} &= \mathbf{F}^T \mathbf{F} \\ &= \left( \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}} \right)^T \left( \mathbf{I} + \frac{\partial \mathbf{u}}{\partial \mathbf{X}} \right) \\ &= \mathbf{I} + \left( \frac{\partial \mathbf{u}}{\partial \mathbf{X}} \right)^T \left( \frac{\partial \mathbf{u}}{\partial \mathbf{X}} \right) + \left( \frac{\partial \mathbf{u}}{\partial \mathbf{X}} \right)^T + \left( \frac{\partial \mathbf{u}}{\partial \mathbf{X}} \right) \end{aligned} \quad (1.8)$$

In addition to the right Cauchy-Green deformation tensor introduce the Lagrangian-Green strain tensor  $\mathbf{E}$ ,

$$\mathbf{E} = \frac{1}{2} (\mathbf{C} - \mathbf{I}) \quad (1.9)$$

or in index notation

$$E_{ij} = \frac{1}{2} (F_{ki}F_{kj} - \delta_{ij}), \quad i, j \in \{1, 2, 3\} \quad (1.10)$$

The main diagonals of 2nd order tensor  $\mathbf{E}$  in terms of the displacements,

$$E_{ii} = \frac{\partial u_i}{\partial X_i} + \frac{1}{2} \left[ \left( \frac{\partial u_1}{\partial X_1} \right)^2 + \left( \frac{\partial u_2}{\partial X_2} \right)^2 + \left( \frac{\partial u_3}{\partial X_3} \right)^2 \right] \quad (1.11)$$

and the off diagonals,

$$E_{ij} = \frac{1}{2} \left[ \frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i} + \frac{\partial u_1}{\partial X_i} \frac{\partial u_1}{\partial X_j} + \frac{\partial u_2}{\partial X_i} \frac{\partial u_2}{\partial X_j} + \frac{\partial u_3}{\partial X_i} \frac{\partial u_3}{\partial X_j} \right], \quad i \neq j \quad (1.12)$$

Here only large strain deformations are considered.

### 1.1.2 Stress

Forces applied to a surface area of the material are expressed as pressure and have the derived quantity of Pascal ( $Pa = \frac{N}{m^2}$ ). The forces per unit area are called stress.

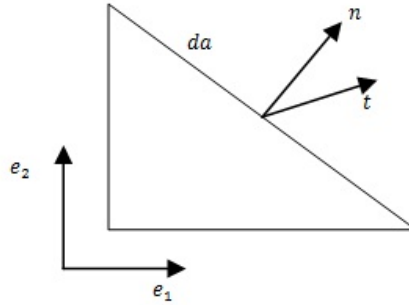


Figure 1.2: Traction  $t$  acting on an infinitesimal element

In figure 1.2 the traction force  $t$  acting on an infinitesimal element  $da$  has been illustrated. Express  $t$  as

$$\mathbf{t} = \lim_{da \rightarrow 0} \frac{d\mathbf{q}}{da} \quad (1.13)$$

where  $d\mathbf{q}$  is the infinitesimal force (N). Now introduce the Cauchy stress tensor  $\sigma$  [Pa] in the current configuration

$$\mathbf{t} = \boldsymbol{\sigma} \cdot \mathbf{n} \quad (1.14)$$

where  $\mathbf{n}$  is the normal vector on element  $da$ . The Cauchy stress is referred to the current, yet unknown configuration, hence the use of a stress measure that refers to the reference configuration makes more sense. So introduce the stress states in the reference configuration which result from a pullback operation on the Cauchy stress tensor to the reference configuration. The First Piola-Kirchhoff stress tensor  $\mathbf{P}$ ,

$$\mathbf{T} = \mathbf{P} \cdot \mathbf{N} \quad (1.15)$$

where  $\mathbf{T}$  is the traction vector with respect to the reference configuration. The stress tensor  $\mathbf{P}$  relates to the Cauchy stress tensor  $\boldsymbol{\sigma}$  as

$$\mathbf{P} = J \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-T} \quad (1.16)$$

here  $J$  is the Jacobian for transformation between the current and the reference configuration. Because the stress tensor  $\mathbf{P}$  can lead to ill conditioned asymmetric systems, introduce the symmetric Second Piola-Kirchhoff stress tensor  $\mathbf{S}$

$$\mathbf{S} = J \cdot \mathbf{F}^{-1} \cdot \boldsymbol{\sigma} \cdot \mathbf{F}^{-T} \quad (1.17)$$

It is emphasized that the Piola-Kirchhoff stress tensors have no direct physical interpretation but are necessary to compute the Cauchy stress. The first and second Piola-Kirchhoff stresses relate as  $\mathbf{P} = \mathbf{F}\mathbf{S}$ .

## 1.2 Equilibrium equation: balance of forces

First of all the laws a material has to obey are introduced. At a fixed moment in time all forces exerted on a body and the reaction forces within the body must be in balance. This is the balance of forces which is a direct result of the balance of momentum. This rule can be expressed as follows (strong formulation):

$$\int_v \mathbf{div}(\boldsymbol{\sigma}) + \mathbf{f} - \rho \mathbf{g} dv = 0 \quad (1.18)$$

where  $\boldsymbol{\sigma}$  represents the forces acting on the body,  $\mathbf{f}$  are body forces which can be considered as source terms and  $\rho \mathbf{g}$  is the gravitational force with density  $\rho$ .

When the system is not (yet) in balance, i.e. external and internal forces are different, this is expressed with the residual equation,

$$\mathbf{r} = \mathbf{div}(\boldsymbol{\sigma}) + \mathbf{f} - \rho \mathbf{g} \quad (1.19)$$

In order to get a balance of forces it makes sense that the residual equation is to be minimized. In other words, the residual equation must be zero to obtain force balance.

Now assume to be at a fixed moment in time. Instead of using the residual equation it is more convenient to use the energy these forces generate when applied to a virtual displacement  $\delta \mathbf{u}$ . This energy is the virtual work,

$$[J] = [N][m] \hat{=} \mathbf{r} \cdot \delta \mathbf{u} \quad (1.20)$$

When using a virtual velocity instead of the virtual displacement and integrating over the whole body, introduce the virtual work per unit volume per unit time,

$$\delta W = \int_v \mathbf{r} \cdot \delta \mathbf{u} dv \quad (1.21)$$

Back to the more general formulation, the virtual work in the current configuration for an arbitrary virtual displacement is,

$$\delta W(\mathbf{x}) = - \int_v \boldsymbol{\sigma} : \delta \dot{\mathbf{d}} dv + \int_v \mathbf{f} \cdot \delta \mathbf{v} dv + \int_a \mathbf{t} \cdot \delta \mathbf{v} da - \int_v \rho \mathbf{g} \cdot \delta \mathbf{v} dv \quad (1.22)$$

And for the reference configuration,

$$\delta W(\mathbf{X}) = - \int_V \mathbf{P} : \delta \dot{\mathbf{F}} dV + \int_V \mathbf{f}_0 \cdot \delta \mathbf{v} dV + \int_A \mathbf{t}_0 \cdot \delta \mathbf{v} dA - \int_V \rho_0 \mathbf{g} \cdot \delta \mathbf{v} dV \quad (1.23)$$

Discriminate the virtual work induced by internal and external forces,

$$\delta W_{int}(\mathbf{X}) = \int_V \mathbf{P} : \delta \dot{\mathbf{F}} dV \quad (1.24)$$

$$\delta W_{ext}(\mathbf{X}) = \int_V \mathbf{f}_0 \cdot \delta \mathbf{v} dV + \int_A \mathbf{t}_0 \cdot \delta \mathbf{v} dA - \int_V \rho_0 \mathbf{g} \cdot \delta \mathbf{v} dV \quad (1.25)$$

where  $\delta \dot{\mathbf{F}}$  is the virtual deformation rate. Because the external forces do not vary in time (or load step) rewrite the external work as

$$\delta W_{ext}(\mathbf{X}) = \delta \mathbf{v} \cdot \mathbf{f}_{ext} \quad (1.26)$$

with

$$\mathbf{f}_{ext} = \int_V \mathbf{f}_0 dV + \int_A \mathbf{t}_0 dA - \int_V \rho_0 \mathbf{g} dV. \quad (1.27)$$

And of course at equilibrium,

$$\delta W(\mathbf{X}) = \delta W_{int}(\mathbf{X}) - \delta W_{ext}(\mathbf{X}) = 0 \quad (1.28)$$

Due to the response and the geometry of the material the virtual work equilibrium equation becomes non-linear. A Newton-Raphson iteration can be used to solve the non-linear system of equations. Keep in mind that at a fixed point in time the balance of forces must be satisfied between two time steps. An expression for the balance of forces in terms of virtual work together with Newton-Raphson iterations will result into the balance the forces algorithm.

### 1.3 Balancing forces

At equilibrium the virtual work must equal zero as shown by equation 1.28. This is equation becomes non-linear for hyper-elastic, plastic and viscous materials. Therefore equation 1.28 must be linearized in order to solve it. First introduce the derivative on an arbitrary function  $g$  in the direction of a vector  $\Delta \mathbf{u}$ ,

$$D_{\Delta \mathbf{u}} [g(\mathbf{x})] = \lim_{\varepsilon \rightarrow 0} \left( \frac{\partial g(\mathbf{x} + \varepsilon \Delta \mathbf{u})}{\partial \varepsilon} \right) \quad (1.29)$$

The directional derivative  $D_{\Delta \mathbf{u}}$  complies to the following rules of differentiation where  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{X}$  are arbitrary tensors,

$$D_{\Delta \mathbf{u}} [\mathbf{A} : \mathbf{B}] = D_{\Delta \mathbf{u}} [\mathbf{A}] : \mathbf{B} + \mathbf{A} : D_{\Delta \mathbf{u}} [\mathbf{B}] \quad (1.30)$$

$$D_{\Delta \mathbf{u}} [\mathbf{A}] = \frac{\partial \mathbf{A}}{\partial \mathbf{X}} : D_{\Delta \mathbf{u}} [\mathbf{X}] \quad (1.31)$$

Assume that at a certain moment in time and space,  $\delta W(\mathbf{X}_0) > 0$  where  $\mathbf{X}_0$  is the spatial vector. Also assume that with a step in the direction of  $\Delta \mathbf{u}$  the equilibrium has been reached, i.e.  $\delta W(\mathbf{X}_0 + \Delta \mathbf{u}) = 0$ . Hence linearize the virtual work equation around  $\mathbf{X}_0$  in the direction of  $\Delta \mathbf{u}$ ,

$$\delta W \doteq \delta W(\mathbf{X}_0) + D_{\Delta \mathbf{u}} [\delta W(\mathbf{X}_0)] \quad (1.32)$$

Of course the linearized virtual work equation must equal to zero at the equilibrium also,

$$\delta W(\mathbf{X}_0) + D_{\Delta \mathbf{u}} [\delta W(\mathbf{X}_0)] = 0. \quad (1.33)$$

Write equation 1.33 as,

$$\delta W_{int}(\mathbf{X}_0) - \delta W_{ext}(\mathbf{X}_0) + D_{\Delta \mathbf{u}} [\delta W_{int}(\mathbf{X}_0)] - D_{\Delta \mathbf{u}} [\delta W_{ext}(\mathbf{X}_0)] = 0 \quad (1.34)$$

where  $\delta W_{int}$ ,  $\delta W_{ext}$  as defined in 1.24 and

$$\begin{aligned} D_{\Delta \mathbf{u}} [\delta W_{int}(\mathbf{X}_0)] &= \int_V D_{\Delta \mathbf{u}} [\mathbf{P}] : \delta \dot{\mathbf{F}} dV + \int_V \mathbf{P} : D_{\Delta \mathbf{u}} [\delta \dot{\mathbf{F}}] dV \\ D_{\Delta \mathbf{u}} [\delta W_{ext}(\mathbf{X}_0)] &= 0 \end{aligned} \quad (1.35)$$

Expand the directional derivative of the internal work for the two integrals separately. First the directional derivative of the first Piola-Kirchoff,

$$D_{\Delta \mathbf{u}} [\mathbf{P}] = \frac{\partial \mathbf{P}}{\partial \mathbf{F}} : D_{\Delta \mathbf{u}} [\mathbf{F}] \quad (1.36)$$

with,

$$D_{\Delta \mathbf{u}} [\mathbf{F}] = D_{\Delta \mathbf{u}} \left[ \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \right] = \lim_{\varepsilon \rightarrow 0} \frac{\partial}{\partial \mathbf{X}} \left( \frac{\partial \mathbf{x} + \varepsilon \Delta \mathbf{u}}{\partial \varepsilon} \right) = \frac{\partial \Delta \mathbf{u}}{\partial \mathbf{X}} = \nabla_0 \Delta \mathbf{u} \quad (1.37)$$

and in [10] it is shown that,

$$\frac{\partial \mathbf{P}}{\partial \mathbf{F}} = \mathbf{I} \otimes \mathbf{S} + \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T. \quad (1.38)$$

where  $\mathbb{C}$  is the fourth order elasticity tensor and is defined as  $\mathbb{C} = \frac{\partial \mathbf{S}}{\partial \mathbf{E}}$ .

The next term in the first integral of equation 1.35 that has to be expanded is the virtual deformation rate  $\delta \dot{\mathbf{F}}$ . It can be expressed as,

$$\delta \dot{\mathbf{F}} = D_{\delta \mathbf{v}} \left[ \frac{\partial \mathbf{v}}{\partial \mathbf{X}} \right] = \lim_{\varepsilon \rightarrow 0} \frac{\partial}{\partial \mathbf{X}} \left( \frac{\partial \mathbf{v} + \varepsilon \delta \mathbf{v}}{\partial \varepsilon} \right) = \frac{\partial \delta \mathbf{v}}{\partial \mathbf{X}} = \nabla_0 \delta \mathbf{v} \quad (1.39)$$

The second integral of equation 1.35 contains the directional derivative of the virtual deformation rate  $\delta \dot{\mathbf{F}}$  which is zero by definition,

$$D_{\Delta \mathbf{u}} [\delta \dot{\mathbf{F}}] = D_{\Delta \mathbf{u}} \left[ \frac{\partial \delta \mathbf{v}}{\partial \mathbf{X}} \right] = 0 \quad (1.40)$$

Substitution of the expressions for  $D_{\Delta \mathbf{u}} [\mathbf{P}]$ ,  $\delta \dot{\mathbf{F}}$  and  $D_{\Delta \mathbf{u}} [\delta \dot{\mathbf{F}}]$  into equation 1.35 yields,

$$\begin{aligned} D_{\Delta \mathbf{u}} [\delta W_{int}(\mathbf{X}_0)] &= \int_V (\mathbf{I} \otimes \mathbf{S} + \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T) : \nabla_0 \Delta \mathbf{u} : \nabla_0 \delta \mathbf{v} dV + 0 \\ &= \int_V ((\nabla_0 \Delta \mathbf{u} : (\mathbf{I} \otimes \mathbf{S})) + (\nabla_0 \Delta \mathbf{u} : \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T)) : \nabla_0 \delta \mathbf{v} dV \\ &= \int_V (\nabla_0 \Delta \mathbf{u} \cdot \mathbf{S}) : \nabla_0 \delta \mathbf{v} dV + \\ &\quad \int_V (\nabla_0 \Delta \mathbf{u} : \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T) : \nabla_0 \delta \mathbf{v} dV \end{aligned} \quad (1.41)$$

Hence, after substitution of an expression for  $\delta \dot{\mathbf{F}}$  and rearranging terms the linearized virtual work equation at equilibrium is,

$$\begin{aligned} \int_V (\nabla_0 \Delta \mathbf{u} \cdot \mathbf{S}) : \nabla_0 \delta \mathbf{v} dV &+ \int_V (\nabla_0 \Delta \mathbf{u} : \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T) : \nabla_0 \delta \mathbf{v} dV \\ &= \delta \mathbf{v} \cdot \mathbf{f}_{ext} - \int_V \mathbf{P} : \nabla_0 \delta \mathbf{v} dV \end{aligned} \quad (1.42)$$

## 1.4 Material response

Now that a model is obtained to balance the forces within a fixed time step, a model for the response of the material to external loads is necessary. In general three different material response models are distinguished,

- Elasticity
- Plasticity
- Viscosity

In real-life simulations the bodies will often consist of different types of material and hence a combination of the three models, elasto-visco-plasticity, is used.

In the next sections a outline on the formal definition of elasticity, plasticity and viscosity is given.

### 1.4.1 Elasticity

The effect of elasticity is illustrated best with a simplified version of a purely elastic material. Hence, imagine a one dimensional spring attached to two moveable boundaries at initial positions  $x_0$  and  $x_1$ .



Figure 1.3: Simplified representation of elastic material.

When an external pressure  $\sigma$  is applied to the boundary at position  $x_1$  the, spring stretches to a new boundary at position  $x_2$ .

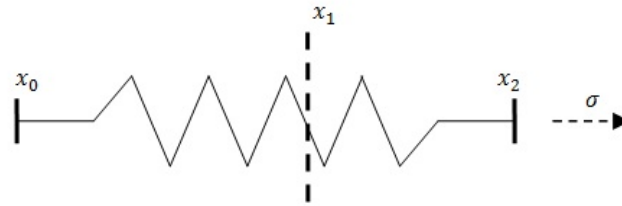


Figure 1.4: Simplified representation of elastic material in stretched state.

The difference between  $x_2$  and  $x_1$  is the strain  $\varepsilon$  of the spring. When the relation between the external pressure  $\sigma$  and the strain  $\varepsilon$  is a linear function in time the spring (i.e. material) shows elastic behavior. However, this linear behavior must be valid for both the loading and unloading phase. In other words, the material must return to its original state. In this case, during unloading, the boundaries of the spring must return to positions  $x_0$  and  $x_1$  respectively.



### 1.4.2 Plasticity

The principle of plasticity is based on the fact that a material can have memory properties. This is best explained by a small mind experiment. Imagine a cube of elastic material. When the two sides of this cube are pulled apart the material will stretch, i.e. deform. When the applied force is not too large the material will regain its original shape after unloading. This effect is elasticity as described in the previous section. However, after a certain threshold the applied force becomes too large and the material will yield. During unloading the material has been deformed permanently. For instance, the body can show an increase of volume. Together with the law of conservation of mass this implies that the density of the body has changed. This effect can be illustrated by figure 1.5.

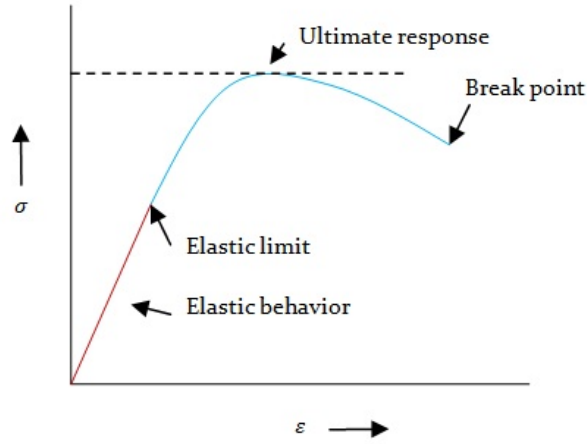


Figure 1.5: Example of relation between strain and stress.

There is a linear (elastic) relation between the strain  $\varepsilon$  and stress  $\sigma$  for small stresses. After a certain point, the elastic limit or yield point, plasticity can be observed. The strain-stress relation is no longer linear and when stretched too much, the material will eventually crack (break point). Plasticity has two phases, the hardening and softening phase. The hardening phase is spanning the range from zero stress to the ultimate response. Clearly, the softening phase represents the spanning from the ultimate response to the break point.

When utilizing plasticity within the balancing of forces, the admissible forces have to be predefined so it is known after which point plastic behavior is observed. Hence, introduce the plastic response surface, a domain of admissible stresses which is illustrated in figure 1.6. Suppose that a displacement of the body  $\mathbf{u}$  has been computed. The plastic response surface is a function of time, stress and displacement. This means that the surface will move in time/iterations.

For uniaxial compression tests as illustrated in figure 1.1 the stress path is depicted by the dotted line in figure 1.6. Here  $I_1$  is the first invariant and represents the normal stress components,  $\sigma_{xx}, \sigma_{yy}$  and  $\sigma_{zz}$ . These are also known as the vol-

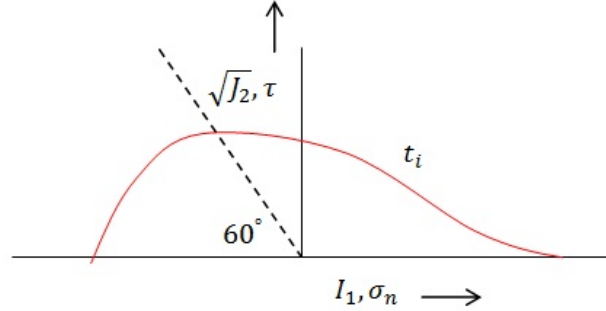


Figure 1.6: Example of plastic yield surface.

umetric stress components. Also,  $\sqrt{J_2}$  is the second invariant and represents the relation between shear stress components,  $\sigma_{xy}$ ,  $\sigma_{xz}$  and  $\sigma_{yz}$  and the normal stress components,  $\sigma_{xx}$ ,  $\sigma_{yy}$  and  $\sigma_{zz}$ .

These are also known as the deviatoric stress components. From figure 1.6 it is apparent that for an uniaxial test the ratio between the shear and normal stress is constant. In this particular case the ratio is equivalent to an angle of  $60^\circ$ .

The total stress, that consists of a normal and shear component, will show elastic behavior when it is still in the domain of admissible stresses. This phase corresponds to the elastic response curve of figure 1.5. However, when the stress reaches the plastic response curve the material will start to building up plasticity. This is the yield point. At first hardening of the material takes place until the point of ultimate plastic response has been reached. From this point the material will no longer harden but it will soften. A more physical interpretation is that the material starts to show micro cracks in its internal structure. When too much pressure is applied the material will eventually crack.

Each phase is characterized by its own hardening and softening parameters that are unique to each material. Note that the stress-strain relation illustrated in figure 1.5 comes from a return mapping procedure that utilizes the plastic response surface. In preemt to Section 1.5 it is easily understood how this surface relates to the stress-strain relation of figure 1.5. As mentioned before, after the yielding point plasticity is being built up. In the return mapping procedure elastic behavior of the material is assumed and the corresponding plastic behavior is back calculated.

### 1.4.3 Viscosity

The phenomena of viscosity can best be described by a small example. When in calm water it still takes effort to paddle a rowboat. This is because of the viscous effects, or in other words, internal friction in a fluid. All visous fluid tend to cling to a solid surface in contact with it. Consider the situation of figure 1.7. A viscous fluid is flowing between two plates. The upper plate moves with constant velocity  $v$  and the lower plate is fixed. Because of the visosity the fluid in contact with each surface

has the same velocity as the surface. Hence, the fluid is moving with velocity  $v$  near the upper plate and is stationary near the lower plate. Now consider the portion of fluid in the area  $abcd$ . of figure 1.7. After a certain moment of time the portion of fluid will transform into shape  $abc'd'$  and a moment later the portion of fluid will become even more distorted. Moreover, the fluid is in a state of continuously increasing shear strain. The shear strain is defined as the ratio of the displacement  $dd'$  to the length of the flow  $l$ . To keep the upper plate moving with velocity  $v$  apply force  $F$  at its right boundary. To keep the lower plate stationary and apply the same force  $F$  in the opposite direction at its left boundary. If  $A$  is the surface area between the two plates, the ratio  $\frac{F}{A}$  represents the shear stress exerted on the fluid.

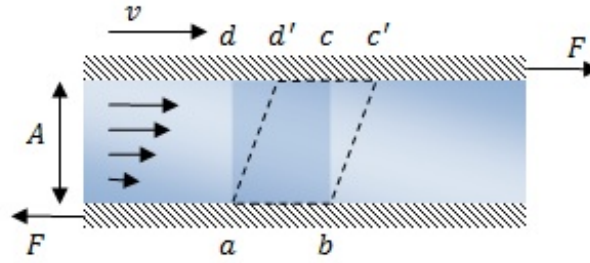


Figure 1.7: Simplified example of viscous laminar flow..

Define the viscosity  $\eta$  of a fluid as the ratio of the shear stress to the change of shear strain.

$$\eta = \frac{F/A}{\Delta dd'/l} = \frac{F/A}{v/l} \quad (1.43)$$

The viscosity of a material strongly depends on the temperature. For higher temperatures materials tend to be more viscous. We could think of asphaltic materials during hot summer days. The upper layer of the asphalt asborbs much sunlight (heat) and the temperature wihtin the material rises quickly. Hence, the asphalt becomes more viscous and more fluid like. The same effect can be seen by applying heavy forces to a material. Due to pressure the material will become more viscous and will soften.

Analogue to plasticity, viscosity is being build up within the material. In Section 1.5 the algorithms for calculcting the deformation corresponding to the viscous properties will appear to be similar to the elasto-plastic material response algorithms.

## 1.5 Implementation of material response

How do the material models relate to the balance of forces of section 1.3. A mathematical framework to implement elasto-visco-plasticity is constructed.

The balance of forces and the material models can be linked with the physical effect of ‘dissipation of energy’. The dissipation of energy will provide the constitutive relations for the material models.

### 1.5.1 Dissipation of energy

The response of the material to the external forces applied is being expressed in the energy-dissipation equation. True dissipation of energy is only valid for inelastic systems. Because of this inelastic behavior (plasticity, viscosity), energy (heat) is dissipated all over the system when the material responds to the applied forces. In other words, when forces are being applied to the system, mechanical processes within the material are initiated. For purely elastic materials these processes are reversible. Here the stress is only a function of the deformation (and temperature), and the system will return to its original state during unloading. However, for plastic and viscous materials the stress becomes a function of deformation and variables associated with the memory properties of the material. From a certain point in time (yielding point), with endured loading, the mechanical processes are irreversible. For instance, when plasticity applies, the system will experience permanent deformation.

This loss of energy (e.g. heat) has been captured in the Clausius-Planck inequality. At any point in the system and at all times the internal dissipation  $\mathcal{D}$  should be non-negative.

$$\mathcal{D} = \mathbf{P} : \dot{\mathbf{F}} - \dot{\Psi} \geq 0 \quad (1.44)$$

Where  $\mathbf{P} : \dot{\mathbf{F}}$  represents the work per unit volume and  $\Psi$  is known as the Helmholtz free-energy function or, when solely a function of the deformation gradient  $\mathbf{F}$ , the strain energy function. The Helmholtz free energy function is a potential, i.e. (virtual) work per unit volume.

### 1.5.2 Multiplicative decomposition

When we combine the three material models it is necessary to extend our current framework of the sole deformation gradient. We need to measure and compute the effects of elasticity, plasticity and viscosity separately. Hence, we introduce the multiplicative decomposition of the deformation gradient.

Again we have the following relation between the current and undeformed configuration,

$$d\mathbf{x} = \mathbf{F} \cdot d\mathbf{X} \quad (1.45)$$

As shown in figure 1.8 the deformation gradient of a material in which the elastoplastic and viscoelastic components act in parallel can be decomposed as,

$$\mathbf{F} = \mathbf{F}_\infty \cdot \mathbf{F}_p \quad (1.46)$$

$$\mathbf{F} = \mathbf{F}_e \cdot \mathbf{F}_v \quad (1.47)$$

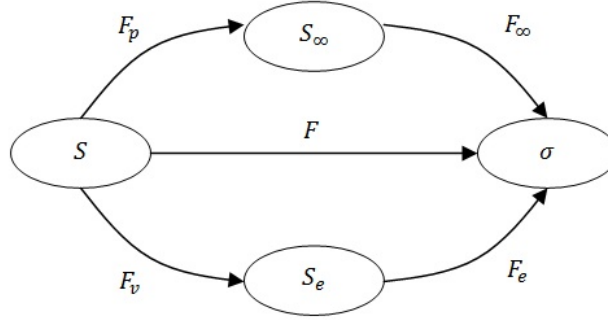


Figure 1.8: Schematic representation of multiplicative decomposition.

In which,

$\mathbf{F}_\infty$  = the elastic component of the deformation gradient of the elastoplastic element

$\mathbf{F}_p$  = the plastic component of the deformation gradient of the elastoplastic element

$\mathbf{F}_e$  = the elastic component of the deformation gradient of the viscoelastic element

$\mathbf{F}_v$  = the viscous component of the deformation gradient of the viscoelastic element

Furthermore the following definitions hold for plasticity,

$$\mathbb{C}_\infty = \mathbf{F}_\infty^T \cdot \mathbf{F}_\infty \quad (1.48)$$

$$\mathbb{C}_p = \mathbf{F}_p^T \cdot \mathbf{F}_p \quad (1.49)$$

and also viscosity,

$$\mathbb{C}_e = \mathbf{F}_e^T \cdot \mathbf{F}_e \quad (1.50)$$

$$\mathbb{C}_v = \mathbf{F}_v^T \cdot \mathbf{F}_v \quad (1.51)$$

Therefore,

$$\mathbb{C} = \mathbf{F}^T \cdot \mathbf{F} \quad (1.52)$$

$$= \mathbf{F}_p^T \cdot \mathbb{C}_\infty \cdot \mathbf{F}_p \quad (1.53)$$

$$= \mathbf{F}_v^T \cdot \mathbb{C}_e \cdot \mathbf{F}_v \quad (1.54)$$

The relation between the Cauchy stress  $\sigma$  and the second Piola-Kirchhoff  $\mathbf{S}$  for plasticity,

$$J_\infty^{-1} \cdot \mathbf{F}_\infty \cdot \mathbf{S}_\infty \cdot \mathbf{F}_\infty^T = \sigma = J^{-1} \cdot \mathbf{F} \cdot \mathbf{S} \cdot \mathbf{F}^T \quad (1.55)$$

and also viscosity,

$$J_e^{-1} \cdot \mathbf{F}_e \cdot \mathbf{S}_e \cdot \mathbf{F}_e^T = \sigma = J^{-1} \cdot \mathbf{F} \cdot \mathbf{S} \cdot \mathbf{F}^T \quad (1.56)$$

hence it is sufficient to compute just the values of  $\mathbf{S}_\infty$  and  $\mathbf{F}_\infty$  to compute the value of  $\mathbf{S}$ .

### 1.5.3 Generalized model local dissipation

The Helmholtz free energy function for a elasto-visco-plastic material model can be expressed as,

$$\Psi = \Psi_v(\mathbb{C}_e) + \Psi_p(\mathbb{C}_\infty, \xi_p) \quad (1.57)$$

Here  $\xi_p$  is a measure of the plastic deformation. The Clausius-Planck inequality of 1.44 leads to,

$$\mathbf{S} : \frac{1}{2} \dot{\mathbf{C}} - \left[ \frac{\partial \Psi_p}{\partial \mathbb{C}_\infty} : \dot{\mathbf{C}}_\infty + \frac{\partial \Psi_p}{\partial \xi_p} \cdot \dot{\xi}_p \right] - \left[ \frac{\partial \Psi_v}{\partial \mathbb{C}_e} : \dot{\mathbf{C}}_e \right] \geq 0 \quad (1.58)$$

It has been shown in [10] that inequality 1.58 can be reformulated as,

$$\left[ \mathbf{S} - 2\mathbf{F}_v^{-1} \cdot \frac{\partial \Psi_v}{\partial \mathbb{C}_e} \cdot \mathbf{F}_v^{-T} - 2\mathbf{F}_p^{-1} \cdot \frac{\partial \Psi_p}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_p^{-T} \right] : \frac{1}{2} \dot{\mathbf{C}} \quad (1.59)$$

$$+ \left[ 2\mathbf{F}_\infty \cdot \frac{\partial \Psi_p}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_\infty^T \cdot \mathbf{F}_\infty^{-T} : \mathbf{F}_\infty \cdot l_p - \frac{\partial \Psi_p}{\partial \xi_p} \cdot \dot{\xi}_p \right] \quad (1.60)$$

$$+ \left[ 2\mathbf{F}_e \cdot \frac{\partial \Psi_v}{\partial \mathbb{C}_e} \cdot \mathbf{F}_e^T \cdot \mathbf{F}_e^{-T} : \mathbf{F}_e \cdot l_v \right] \geq 0 \quad (1.61)$$

By standard arguments the stress tensor  $\mathbf{S}$  can be additively decomposed into a viscoelastic  $\mathbf{S}_v$  and a plastic component  $\mathbf{S}_p$ ,

$$\mathbf{S} = 2\mathbf{F}_v^{-1} \cdot \frac{\partial \Psi_v}{\partial \mathbb{C}_e} \cdot \mathbf{F}_v^{-T} + 2\mathbf{F}_p^{-1} \cdot \frac{\partial \Psi_p}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_p^{-T} \quad (1.62)$$

$$= \mathbf{S}_v + \mathbf{S}_p \quad (1.63)$$

And hence the following inequalities (constitutive relations) are obtained

$$\left[ 2\mathbf{F}_\infty \cdot \frac{\partial \Psi_p}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_\infty^T \cdot \mathbf{F}_\infty^{-T} : \mathbf{F}_\infty \cdot l_p - \frac{\partial \Psi_p}{\partial \xi_p} \cdot \dot{\xi}_p \right] \geq 0 \quad (1.64)$$

$$\left[ 2\mathbf{F}_e \cdot \frac{\partial \Psi_v}{\partial \mathbb{C}_e} \cdot \mathbf{F}_e^T \cdot \mathbf{F}_e^{-T} : \mathbf{F}_e \cdot l_v \right] \geq 0 \quad (1.65)$$

### 1.5.4 Plastic response

In [8] an algorithm is given for computing the plastic response. In the intermediate configuration, for the elastoplastic component of the model, the Helmholtz free energy can be set up as

$$\Psi = \Psi(\mathbb{C}_\infty, \xi) \quad (1.66)$$

$$\dot{\Psi} = \frac{\partial \Psi}{\partial \mathbb{C}_\infty} : \dot{\mathbb{C}}_\infty + \frac{\partial \Psi}{\partial \xi} : \dot{\xi} \quad (1.67)$$

Rewrite equation 1.67 with respect to the deformation gradients as,

$$\dot{\Psi} = 2\mathbf{F}_\infty \cdot \frac{\partial \Psi}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_p^{-T} : \dot{\mathbf{F}} - 2\mathbb{C}_\infty \cdot \frac{\partial \Psi}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_p^{-T} : \dot{\mathbf{F}}_p - \frac{\partial \Psi}{\partial \xi} : \dot{\xi} \quad (1.68)$$

So that the Clausius-Planck local dissipation inequality reads

$$\mathcal{D} = \mathbf{P} : \dot{\mathbf{F}} - \dot{\Psi} \quad (1.69)$$

$$= \left[ \mathbf{P} - 2\mathbf{F}_\infty \cdot \frac{\partial \Psi}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_p^{-T} \right] : \dot{\mathbf{F}} + 2\mathbb{C}_\infty \cdot \frac{\partial \Psi}{\partial \mathbb{C}_\infty} : l_p + q\dot{\xi} \geq 0 \quad (1.70)$$

from which by standard argumentation the first Piola-Kirchhoff stress tensor is obtained as

$$\mathbf{P} = 2\mathbf{F}_\infty \cdot \frac{\partial \Psi}{\partial \mathbb{C}_\infty} \cdot \mathbf{F}_p^{-T} \quad (1.71)$$

And the dissipation inequality,

$$\boldsymbol{\Sigma} : l_p + q\dot{\xi} \geq 0 \quad (1.72)$$

Where  $\boldsymbol{\Sigma} = 2\mathbb{C}_\infty \frac{\partial \Psi}{\partial \mathbb{C}_\infty}$  is the Mandel stress and  $\mathbf{S}_\infty$  is the second Piola-Kirchhoff stress tensor defined in the intermediate configuration.

On the basis of inequality 1.72 the following constrained minimization statement can be set up

$$\min \quad - \left( \boldsymbol{\Sigma} : l_p + q\dot{\xi} \right) \quad (1.73)$$

$$s.t. \quad f(\boldsymbol{\Sigma}, q) \quad (1.74)$$

Which is equivalent to the following set of plastic evolution equations

$$l_p = \dot{\mathbf{F}}_p \cdot \mathbf{F}_p^{-1} = \lambda \mathbf{N} \quad (1.75)$$

$$\dot{\xi} = \lambda \frac{\partial f}{\partial q} \quad (1.76)$$

$$\lambda \geq 0 ; f(\boldsymbol{\Sigma}, q) \leq 0 ; \lambda f(\boldsymbol{\Sigma}, q) = 0 \quad (1.77)$$

In which  $\lambda$  is the plastic consistency parameter,  $\mathbf{N} = \frac{\partial f}{\partial \boldsymbol{\Sigma}}$  and  $f(\boldsymbol{\Sigma}, q)$  is a plastic response surface. The flow rule expressed by 1.75 can be written as

$$\frac{\partial \mathbf{F}_p}{\partial t} = \lambda \mathbf{N} \cdot \mathbf{F}_p \quad (1.78)$$

In section 1.5.2 it was indicated that to obtain the second Piola-Kirchhoff stress in the reference configuration we need to compute the elastic deformation gradient,

$$\mathbf{F}_\infty^{t+\Delta t} = \mathbf{F}^{t+\Delta t} \cdot (\mathbf{F}_p^{t+\Delta t})^{-1} \quad (1.79)$$

If we assume no plastic deformation takes place during a motion in the time interval  $[t, t + \Delta t]$  then,

$$\mathbf{F}_p^{t+\Delta t} = \mathbf{F}_p^t, \quad (1.80)$$

$$\boldsymbol{\xi}^{t+\Delta t} = \boldsymbol{\xi}^t. \quad (1.81)$$

We introduce an approximation for the elastic deformation gradient  $\mathbf{F}_\infty$ ,

$$\tilde{\mathbf{F}}_\infty^{t+\Delta t} = \mathbf{F}^{t+\Delta t} \cdot (\mathbf{F}_p^t)^{-1} \quad (1.82)$$

Solve the evolution laws of equation 1.78 for the time interval  $[t, t + \Delta t]$  analytically,

$$\mathbf{F}_p^{t+\Delta t} = \left[ e^{\Delta \lambda \mathbf{N}} \right]^{t+\Delta t} \mathbf{F}_p^t \quad (1.83)$$

hence,

$$\mathbf{F}_\infty^{t+\Delta t} = \mathbf{F}^{t+\Delta t} \cdot (\mathbf{F}_p^t)^{-1} \left[ e^{-\Delta \lambda \mathbf{N}} \right]^{t+\Delta t} \quad (1.84)$$

$$= \tilde{\mathbf{F}}_\infty^{t+\Delta t} \left[ e^{-\Delta \lambda \mathbf{N}} \right]^{t+\Delta t} \quad (1.85)$$

The exponential can be approximated by a first order Taylor expansion,

$$e^{-\Delta \lambda \mathbf{N}} = \mathbf{I} - \Delta \lambda \mathbf{N} + \frac{(\Delta \lambda)^2}{2!} \mathbf{N}^2 + \dots \quad (1.86)$$

Elaborate equation 1.84 with the use of expression 1.86 and ignoring the second order term further to,

$$\mathbf{F}_\infty^{t+\Delta t} = \tilde{\mathbf{F}}_\infty^{t+\Delta t} - \Delta \lambda \mathbf{W}^{t+\Delta t} \quad (1.87)$$

where,

$$\mathbf{W}^{t+\Delta t} = \tilde{\mathbf{F}}_\infty^{t+\Delta t} \cdot \mathbf{N}^{t+\Delta t} \quad (1.88)$$

It is apparent that equation 1.87 constitutes an elastic predictor – plastic corrector solution for the elastic right Cauchy-Green tensor.



Also on the basis of equation 1.75 a backward Euler integration scheme results to the following algorithmic scheme for the hardening rule,

$$\xi^{t+\Delta t} = \xi^t + \left[ \Delta\lambda \left( \frac{\partial f}{\partial q} \right) \right]^{t+\Delta t} \quad (1.89)$$

A system of residual equations can be formulated using equations 1.87 and 1.89

$$R = \begin{pmatrix} R_{\mathbf{F}_\infty} \\ R_f \end{pmatrix} = \begin{pmatrix} \mathbf{F}_\infty^{t+\Delta t} - \tilde{\mathbf{F}}_\infty^{t+\Delta t} + \Delta\lambda \mathbf{W}^{t+\Delta t} \\ [f(\boldsymbol{\Sigma}, q)]^{t+\Delta t} \end{pmatrix} \quad (1.90)$$

Note that the residual  $R_f$  is equal to the plastic response surface  $[f(\boldsymbol{\Sigma}, q)]^{t+\Delta t}$ . This corresponds to the objective of reducing the (trial) elastic stress state to the plastic response surface as described in section 1.4.2. Hence, the plastic response on time  $t + \Delta t$  is desired to be zero.

Use a Newton-Raphson procedure [2] to solve the preceding residual equations,

$$\begin{pmatrix} \mathbf{F}_\infty^{t+\Delta t} \\ [\Delta\lambda]^{t+\Delta t} \end{pmatrix}_{r+1} = \begin{pmatrix} \mathbf{F}_\infty^{t+\Delta t} \\ [\Delta\lambda]^{t+\Delta t} \end{pmatrix}_r - \left( ([J]^{t+\Delta t})^{-1} \right)_r \begin{pmatrix} [R_{\mathbf{F}_\infty}]^{t+\Delta t} \\ [R_f]^{t+\Delta t} \end{pmatrix}_r \quad (1.91)$$

where,

$$\left( ([J]^{t+\Delta t})^{-1} \right)_r = \begin{pmatrix} \frac{\partial(R_{\mathbf{F}_\infty})_{11}}{\partial(\mathbf{F}_\infty)_{11}} & \frac{\partial(R_{\mathbf{F}_\infty})_{11}}{\partial(\mathbf{F}_\infty)_{12}} & \cdots & \frac{\partial(R_{\mathbf{F}_\infty})_{11}}{\partial(\mathbf{F}_\infty)_{33}} & \frac{\partial(R_{\mathbf{F}_\infty})_{11}}{\partial(\Delta\lambda)} \\ \frac{\partial(R_{\mathbf{F}_\infty})_{12}}{\partial(\mathbf{F}_\infty)_{11}} & \frac{\partial(R_{\mathbf{F}_\infty})_{12}}{\partial(\mathbf{F}_\infty)_{12}} & \cdots & \frac{\partial(R_{\mathbf{F}_\infty})_{12}}{\partial(\mathbf{F}_\infty)_{33}} & \frac{\partial(R_{\mathbf{F}_\infty})_{12}}{\partial(\Delta\lambda)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial(R_{\mathbf{F}_\infty})_{33}}{\partial(\mathbf{F}_\infty)_{11}} & \frac{\partial(R_{\mathbf{F}_\infty})_{33}}{\partial(\mathbf{F}_\infty)_{12}} & \cdots & \frac{\partial(R_{\mathbf{F}_\infty})_{33}}{\partial(\mathbf{F}_\infty)_{33}} & \frac{\partial(R_{\mathbf{F}_\infty})_{33}}{\partial(\Delta\lambda)} \\ \frac{\partial(R_f)}{\partial(\mathbf{F}_\infty)_{11}} & \frac{\partial(R_f)}{\partial(\mathbf{F}_\infty)_{12}} & \cdots & \frac{\partial(R_f)}{\partial(\mathbf{F}_\infty)_{13}} & \frac{\partial(R_f)}{\partial(\Delta\lambda)} \end{pmatrix} \quad (1.92)$$

### 1.5.5 Viscoelastic response

Similar to the elastoplastic component, the formulations for the viscoelastic component in the intermediate configurations need to be set-up. In the intermediate configuration, for the viscoelastic component of the model, the Helmholtz free energy function can be set up as

$$\Psi = \Psi(\mathbb{C}_e) \quad (1.93)$$

Since the Helmholtz free energy function of the viscoelastic component only depends on the strain tensor, it can also be referred to as a Strain Energy function. Its time derivative can therefore be found as

$$\dot{\Psi} = \frac{\partial \Psi}{\partial \mathbb{C}_e} : \dot{\mathbb{C}}_e \quad (1.94)$$

By means of expression 1.50, equation 1.94 can be further elaborated as

$$\dot{\Psi} = 2\mathbf{F}_e \cdot \frac{\partial \Psi}{\partial \mathbb{C}_e} \cdot \mathbf{F}_v^{-T} : \dot{\mathbf{F}} - 2\mathbb{C}_e \cdot \frac{\partial \Psi}{\partial \mathbb{C}_e} \cdot \mathbf{F}_v^{-T} : \dot{\mathbf{F}}_v \quad (1.95)$$

Hence the Clausius-Planck local dissipation inequality reads

$$\mathcal{D} = \mathbf{P} : \dot{\mathbf{F}} - \dot{\Psi} \quad (1.96)$$

$$= \left[ \mathbf{P} - 2\mathbf{F}_e \cdot \frac{\partial \Psi}{\partial \mathbb{C}_e} \cdot \mathbf{F}_v^{-T} \right] : \dot{\mathbf{F}} + 2\mathbb{C}_e \cdot \frac{\partial \Psi}{\partial \mathbb{C}_e} : l_v \geq 0 \quad (1.97)$$

with  $l_v = \dot{\mathbf{F}}_v \mathbf{F}_v^{-1}$ . From which by standard argumentation the first Piola-Kirchhoff stress tensor is obtained as

$$\mathbf{P} = 2\mathbf{F}_e \cdot \frac{\partial \Psi}{\partial \mathbb{C}_e} \cdot \mathbf{F}_v^{-T} \quad (1.98)$$

and the dissipation inequality

$$\Sigma : l_v \geq 0 \quad (1.99)$$

Where  $\Sigma = 2\mathbb{C}_e \frac{\partial \Psi}{\partial \mathbb{C}_e}$  is the Mandel stress and  $\mathbf{S}_e$  is the second Piola-Kirchhoff stress tensor defined in the intermediate configuration. The following evolution law can be found

$$l_v = \mathbb{C}_v^{-1} : \Sigma \quad (1.100)$$

With

$$\mathbb{C}_v^{-1} = \frac{1}{2\eta_D} \left( \mathbf{I} - \frac{1}{3} I \otimes I \right) + \frac{1}{9\eta_V} I \otimes I \quad (1.101)$$

While  $\eta_D$  and  $\eta_V$  are the deviatoric and volumetric viscosities which may be deformation dependent

$$\eta_D = \eta_D(\Sigma) > 0, \quad (1.102)$$

$$\eta_V = \eta_V(\Sigma) > 0 \quad (1.103)$$

Therefore

$$l_v = \dot{\mathbf{F}}_v \mathbf{F}_v^{-1} = \mathbb{C}_v^{-1} : \Sigma \quad (1.104)$$

Which can be written as

$$\frac{\partial \mathbf{F}_v}{\partial t} = (\mathbb{C}_v^{-1} : \Sigma) \cdot \mathbf{F}_v \quad (1.105)$$

In section 1.5.2 it was indicated that to obtain the second Piola-Kirchhoff stress in the reference configuration we need to compute the elastic deformation gradient,

$$\mathbf{F}_e^{t+\Delta t} = \mathbf{F}^{t+\Delta t} \cdot (\mathbf{F}_v^{t+\Delta t})^{-1} \quad (1.106)$$

If we assume no further viscous deformation takes place during a motion in the time interval  $[t, t + \Delta t]$  then,

$$\mathbf{F}_v^{t+\Delta t} = \mathbf{F}_v^t \quad (1.107)$$

We introduce an approximation for the elastic deformation gradient  $\mathbf{F}_e$ ,

$$\tilde{\mathbf{F}}_e^{t+\Delta t} = \mathbf{F}^{t+\Delta t} \cdot (\mathbf{F}_v^t)^{-1} \quad (1.108)$$

Now solve the evolution laws of equation 1.105 for the time interval  $[t, t + \Delta t]$  analytically,

$$\mathbf{F}_v^{t+\Delta t} = \left[ e^{\Delta \mathbb{C}_v^{-1} : \boldsymbol{\Sigma}} \right]^{t+\Delta t} \mathbf{F}_v^{t+\Delta t} \quad (1.109)$$

where  $\Delta \mathbb{C}_v^{-1} = \mathbb{C}_v^{-1} \Delta t$ . Hence,

$$\mathbf{F}_e^{t+\Delta t} = \mathbf{F}^{t+\Delta t} \cdot (\mathbf{F}_v^{t+\Delta t})^{-1} \left[ e^{-\Delta \mathbb{C}_v^{-1} : \boldsymbol{\Sigma}} \right]^{t+\Delta t} \quad (1.110)$$

$$= \tilde{\mathbf{F}}_e^{t+\Delta t} \left[ e^{-\Delta \mathbb{C}_v^{-1} : \boldsymbol{\Sigma}} \right]^{t+\Delta t} \quad (1.111)$$

The exponential can be approximated by a first order Taylor expansion,

$$e^{-\Delta \mathbb{C}_v^{-1} : \boldsymbol{\Sigma}} = \mathbf{I} - \Delta \mathbb{C}_v^{-1} : \boldsymbol{\Sigma} + \frac{(\Delta \mathbb{C}_v^{-1})^2}{2!} \boldsymbol{\Sigma}^2 + \quad (1.112)$$

Elaborate equation 1.110 with the use of expression 1.112 and ignoring the second order term to,

$$\mathbf{F}_e^{t+\Delta t} = \tilde{\mathbf{F}}_e^{t+\Delta t} - \Delta \mathbb{C}_v^{-1} : \mathbf{W}^{t+\Delta t} \quad (1.113)$$

where,

$$\mathbf{W}^{t+\Delta t} = \tilde{\mathbf{F}}_e^{t+\Delta t} \boldsymbol{\Sigma}^{t+\Delta t} \quad (1.114)$$

A system of residual equations can be formulated using equation 1.113,

$$R = R_{\mathbf{F}_e} = \mathbf{F}_e^{t+\Delta t} - \tilde{\mathbf{F}}_e^{t+\Delta t} + \Delta \mathbb{C}_v^{-1} : \mathbf{W}^{t+\Delta t} \quad (1.115)$$

We use a Newton-Raphson procedure to solve the preceding residual equations,

$$(\mathbf{F}_e^{t+\Delta t})_{r+1} = (\mathbf{F}_e^{t+\Delta t})_r - \left( \left( [J]^{t+\Delta t} \right)^{-1} \right)_r \left( [R_{\mathbf{F}_e}]^{t+\Delta t} \right)_r \quad (1.116)$$

where,

$$\left( \left( [J]^{t+\Delta t} \right)^{-1} \right)_r = \begin{pmatrix} \frac{\partial(R_{\mathbf{F}_e})_{11}}{\partial(\mathbf{F}_e)_{11}} & \frac{\partial(R_{\mathbf{F}_e})_{11}}{\partial(\mathbf{F}_e)_{12}} & \dots & \frac{\partial(R_{\mathbf{F}_e})_{11}}{\partial(\mathbf{F}_e)_{33}} \\ \frac{\partial(R_{\mathbf{F}_e})_{12}}{\partial(\mathbf{F}_e)_{11}} & \frac{\partial(R_{\mathbf{F}_e})_{12}}{\partial(\mathbf{F}_e)_{12}} & \dots & \frac{\partial(R_{\mathbf{F}_e})_{12}}{\partial(\mathbf{F}_e)_{33}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial(R_{\mathbf{F}_e})_{33}}{\partial(\mathbf{F}_e)_{11}} & \frac{\partial(R_{\mathbf{F}_e})_{33}}{\partial(\mathbf{F}_e)_{12}} & \dots & \frac{\partial(R_{\mathbf{F}_e})_{33}}{\partial(\mathbf{F}_e)_{33}} \end{pmatrix} \quad (1.117)$$

# Discretization

---

In order to solve the systems of equations in algorithm 1 we need to discretize in space and time. For static mechanics time plays no role and the external load steps are increased by discrete quantities already. For dynamic mechanics time plays a vital role, as the system can change between two load steps. The discretization for static and dynamic mechanics will be introduced separately. The spatial discretization will be based on the finite elements (FE) approach [1] which has been an industry standard for many years. In the FE approach the choice of elements and shape functions affects the stability and accuracy of the numerical solution. Hence, the focus is on these two aspects of FE. In preempt to dynamic mechanics, time will be discretized with a Newmark integration scheme [11].

## 2.1 Finite elements

In general in CAPA-3D arbitrary geometrical objects are subjected to stress simulation tests. In figure 2.1 such an arbitrary shape, e.g. a column of asphaltic material, is illustrated. The body is being sliced into thin layers of material and each layer is converted into a mesh of elements. In the right part of figure 2.1 a meshed layer is presented. Note that the elements in figure 2.1 have a triangular shape but it is not mandatory. One may observe various different shaped elements within the FE approach. The CAPA-3D software can handle triangular (tetrahedrals for 3D) as well as rectangular (cubes for 3D) shaped elements. It is apparent that meshes consist of elements and elements are made out of nodes and vertices. Obviously, the solution of the mathematical model for structural mechanical problems benefits from more refined meshes as approximations of the stresses and strains will be more accurate. Hence, large systems of equations can be observed regularly for even small physical experiment simulations.

### 2.1.1 Element and shape functions

Without loss of generality we introduce a tetrahedral element with a new local coordinate system  $(\xi, \eta, \zeta)$  and is illustrated in figure 2.2(a).

For discretization of the spatial coordinates we use shape functions to interpolate the coordinates between the element nodes. In figure 2.2(b) first order shape functions are drawn for a 1D element. The new local coordinate system,  $(\xi)$ , corresponds with the two nodes of the element,  $x_1$  and  $x_2$ . Every coordinate in the interval  $[x_1, x_2]$  can now be written as a function of the node coordinates as follows

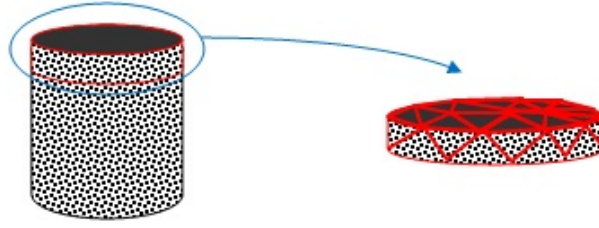
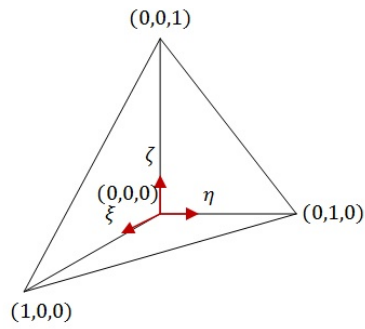
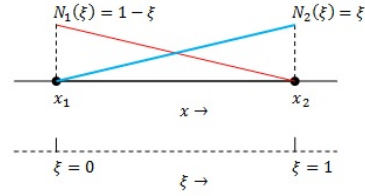


Figure 2.1: Finite element mesh applied to computer model of asphalt column.



(a) Tetrahedral element with local coordinate system  $(\xi, \eta, \zeta)$



(b) Linear shape function for 1D element.

$$x = N_1(\xi) x_1 + N_2(\xi) x_2 \quad (2.1)$$

where  $N_1$  and  $N_2$  are the linear shape functions. It is apparent that for each type of element and choice of shape functions we must have

$$\sum_{k=1}^m N_k(\xi) = 1 \quad (2.2)$$

where  $m$  is the number of nodes and thus shape functions of an element and  $\xi$  is the local coordinate vector.

Now extend this philosophy to 3D elements. The tetrahedral element of figure 2.2(a), has four nodes and hence four shape functions are introduced. The coordinate vector  $\mathbf{x}$  is expressed as

$$\mathbf{x} = \sum_{i=1}^4 \mathbf{N}_i(\xi, \eta, \zeta) \cdot \mathbf{x}_i \quad (2.3)$$

where,

$$\mathbf{N}_1(\xi, \eta, \zeta) = 1 - \xi - \eta - \zeta \quad (2.4)$$

$$\mathbf{N}_2(\xi, \eta, \zeta) = \xi \quad (2.5)$$

$$\mathbf{N}_3(\xi, \eta, \zeta) = \eta \quad (2.6)$$

$$\mathbf{N}_4(\xi, \eta, \zeta) = \zeta \quad (2.7)$$

Use an isoparametric formulation, hence discretize the displacement vector  $\mathbf{u}$  and position vector  $\mathbf{x}$  with the same shape functions.

$$\mathbf{u} = \sum_{i=1}^4 \mathbf{N}_i(\xi, \eta, \zeta) \cdot \mathbf{u}_i \quad (2.8)$$

### 2.1.2 Gauss points and numerical integration

One of the major advantages of the FE approach is numerical evaluation of integrals. Consider a continuous function  $f$  defined in the domain  $\Omega$ . Hence, the integral  $I_f$  of  $f$  over  $\Omega$  is defined as

$$I_f = \int_{\Omega} f d\Omega \quad (2.9)$$

The domain  $\Omega$  is meshed into  $n$  tetrahedral elements, therefore  $\Omega = \bigcup_{i=1}^n \Omega_i$  and write  $I_f$  as

$$I_f = \int_{\Omega_1} f d\Omega_1 + \cdots + \int_{\Omega_n} f d\Omega_n = \sum_{i=1}^n \int_{\Omega_i} f d\Omega_i \quad (2.10)$$

Use numerical integration to evaluate the integrals of the elements. The relation between the local and the natural coordinate system is represented by the Jacobian  $J$ ,

$$\begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{pmatrix} = J \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix} \quad (2.11)$$

Hence write  $I_{f_i} = \int_{\Omega_i} f d\Omega_i$  as

$$\begin{aligned} I_{f_i} &= \int_{\Omega_i} f d\Omega_i \\ &= \int_{\Omega_i} f dx dy dz \\ &= \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} f |J| d\xi d\eta d\zeta \end{aligned} \quad (2.12)$$

For the numerical evaluation of 2.12 we introduce Gauss points. A well known technique for the approximation of integrals is *Gaussian* integration. The integral in  $(\xi, \eta, \zeta)$  space of equation 2.12 can be approximated by

$$\begin{aligned}
\int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} f |J| d\xi d\eta d\zeta &= f(\xi_a, \eta_a, \zeta_a) |J| \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} d\xi d\eta d\zeta \\
&\hat{=} f(\xi_a, \eta_a, \zeta_a) |J| \frac{1}{6} \\
&= \tilde{I}_{f_i}
\end{aligned} \tag{2.13}$$

where  $(\xi_a, \eta_a, \zeta_a)$  is the Gauss point of element  $\Omega_i$ . For each different geometrical shaped element one can calculate the optimal position of the Gauss point such that error  $|I_{f_i} - \tilde{I}_{f_i}|$  is minimized. It is apparent that adding more Gauss point to an element will increase the accuracy of the numerical evaluation of the integral. Again, optimal positions for Gauss points have been calculated for many types of elements and the values can be found in textbooks.

## 2.2 Discretization balancing of forces

In the previous sections we discussed the balancing of forces and introduced analytical expressions for the stiffness matrix and the internal and external forces acting on the material. With the FE approach of the preceding we have a powerful tool to evaluate the integrals numerically and operate in discretized space. The balancing of forces algorithm 1 only describes static mechanics. This means that between to load steps time is not a variable. Hence, velocity of the material bodies was not embedded in the algorithm. In most real-life applications time does play a role, so velocity and acceleration will be taken into account. As both types of tests are common practice, the discretization of time and space are considered separately. For static mechanics, there is only spatial discretization in contrary to dynamic mechanics where time will be discretized too. Furthermore, this section will evaluate a few examples of static and dynamic simulations.

### 2.2.1 Static mechanics

In section 1.3 we have derived the linearized virtual work equation at the equilibrium. This equation applies to every point in body  $V$ . In previous sections we have seen how to divide body or domain  $V$  into  $n$  subdomains  $V_i$ . When the subdomains represent grid cells resulting from a mesh the finite elements formulation of Section 2.1.1 is applied to the linearized virtual work equation,

$$\begin{aligned}
\int_V (\nabla_0 \Delta \mathbf{u} \cdot \mathbf{S}) : \nabla_0 \delta \mathbf{v} dV &+ \int_V (\nabla_0 \Delta \mathbf{u} : \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T) : \nabla_0 \delta \mathbf{v} dV \\
&= \delta \mathbf{v} \cdot \mathbf{f}_{ext} - \int_V \mathbf{P} : \nabla_0 \delta \mathbf{v} dV
\end{aligned} \tag{2.14}$$

In this way a discretized linearized virtual work equation is derived.

Assume that we are working on an element with arbitrary shape and  $N$  nodes, e.g. a tetrahedral. Therefore we have  $N$  shape functions,  $N_k$  as described in Section



2.1.1. Because we have an isoparametric formulation the following quantities can be derived,

$$\nabla_0 \Delta \mathbf{u} = \sum_{k=1}^N \Delta \mathbf{u}_k \otimes \nabla_0 N_k = \sum_{k=1}^N \Delta \mathbf{u}_k \otimes \frac{\partial N_k}{\partial \mathbf{X}} \quad (2.15)$$

$$\nabla_0 \delta \mathbf{v} = \sum_{k=1}^N \delta \mathbf{v}_k \otimes \nabla_0 N_k = \sum_{k=1}^N \delta \mathbf{v}_k \otimes \frac{\partial N_k}{\partial \mathbf{X}} \quad (2.16)$$

in which  $\Delta \mathbf{u}$  and  $\delta \mathbf{v}$  are the displacement increment and virtual velocity respectively. Substitute former expression into equation 2.14. and we obtain expressions for all integrals.

First, the virtual internal work.

$$\begin{aligned} \delta W_{int} &= \int_V \mathbf{P} : \nabla_0 \delta \mathbf{v} dV \\ &= \int_V \mathbf{P} : \left( \sum_{k=1}^N \delta \mathbf{v}_k \otimes \frac{\partial N_k}{\partial \mathbf{X}} \right) dV \\ &= \sum_{k=1}^N \delta \mathbf{v}_k \int_V \mathbf{P} : \left( \mathbf{I} \otimes \frac{\partial N_k}{\partial \mathbf{X}} \right) dV \\ &= \sum_{k=1}^N \delta \mathbf{v}_k \int_V \mathbf{P} : \mathbf{B}_0 dV \end{aligned} \quad (2.17)$$

The derivative in the direction of the incremental displacement of the internal work,

$$\begin{aligned} D_{\Delta \mathbf{u}} \delta W_{int} &= \int_V (\nabla_0 \Delta \mathbf{u} \cdot \mathbf{S}) : \nabla_0 \delta \mathbf{v} dV + \int_V (\nabla_0 \Delta \mathbf{u} : \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T) : \nabla_0 \delta \mathbf{v} dV \\ &= \int_V (\nabla_0 \Delta \mathbf{u} \cdot \mathbf{S}) : \nabla_0 \delta \mathbf{v} dV + \int_V (\mathbb{C} : \mathbf{F}^T \cdot \nabla_0 \Delta \mathbf{u}) : \mathbf{F}^T \cdot \nabla_0 \delta \mathbf{v} dV \\ &= \int_V \left( \left( \sum_{l=1}^N \Delta \mathbf{u}_l \otimes \frac{\partial N_l}{\partial \mathbf{X}} \right) \mathbf{S} \right) : \left( \sum_{k=1}^N \delta \mathbf{v}_k \otimes \frac{\partial N_k}{\partial \mathbf{X}} \right) dV \\ &\quad + \int_V \mathbf{F}^T \left( \sum_{l=1}^N \Delta \mathbf{u}_l \otimes \frac{\partial N_l}{\partial \mathbf{X}} \right) : \mathbb{C} : \mathbf{F}^T \left( \sum_{k=1}^N \delta \mathbf{v}_k \otimes \frac{\partial N_k}{\partial \mathbf{X}} \right) dV \end{aligned} \quad (2.18)$$

Because it holds that (Appendix [?]),

$$\left( \Delta \mathbf{u}_l \otimes \frac{\partial N_l}{\partial \mathbf{X}} \right) \cdot \mathbf{S} : \left( \delta \mathbf{v}_k \otimes \frac{\partial N_k}{\partial \mathbf{X}} \right) = \Delta \mathbf{u}_l \cdot \frac{\partial N_l}{\partial \mathbf{X}} \cdot \mathbf{S} \cdot \frac{\partial N_k}{\partial \mathbf{X}} \cdot \mathbf{I} \cdot \delta \mathbf{v}_k \quad (2.19)$$

and also,

$$\mathbf{F}^T \cdot \left( \Delta \mathbf{u}_l \otimes \frac{\partial N_l}{\partial \mathbf{X}} \right) = \Delta \mathbf{u}_l \cdot \left( \mathbf{F} \otimes \frac{\partial N_l}{\partial \mathbf{X}} \right) \quad (2.20)$$

rewrite equation 2.18 as,

$$\begin{aligned}
D_{\Delta \mathbf{u}} \delta W_{int} &= \delta \mathbf{v}_k \sum_{l=1}^N \sum_{k=1}^N \int_V \frac{\partial N_l}{\partial \mathbf{X}} \cdot \mathbf{S} \cdot \frac{\partial N_k}{\partial \mathbf{X}} \cdot \mathbf{I} dV \\
&+ \int_V \left( \mathbf{F} \otimes \frac{\partial N_l}{\partial \mathbf{X}} \right) : \mathbb{C} : \left( \mathbf{F} \otimes \frac{\partial N_k}{\partial \mathbf{X}} \right) dV \Delta \mathbf{u}_l \\
&= \delta \mathbf{v}_k \sum_{l=1}^N \sum_{k=1}^N \int_V \nabla_0 N_l \cdot \mathbf{S} \cdot \nabla_0 N_k \cdot \mathbf{I} dV + \int_V (\mathbf{B}_L)_l : \mathbb{C} : (\mathbf{B}_L)_k dV \Delta \mathbf{u}_l
\end{aligned} \tag{2.21}$$

where,

$$\nabla_0 N_l = \frac{\partial N_l}{\partial \mathbf{X}} \tag{2.22}$$

$$(\mathbf{B}_L)_l = \left( \mathbf{F} \otimes \frac{\partial N_l}{\partial \mathbf{X}} \right) \tag{2.23}$$

At last the discretized form of the external virtual work is expressed as,

$$\delta W_{ext} = \sum_{k=1}^N \delta \mathbf{v}_k \cdot (\mathbf{f}_{ext})_k. \tag{2.24}$$

This leads to the final form of the discretized, linearized virtual work equation at equilibrium,

$$\begin{aligned}
\delta \mathbf{v}_k \sum_{l=1}^N \sum_{k=1}^N \int_V \nabla_0 N_l \cdot \mathbf{S} \cdot \nabla_0 N_k \cdot \mathbf{I} dV &+ \int_V (\mathbf{B}_L)_l : \mathbb{C} : (\mathbf{B}_L)_k dV \Delta \mathbf{u}_l \\
&= \delta \mathbf{v}_k \sum_{k=1}^N (\mathbf{f}_{ext})_k - \delta \mathbf{v}_k \sum_{k=1}^N \int_V \mathbf{P} : \mathbf{B}_0 dV.
\end{aligned} \tag{2.25}$$

Clearly equation 2.25 must hold for all  $\delta \mathbf{v}_k$  by definition, hence eliminate  $\delta \mathbf{v}_k$  from the equation. Write the discrete equilibrium equation as,

$$(\mathbf{K}^\sigma + \mathbf{K}^c) \cdot \Delta \mathbf{u} = \Delta \mathbf{f}_{ext} - \Delta \mathbf{f}_{int} \tag{2.26}$$

Define  $\mathbf{K} = \mathbf{K}^\sigma + \mathbf{K}^c$  as the stiffness matrix. This matrix represents the reaction of the material to a change/misbalance in/of forces. It is a unique material property. Here  $\mathbf{K}^\sigma$ ,  $\mathbf{K}^c$  represent the internal forces and correspond with the geometrical stress components and the constitutive components respectively.

Finally, the balancing of forces algorithm is constituted by equation 2.26 and given by Algorithm 1. Note that the computation of the material response, which is represented by  $\mathbf{f}_{int}^i$  results from the algorithm that have been introduced in Section

1.5.2. The external force  $\mathbf{f}_{ext}^i$  results from the evaluation of  $\delta W(\mathbf{X}_0)$  and the term  $\mathbf{K}^t \cdot \Delta \mathbf{u}_p$  where  $\Delta \mathbf{u}_p$  are the prescribed displacements of the body. Furthermore, it must be emphasized that time plays no visible role in these algorithms, as we are dealing with load steps instead of time steps. But of course these load steps endure a fixed period of time. The first step in Algorithm 1 is the appliance of an external load to the body. This is either stress or strain regulated. The second step is the calculation of the displacements as a response to the external load, which is equal to solving linear system equation 2.26. The third and last step is the appliance of the material properties, i.e. the calculation of the non-linear internal stresses.

---

**Algorithm 1** Balancing of forces

---

```

for  $t = 0 \dots t_{\text{end}}$  do
  Compute external load  $\mathbf{f}_{ext}^t$ 
  for  $i = 0$  until convergence do
    Assemble stiffnessmatrix  $K^{t,i}$ 
    if  $i = 0$  then
       $\mathbf{f}_{int}^0 = 0$  and  $\Delta \mathbf{f}^0 = \mathbf{f}_{ext}^0 - \mathbf{f}_{int}^0$ 
    Solve system  $K^{t,i} \Delta \mathbf{u}^i = \Delta \mathbf{f}^i$ 
    Update displacements,  $\mathbf{u}^{i+1} = \mathbf{u}^i + \Delta \mathbf{u}^i$ 
    Compute internal force,  $\mathbf{f}_{int}^{i+1}$  and  $\Delta \mathbf{f}^{i+1} = \mathbf{f}_{ext}^{i+1} - \mathbf{f}_{int}^{i+1}$ 
    Test for convergence,  $\frac{\Delta \mathbf{f}^{i+1}}{\Delta \mathbf{f}^0} < \varepsilon$ 

```

---

### 2.2.2 Dynamic mechanics

The static mechanics theory can be extended to the time domain yielding dynamic mechanics. In time space not only the displacement field but also velocity and acceleration of the body have to be taken into account. Consider the linearized virtual work equation extended to the time domain in 2.27. In addition to the static virtual work equation damping matrix  $C$ , mass matrix  $M$ , velocity  $\mathbf{v}$  and acceleration  $\mathbf{a}$  have been added.

$$K\mathbf{u} + C\mathbf{v} + M\mathbf{a} = \mathbf{f}_{ext} \quad (2.27)$$

It is apparent that equation 2.27 is a second order differential equation and will have to be solved numerically. Introduce the Newmark integration scheme for solving equation 2.27. The virtual work equation is still solved for the unknown displacement field  $\mathbf{u}$ , hence the acceleration and velocity are written as function of displacement.

$$\begin{aligned} \mathbf{u}^{t+\Delta t} &= \mathbf{u}^t + \Delta t \mathbf{v}^t + (\Delta t)^2 \left[ \left( \frac{1}{2} - \beta \right) \mathbf{a}^t + \beta \mathbf{a}^{t+\Delta t} \right] \\ \mathbf{v}^{t+\Delta t} &= \mathbf{v}^t + \Delta t \left[ (1 - \gamma) \mathbf{a}^t + \gamma \mathbf{a}^{t+\Delta t} \right] \end{aligned} \quad (2.28)$$

Introduce predictor values at beginning of time step  $t$  using those terms of equation 2.28 that refer to the end of the previous time step,

$$\begin{aligned}\bar{\mathbf{u}}^{t+\Delta t} &= \mathbf{u}^t + \Delta t \mathbf{v}^t + (\Delta t)^2 \left(\frac{1}{2} - \beta\right) \mathbf{a}^t \\ \bar{\mathbf{v}}^{t+\Delta t} &= \mathbf{v}^t + \Delta t (1 - \gamma) \mathbf{a}^t\end{aligned}\quad (2.29)$$

Rewrite equation 2.28 by substitution of equation 2.29,

$$\begin{aligned}\mathbf{u}^{t+\Delta t} &= \bar{\mathbf{u}}^{t+\Delta t} + (\Delta t)^2 \beta \mathbf{a}^{t+\Delta t} \\ \mathbf{v}^{t+\Delta t} &= \bar{\mathbf{v}}^{t+\Delta t} + \Delta t \gamma \mathbf{a}^{t+\Delta t}\end{aligned}\quad (2.30)$$

In addition to the velocity an expression for the acceleration is obtained from equation 2.30,

$$\mathbf{a}^{t+\Delta t} = \frac{1}{(\Delta t)^2 \beta} (\mathbf{u}^{t+\Delta t} - \bar{\mathbf{u}}^{t+\Delta t}) \quad (2.31)$$

For every timestep  $\Delta t$  a new force equilibrium has to be computed, hence the Newmark integration scheme has to be embedded into the Newton-Raphson iteration scheme of Algorithm 1 in order to solve system 2.27. Define the displacements, velocity and acceleration for time  $t + \Delta t$  at Newton-Raphson iteration number  $k + 1$  as,

$$\begin{aligned}\mathbf{u}_{k+1}^{t+\Delta t} &= \mathbf{u}_k^{t+\Delta t} + \Delta \mathbf{u}_k^{t+\Delta t} \\ \mathbf{v}_{k+1}^{t+\Delta t} &= \bar{\mathbf{v}}^{t+\Delta t} + \frac{\gamma}{(\Delta t)\beta} (\mathbf{u}_{k+1}^{t+\Delta t} - \bar{\mathbf{u}}^{t+\Delta t}) \\ \mathbf{a}_{k+1}^{t+\Delta t} &= \frac{1}{(\Delta t)^2 \beta} (\mathbf{u}_{k+1}^{t+\Delta t} - \bar{\mathbf{u}}^{t+\Delta t})\end{aligned}\quad (2.32)$$

where  $\mathbf{u}_0^{t+\Delta t} = \bar{\mathbf{u}}^{t+\Delta t}$ . Substitution of the expressions from equation 2.32 into equation 2.27 yields,

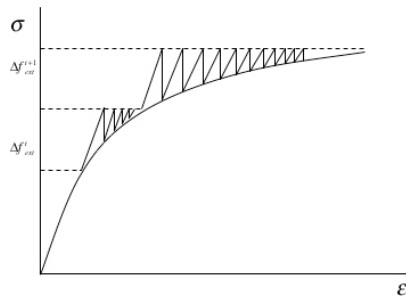
$$K^* \Delta \mathbf{u}_{k+1}^{t+\Delta t} = \tilde{\mathbf{f}}_{ext,k+1}^{t+\Delta t} - \mathbf{f}_{int,k+1}^{t+\Delta t} \quad (2.33)$$

where,

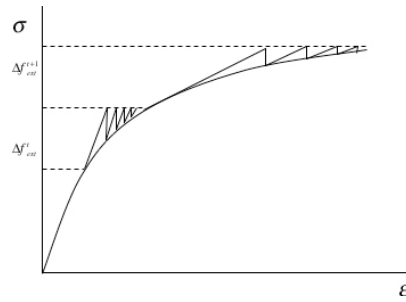
$$\begin{aligned}K^* &= K + \frac{\gamma}{\Delta t \beta} C + \frac{1}{(\Delta t)^2 \beta} M \\ \tilde{\mathbf{f}}_{ext,k+1}^{t+\Delta t} &= \mathbf{f}_{ext}^{t+\Delta t} - C \mathbf{v}_k^{t+\Delta t} - M \mathbf{a}_k^{t+\Delta t} \\ \mathbf{f}_{int,k+1}^{t+\Delta t} &= K \mathbf{u}_k^{t+\Delta t}\end{aligned}$$

## 2.3 Non-linear material properties

When plasticity or viscosity builds up the stiffness of the body changes the initial stiffness matrix does not represent the stiffness of the body at an arbitrary load step. The approximation of the stiffness matrix affects the performance of the Newton-Raphson method and yields bad convergence rates and a large number of iterations. To reduce the number of iterations modification of the stiffness matrix is required, preferably every iteration of the Newton-Raphson scheme in Algorithm 1.



(d) Newton-Raphson convergence pattern with initial stiffness approach.



(e) Newton-Raphson convergence pattern with modified stiffness approach.



# Problems arising in structural mechanics

---

## 3.1 Problem sizes

In the previous section we have set up a computational framework for structural mechanics. This framework is implemented within the CAPA-3D software. In the field of structural mechanics and pavement engineering in particular, the scale of the problems we handle are of major importance. Roughly, we distinguish four levels of modeling the materials.

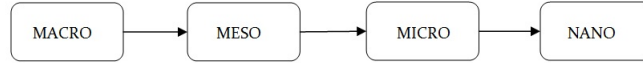


Figure 3.1: The four levels of modeling materials

At macro level, we consider the body as a multi-phase, homogenous material but in reality it consists of more than one material.. This material has just one specific stiffness and one set of plasticity and viscosity parameters. In the context of the finite elements approach, each element will have the same material properties.

At meso level, the body is a non-homogenous material. In other words, consider the body as a mixture of different materials. The body can be divided in smaller bodies, made of homogenous materials.

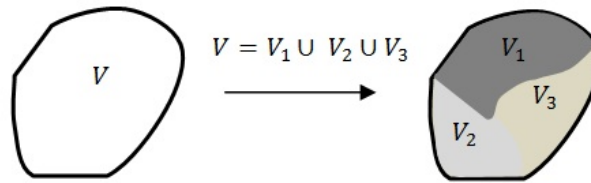


Figure 3.2: Division of material into smaller bodies.

Each of the sub bodies has a specific material property and multiply bodies consist of the same material. A good example in pavement engineering is asphalt. This material is a mixture of aggregates, bitumen and air voids. In the context of the finite elements approach, material properties can vary per element. These grid variations will have a big impact on the development of new solvers.

At micro level we no longer use bodies of material but clusters of molecules. A well known cluster of molecules is the crystal shape.

At nano level the cluster approach is abandoned and individual molecules are modeled.

Unfortunately, the current generation of computers are not capable of running any meaningful tests with micro and nano level approaches. The molecular scale asks for too much grid points and hence matrices will be extremely large and unmanageable. The micro and nano level approaches will come into play when local effects have to be treated and problem sizes remain small. Evidently CAPA3D will handle macro and meso level problem sizes only. However, as CPU power and memory sizes increase every year, micro and nano level computations will be feasible in the nearby future.

The aim of improving this software will be the ability of handling fine meshes with a larger number of degrees of freedom. Obviously this will imply a large computational effort when it comes to the evaluation of the outer iteration loop ( $K\Delta u = \Delta f$ ) as well as the inner iteration loop (return mapping procedure, compute  $F_{int}$ ) of the () algorithm.

## 3.2 Improvements algorithm

In general, three parts of the CAPA3D software have to be improved. First the existing Gaussian-elimination of solving the system  $K\Delta u = \Delta f$  has to be replaced by an efficient (parallel)direct or iterative solver. Secondly, the newly developed solver has to accommodate rescaling of our problem definition. In practice this means that the use of finer meshes still requires the same amount of CPU power and memory size. The third and last part will be the improvement of the inner iteration loop. The inner iteration loop is a pre-determined Newton-Raphson procedure and therefore we focus on the efficiency instead of scaling.

## 3.3 Development issues

Now that the specifications of the new solver are defined, the possible difficulties that lie ahead can be addressed.

## 3.4 Singularity of the stiffness matrix

Singularity of the stiffness matrix is one of the most challenging problems. In two cases we the stiffness matrix  $K$  can have singularities.

In preceding sections the stress-strain curve that can be found for every material was introduced. When not only purely elastic effects but also plasticity and viscosity effects tribute to the stress-strain relation, a point of maximum stress within the curve can be observed. At this point the differential of the stress,  $\Delta\sigma$ , tends to zero



and remains small in the neighborhood of this point. This is illustrated best by the following relation which is valid for small strains,

$$\sigma = D\varepsilon \rightarrow \Delta\sigma = D\Delta\varepsilon \rightarrow D = \frac{\Delta\sigma}{\Delta\varepsilon} \rightarrow D^{-1} = \frac{\Delta\varepsilon}{\Delta\sigma} \quad (3.1)$$

Also for small strains the stiffness matrix  $K$  is defined as,

$$K = \int_V B^T D B dV \quad (3.2)$$

which establishes the fact that  $K$  becomes (almost) singular for very small values of  $\Delta\sigma$ . When realize that the relation between strain and stress depends on time, this point of maximum stress can be avoided by increasing the time steps. However, this is a rule of thumb and gives no guarantee that this point can be passed without any problems (singularities). One way of tackling this effect is the arc length approach and can be found in [arc. length reference hier!!]

The second reason of singularity is the difference of material properties. The finite elements approach determines the assembly of the stiffness matrix. When two elements of the mesh represent two different materials with enormous differences in stiffness, this can yield a singularity in the stiffness matrix. In preempt to later sections, these singularities can also occur when an interface is applied between two materials. Usually these interfaces have a very high normal stiffness coefficient and smaller tangential stiffness coefficient. When embedded in the stiffnessmatrix  $K$  the differences between stiffness coefficients can lead to an ill-conditioned system.

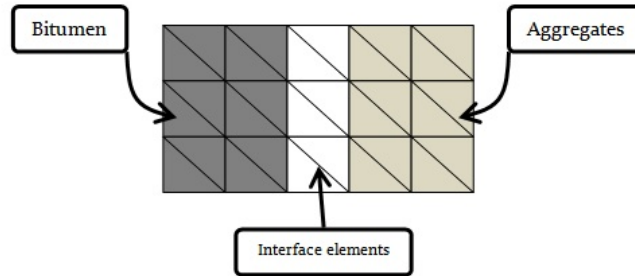


Figure 3.3: Interface elements

Another way of embedding interface elements is to add new constraints to the system. Expand the original system with constraint matrices  $L$  and  $M$  and put constraints on the displacements.

$$K\Delta u = \begin{bmatrix} \hat{K} & L \\ L^T & M \end{bmatrix} \begin{bmatrix} \Delta\hat{u} \\ \Delta\hat{\sigma} \end{bmatrix} = \begin{bmatrix} \Delta F \\ a^* \end{bmatrix} \quad (3.3)$$

Where  $\hat{K}$  is the stiffness matrix without the interface elements with corresponding displacements  $\Delta\hat{u}$  and  $M$  is the stiffness matrix of the interface elements with corresponding stress  $\Delta\hat{\sigma}$ . Inevitable the matrix  $K$  will become nearly singular when  $K_{ij} \gg M_{ij}$ .

As singularity is a well known reason for breaking down direct solution methods it can have a negative influence on the convergence rates of iterative methods as well. Many of these methods utilize direct methods within their iteration loops. Also grid dependent operators within these methods can have problems with local singularities.

### 3.5 Misbalancing due to plasticity and viscosity

In the previous sections it has become clear that plastic and viscous behavior is extremely difficult to model correctly. The return mapping procedures for both plasticity and viscosity models have a major drawback on computation times, with the amount of work per iteration equal to  $\mathcal{O}(N)$ , where  $N$  is the number of grid cells. Parallelization of the algorithm and the use of sub domains or coarser grids have the potential to reduce the computation times considerably. Unfortunately, these methods depend strongly on the balancing of forces. Small distortions of the real displacements can easily result into divergence of the balancing of forces algorithm.

#### 3.5.1 Plastic response surface

In the preceding discussion about plasticity the importance of the plastic response surface became clear. When plasticity is being built up (hardening) the stress can be calculated and with the return mapping procedure, the admissible stress state at the plastic response surface can be reached. However, when too much load is being applied and the increase of the stress is too large the corresponding maximum response can be miscalculated. This effect is illustrated in figure 3.4.

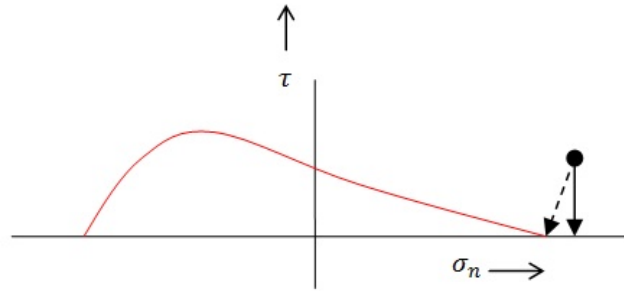


Figure 3.4: Example of miscalculation of plasticity.

A direct consequence of these miscalculations is that the resulting displacements are miscalculated also. Hence, in the next step again the wrong load is being applied and chances are high that the plastic response is miscalculated also. It is apparent that all these miscalculations contribute to a major distortion of the real solution and could result into divergence of the balancing of forces algorithm. The initial load could be induced by wrong initialization of a sub domain or coarser grid.

### 3.5.2 Viscosity

The viscosity effects also contribute to the stiffness matrix. This effect can be easily addressed for small strain problems. Consider an arbitrary material with the following stiffness coefficient,

$$\frac{E}{(1 - 2\nu)(1 + \nu)} \quad (3.4)$$

where  $E$  is the Youngs' modules and  $\nu$  the poisson ratio. When the material becomes more viscous the poisson ratio  $\nu$  tends to  $\frac{1}{2}$  hence the material becomes almost incompressible. For large strain models this effect is somehow more difficult to address but the philosophy remains the same. Apparently, when using different materials within one body and thus finite element mesh, viscosity parameters can differ from element to element. Therefore large differences between stiffness coefficients can occur and the stiffness matrix can become ill-conditioned.

## 3.6 Domain decomposition and multiple grids

Some of the solution methods depend on domain decomposition or use multiple grids. In other words, the division of the original mesh into sub domains or coarser grids. In this case we distinguish between domain decomposition and multi(ple) grid methods.

The first type is a true domain decomposition and cuts the existing domain in a number of (equal) sub domains. As an arbitrary point in the system has a direct or indirect link to every other point, there must be communication between the sub domains. Also the interfaces between the domains must be defined. The grid is moving in time, hence either the domains must be moving as well or there must be a rule to assign grid points to a specific domain.

A second way is the introduction of coarse and fine grids. If the grid is defined as a set of grid points, then a coarse grid is a subset of these points. In practice, multiple cells are glued together to form one bigger, super grid cell. For instance, when using tetrahedral shaped cells, combining three cells into one yields the removal of  $12 - 4 = 8$  grid points from the fine grid.

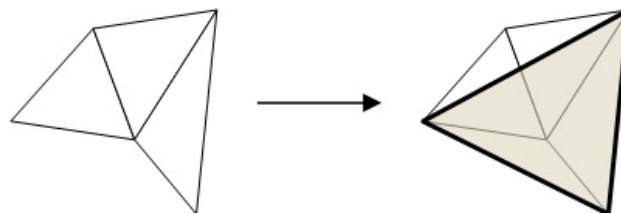


Figure 3.5: Combining grid cells of triangular mesh.

In the notion of coarse grids, combining grid cells is a difficult operation. When utilizing the finite elements method shape functions interpolate the values of the

unknowns through the element. If we use first order, linear shape functions, the grid cells must be small to obtain good convergence rates. Otherwise the approximation of stress and strain in an element becomes too poor and there will be no balance of forces. However, when coarser grids are defined linear shape functions could prove to be insufficient for yielding good approximations. Hence, the choice of shape functions must be taken into account.

Another side effect of gluing grid cells together comes from the choice of coarser grid cells. From figure () it is apparent that the gray, coarser grid cell is not a one-to-one projection of the fine grid to the coarse grid. The vertices of the coarse grid cell do coincide with the fine grid but the boundaries of the cell do not lie on the boundaries of the fine grid at all. We need a smart algorithm that can construct a coarse grid that follows the shape of the fine grid wherever it is possible. Furthermore, in preempt to the introduction of iterative methods that use coarser grids, we will see that because coarse and fine grids do not coincide, the propagation of material properties is far from trivial. When elements are combined that consists of different materials we need to determine what the material properties will be of this coarser element.

# Basics numerical computations and analysis

---

In previous sections we have introduced the basics of structural mechanics, set up a framework to do finite elements analysis and discussed the current limitations of the () algorithms. For this purpose a number of numerical solution methods are considered so more insight in how and where algorithm () will have to be adapted will be obtained. This Section emphasizes on best-practice numerical methods. In order to fully comprehend more advanced methods like multigrid and Krylov subspace methods a brief and general introduction to numerical analysis is provided. This introduction is based on the works of [13], [9] and [14]. More elaborated readers can shift directly to subsections (4.5) till (??).

In general a linear system of equations will be solved,

$$A\mathbf{x} = \mathbf{b}, \quad (4.1)$$

where  $A$  is a matrix of dimension  $n \times n$  and  $\mathbf{x}$ ,  $\mathbf{b}$  represent vectors of dimension  $n$ . Note that in many engineering problems the system of equation (4.1) often results from discretization or linearization of the governing equations of your model. For solving system (4.1) there are many suitable methods available but they can be discriminated as two groups. The direct solution methods of Section (4.1) and the iterative methods of Sections (4.2) to (4.4). The main difference rests in the underlying mathematical philosophy. Where direct solution methods invert matrix  $A$  of (4.1) to obtain a solution  $\mathbf{x}$  instantaneously, the iterative methods need (many) iteration sweeps to converge to a solution with tolerated error. Both approaches have their limitations, hence it is common practice that both methods go hand-in-hand to form one integrated solver.

Not only is there a difference between direct- and iterative solution methods, but there lies a massive spectrum of different solvers within these two methods. For direct solvers there are  $LU$  decomposition, Schwarz and Schur complements and also rearranging algorithms to speed up the decomposition process. The iterative solvers span a world of their own. There are methods based on the discretization of the model like domain decomposition and multigrid. Many methods are distinguished based on the splitting of the matrix, like Jacobi, Gauss-Seidel and SSOR. There are Krylov subspace methods that construct a new space with the residuals of the iterations as basis vectors. Finally, algorithms for conditioning of the matrix play an important role in the robustness and efficiency of the iterative solution methods.

Despite this vast number of possible solution methods, the characterisation of  $A$  often determines your method of choice. For ill-conditioned systems, direct solution methods break down. On the other hand, an iterative method like multigrid needs smooth quantities in the solution to be a robust and stable solver. The system that has to be solved constitutes the method of choice and hence it is of great importance that stability, robustness and efficiency issues within our numerical recipes as well as in the system itself are addressed.

The first Section (4.1) will discuss direct methods and Section (4.2) will introduce the iterative methods. At the end of this chapter, Section (??) will discuss different advanced iterative methods.

## 4.1 Direct solution methods

The most straightforward approach for solving system 4.1 would be a direct inversion of matrix  $A$ . The LU decomposition of matrix  $A$  is a well known technique for computing the inverse. In fact, the inverse will not be computed at all but the LU decomposition yields the same result as the real inversion of matrix  $A$ . For small matrices the LU decomposition can be done on a single computing node. However, for large systems lack of memory and limited CPU power increase computation times significantly. Above certain limits the LU decomposition will not fit into memory anymore and other (parallel) approaches will have to be used.

Not only physical barriers limit the use of direct solution methods. With ill conditioned systems the matrix will be (nearly) singular. From () we know that for these systems an inverse matrix does not exist or is very hard to compute without exact arithmetics. In those cases we need to pre-condition the system when inverting the matrix remains the method of choice.

Section () will introduce parallel direct solution methods that rely on the same principle as LU decomposition but can handle much larger systems.

### 4.1.1 LU decomposition

For matrix  $A$  is non-singular there are Gauss transformations  $M_1 \dots M_{k-1}$  such that the matrix  $U$  given by,

$$M_{n-1}M_{n-2} \cdots M_2M_1A = U \quad (4.2)$$

is upper triangular [14]. It can be derived that the inverse of  $M_{n-1} \dots M_1$  can be given by,

$$L = (M_1 \dots M_{n-1})^{-1} \quad (4.3)$$

which implies that  $A = LU$ . The matrix  $L$  is lower triangular and  $\text{diag}(L) = I$ . Once the LU decomposition is obtained  $Ax = b$  is easily solved. First solve  $Ly = b$  and then the upper triangular system  $Ux = y$ .

The algorithm for finding matrices  $L$  and  $U$  is called the Gaussian elimination algorithm 2.

---

**Algorithm 2** Gaussian elimination algorithm
 

---

Compute  $A = LU$  with  $A \in \mathbb{R}^{n \times n}$

```

for  $k = 1, \dots, n - 1$  do
  if  $a_{kk} = 0$  then
    quit
  else
    for  $i = k + 1, \dots, n$  do
       $\eta = \frac{a_{ik}}{a_{kk}}$ 
       $a_{ik} = \eta$ 
      for  $j = k + 1, \dots, n$  do
         $a_{ij} = a_{ij} - \eta a_{kj}$ 
  
```

---

Take into account that due to round off errors and machine precision Gaussian elimination can give arbitrary poor results, even for well conditioned systems. With partial or complete pivoting or by applying iterative improvements the algorithm of table 2 can be improved significantly. Implementations of the LU decomposition that can be found in freely available LAPACK software package make use of these enhanced algorithms. Hence, in general stability of this solution method for well conditioned systems should be ensured.

### 4.1.2 Cholesky factorization

In Chapter 2 we seen that the matrix resulting from the discretization of the virtual work equation was both positive definite and symmetric. For these matrices there exists an unique lower triangular  $R \in \mathbb{R}^{n \times n}$  with positive diagonal entries such that  $A = RR^T$ . This is the Cholesky factorization. Because of the symmetry of  $A$  only the upper part has to be stored. In general the computation of the Cholesky factorization takes halve the amount of work and memory compared to the Gaussian elimination. In later sections (incomplete) Cholesky factorization is used for preconditioning.

## 4.2 Basic iterative methods

We recall the system of equation (4.1),

$$A\mathbf{x} = \mathbf{b}.$$

Many iterative methods are based on the fixed point iteration for solving system (4.1),

$$\mathbf{x}^{k+1} = G\mathbf{x}^k + \mathbf{f} \quad (4.4)$$

where  $\mathbf{x}^k$  is an approximation of the exact solution  $\mathbf{x}$  of system (4.1). Here  $G$  is the iteration matrix and  $\mathbf{f}$  is the source vector. Matrix  $M$  and vector  $\mathbf{s}$  are determined by our method of choice. When utilizing an iterative method it is important that we introduce a proper stopping criterion. Otherwise our method could iterate until infinity ( $k \rightarrow \infty$ ) or stop too early. One common used stopping criterion is the relative norm of the residual and is defined as,

$$\frac{\|\mathbf{b} - A\mathbf{x}^k\|}{\|\mathbf{b}\|} < \varepsilon \quad (4.5)$$

for which  $\|\cdot\|$  is a norm and  $\varepsilon$  is the tolerance. Also important is the initial guess  $\mathbf{x}^0$ , one often uses the zero vector.

In order to measure the efficiency of an iterative method we compare the rates of convergence. We say that the rate of convergence of the general method in equation (4.4) is characterized by the spectral radius  $\rho(G)$  and is defined as,

$$\rho(G) = \max \{|\lambda| : \lambda \text{ eigenvalue } G\}. \quad (4.6)$$

In fact, the spectral radius is the asymptotic convergence factor of the iteration [13]. Asymptotically, i.e. for  $k \rightarrow \infty$ , we have  $\|\mathbf{x} - \mathbf{x}^{k+1}\| \leq \rho(G) \cdot \|\mathbf{x} - \mathbf{x}^k\|$ .

There are many ways to construct an iteration matrix  $G$ , leading to different solvers and eventually all basic iterative methods can be expressed as the system in (4.4). This section will only discuss one example of a basic iterative method based on the splitting of matrix  $A$  in (4.1).

### 4.2.1 Splitting of matrix

Introduce the splitting of the matrix  $A$ ,

$$A = M - N \quad (4.7)$$

When  $\mathbf{x}^{k+1}, \mathbf{x}^k$  are close to  $\mathbf{x}$  we have  $\mathbf{x}^{k+1} \simeq \mathbf{x}^k \simeq \mathbf{x}$  hence it makes sense to introduce the following iteration scheme,

$$M\mathbf{x}^{k+1} = N\mathbf{x}^k + \mathbf{b} \quad (4.8)$$

Rewrite this in terms of system (4.1),

$$G = M^{-1}N = M^{-1}(M - A) = I - M^{-1}A \quad (4.9)$$

$$\mathbf{f} = M^{-1}\mathbf{b} \quad (4.10)$$

In many engineering applications we see a reoccurrence of two splitting-based iterative methods, Gauss-Seidel and Jacobi. These methods are cheap in terms of computing resources and converge conditionally. Furthermore, in more sophisticated iterative methods like multigrid and Krylov subspace solvers these methods play an important role in reducing and smoothing error components or as preconditioner.



Therefore both methods are only discussed briefly. We should emphasize that these methods are not enhanced enough to handle ill-conditioned systems of equations and convergence rates can be very poor.

#### 4.2.1.1 Jacobi

The choice of  $M = D$  and  $N = -(L + U)$  leads to the Jacobi iteration, where  $D$  contains the diagonal elements of  $A$  and  $L, U$  contain the lower and upper diagonal elements of  $A$  respectively. Hence, write Jacobi as,

$$\mathbf{x}^{k+1} = D^{-1} (L + U) \mathbf{x}^k + D^{-1} \mathbf{b} \quad (4.11)$$

The Jacobi algorithm [14] is described in (3). It is apparent that the Jacobi iteration is very efficient with respect to the storage of matrices and vectors. There is no need for extra storage with respect to the splitting of matrix  $A$ . Hence, the Jacobi method is a memory efficient algorithm and one iteration costs approximately as much work as a matrix vector product. The Jacobi method converges for any start vector  $\mathbf{x}^0$  if matrix  $A$  is non-singular and  $\rho(M^{-1}N) < 1$ . A proof can be found in [14].

---

#### Algorithm 3 Jacobi algorithm

---

**for**  $i = 1 \dots n$  **do**  
 $x_i^{k+1} = \left( b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^k \right) / a_{ii}$

---

#### 4.2.2 Gauss-Seidel

The Gauss-Seidel (GS) method is given by  $M = D + L$  and  $N = -U$  where  $D, L$  and  $U$  contain the same elements of  $A$  as for the Jacobi method described previously. Write the GS method as,

$$\mathbf{x}^{k+1} = (D + L)^{-1} U \mathbf{x}^k + (D + L)^{-1} \mathbf{b} \quad (4.12)$$

The GS algorithm [14] is described in (3). The GS method is a little bit cheaper in terms of memory because we do not need an extra storage vector for solution  $\mathbf{x}^k$ . However, the order of work remains the same. The method converges for any start vector  $\mathbf{x}^0$  when matrix  $A$  is symmetric and positive definite, i.e.  $\forall i \neq j, A_{ij} = A_{ji}$  and  $\forall \mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq 0, (A\mathbf{x}, \mathbf{x}) > 0$  respectively. The GS iteration scheme is often extended with a relaxation parameter which results into the Successive-over-relaxation (SOR) and Symmetric-successive-over-relaxation schemes. By accounting for the weight of historic information  $(\mathbf{x}^k, \mathbf{x}^{k-1}, \dots, \mathbf{x}^0)$  the iteration scheme can be accelerated. The SOR is defined as,

$$\mathbf{x}^{k+1} = M_\omega^{-1} N_\omega \mathbf{x}^k + M_\omega^{-1} \mathbf{b} \quad (4.13)$$

where,  $M_\omega = D + \omega L$ ,  $N_\omega = (1 - \omega)D - \omega L$  and  $\omega A = M_\omega - N_\omega$ . The SOR algorithm is described in (5). It can be shown that the SOR method converges for  $0 < \omega < 2$  when  $A$  is symmetric and positive definite. A smart choice of  $\omega$  can minimize  $\rho(M_\omega^{-1}N_\omega)$  and speed up the iteration process.

---

**Algorithm 4** Gauss-Seidel algorithm

---

**for**  $i = 1 \dots n$  **do**  
 $x_i^{k+1} = \left( b_i - \sum_{j=1, j \neq i}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1, j \neq i}^n a_{ij}x_j^k \right) / a_{ii}$

---



---

**Algorithm 5** Successive-over-relaxation (SOR) algorithm

---

**for**  $i = 1 \dots n$  **do**  
 $x_i^{k+1} = \omega \left( b_i - \sum_{j=1, j \neq i}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1, j \neq i}^n a_{ij}x_j^k \right) / a_{ii} + (1 - \omega)x_i^k$

---

### 4.3 Preconditioning

In preceding Sections () and () we have seen that the stiffness matrix  $K$  of equation (2.26) will be ill conditioned despite its symmetric and positive-definite properties. Obviously the system of (2.26) can be rewritten in terms of system (4.1). Hence, in this section we will refer to the matrix  $A$  of (4.1) instead of stiffness matrix  $K$ . We have seen in Section () that especially asphaltic materials consists of elements that can vary in stiffness significantly. Hence, the spectral decomposition of  $A$  yields presumably large differences in eigenvalues and therefore the condition number of  $A$  will be relatively high. This means that when utilizing non-customized iterative methods the rate of convergence will probably be low, yielding slow convergence. Preconditioning of the system can be a convenient way of accelerating the iterative method. The philosophy of preconditioning is the reduction of the condition number of the preconditioned system, thus matrix  $A$  of system (4.1). This is best illustrated by an example. Observe a system analogue to equation (4.1). Suppose that matrix  $A$  is ill conditioned, i.e. the condition number is high. We could transform the system as follows,

$$M^{-1}Ax = M^{-1}\mathbf{b} \quad (4.14)$$

where  $M^{-1}$  is the (left) preconditioner. For a smart choice of  $M$  we could achieve a much lower condition number of the system  $M^{-1}$  compared to the condition number of  $A$ . As a result  $M^{-1}A$  could be more solvable for basic iterative methods. Hence, a smaller  $\rho(I - M^{-1}A)$  may result in better convergence rates. In preempt to later sections we will see that Krylov subspace methods benefit highly from preconditioned systems. These methods will play an important role in the development of a large scale algebraic solver.

Unfortunately, choosing a proper  $M$  is a challenging task and in most cases far from trivial. It is apparent that for  $M = A$  we obtain a solution of  $x$  in one

step. Therefore it makes perfect sense to use an approximation for matrix  $A$  as a preconditioner. However, introducing a preconditioner will yield extra work when multiplying  $M^{-1}$  with a vector. So we must choose  $M^{-1}$  such that the amount of work does not increase significantly with respect to the solution method of choice and the system  $M^{-1}A$  still has a favorable distribution of eigenvalues. This Section will discuss three simple choices for preconditioning to be familiarized with this concept. In later sections more sophisticated choices of  $M$  are developed and the use of multigrid and domain decomposition as preconditioners is investigated.

Note that the system of equation (4.14) does not preserve its symmetry for most choices of  $M$ . However, the matrix product  $M^{-1}A$  is selfadjoint for the adapted  $M$ -inner product  $(\mathbf{x}, \mathbf{y})_M \equiv (M\mathbf{x}, \mathbf{y}) = (\mathbf{x}, M\mathbf{y})$ ,  $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ , hence  $(M^{-1}A\mathbf{x}, \mathbf{y})_M = (A\mathbf{x}, \mathbf{y}) = (\mathbf{x}, A\mathbf{y}) = (\mathbf{x}, M(M^{-1}A)\mathbf{y}) = (\mathbf{x}, M^{-1}A\mathbf{y})_M$ . When the  $M$ -inner product is utilized within for instance the CG method of Section (), we regain a symmetric system and the algorithms do not have to be adapted. In later sections it will become clear that the evaluation of  $M$ -inner product is also not necessary and the Euclidean inner product can still be used. Another approach for preserving symmetry is to let  $M = PP^T$ , where  $P$  is a non-singular matrix. The preconditioned system of equation (4.14) transforms into,

$$P^{-1}AP^{-T}\tilde{\mathbf{x}} = P^{-1}\mathbf{b} \quad (4.15)$$

where  $\tilde{\mathbf{x}} = P^{-T}\mathbf{x}$ . Stick to the notation of equation (4.14).

### 4.3.1 Basic iterative methods

The principle behind Gauss-Seidel or Jacobi preconditioners lies within the splitting of matrix  $A$  of system (4.1). Again, we introduce the general splitting of  $A = M - N$ . The basic iterative methods are based on the fixed point iteration  $\mathbf{x}^{k+1} = G\mathbf{x}^k + \mathbf{f}$ . We have seen that in the case of splitting  $G = M^{-1}N = I - M^{-1}A$  and  $\mathbf{f} = M^{-1}\mathbf{b}$ . Obviously preceding fixed point iteration applies also on the linear system  $(I - G)\mathbf{x} = \mathbf{f}$  which for preceding  $G = I - M^{-1}A$  transforms into  $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$ . Hence, the definition of preconditioning and splitting of matrix  $A$  in combination with the fixed point iterations are mathematically equivalent.

#### 4.3.1.1 Diagonal scaling (Jacobi)

One of the most obvious choices of  $M$  based on an approximation of  $A$  is  $M_{ii} = A_{ii}$ ,  $M_{ij} = 0$  for  $i \neq j$ . In other words,  $M = D$  and contains only the main diagonal elements of  $A$ . Multiplying  $M^{-1}$  and  $A$  yields scaling of the main diagonal elements of  $A$  to ones only. A major advantage of this approach is the easy calculation of  $M^{-1}A$ , because  $M$  is a diagonal matrix. However, this method is still performing badly for most ill conditioned systems, though it is a good initial test for benchmarking the response of different numerical methods to preconditioning.

### 4.3.1.2 Gauss-Seidel

Using basic iterative methods like Gauss-Seidel as preconditioner is also common practice. These methods comply to the speed condition of preconditioners. In other words, they are cheap to compute in terms of CPU and memory. Use one sweep of Gauss-Seidel as preconditioner or, to preserve the symmetry of the system, use two sweeps with opposite directions.

## 4.3.2 Incomplete factorization

LU decomposition methods generate a lower triangular matrix  $L$  and upper triangular matrix  $U$  such that  $A = LU$ . The Gaussian elimination of Section (4.1) provides us with such  $L$  and  $U$ . However, because of the fill in, complete decomposition of an arbitrary large, sparse matrix is often expensive in terms of CPU time and memory size. Recall that a good preconditioner must be cheap to apply and still be a reasonable approximation of matrix  $A$ . Hence, it makes sense to use a splitting  $A = LU - R$  where  $R \neq \emptyset$  and  $A \approx LU$ . The simplest form of this incomplete decomposition is zero fill-in LU factorization or in abbreviation ILU(0).

### 4.3.2.1 ILU(0) decomposition

The idea behind ILU(0) decomposition is to find  $L, U$  such that  $A_{ij} = (LU)_{ij}$  for  $A_{ij} \neq 0$  and  $R_{ij} = (LU)_{ij}$  for  $A_{ij} = 0$ . This means that  $R = A - LU$  is zero in the non-zero entries of  $A$ . In general there exist many pairs of  $L$  and  $U$  that satisfy these requirements. Table (6) holds an algorithm that produces one possible ILU(0) decomposition for given  $A$ . Clearly an advantage of using ILU(0) is that the zero pattern of  $A$  determines the number of extra memory positions needed to perform the decomposition.

---

#### Algorithm 6 ILU(0) algorithm

---

```

for  $i = 2 \dots n$  do
  for  $k = 1 \dots i - 1$  and  $(i, k) \in NZ(A)$  do
     $a_{ik} = \frac{a_{ik}}{a_{kk}}$ 
    for  $j = k + 1 \dots n$  and  $(i, j) \in NZ(A)$  do
       $a_{ij} = a_{ij} - a_{ik}a_{kj}$ 

```

---

## 4.4 Krylov subspace methods

In preceding sections we have discussed direct solution methods as well as basic iterative methods. Where direct solution methods break down for ill conditioned systems, the basic iterative methods show low convergence rates. Hence, an extension to the existing basic iterative methods will be needed.

We introduce the Krylov subspace  $\mathcal{K}_m$ ,

$$\mathcal{K}_m(A, \mathbf{v}) = \text{span} \{ \mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \dots, A^{m-1}\mathbf{v} \} \quad (4.16)$$

where  $\forall \mathbf{x} \in \mathcal{K}_m(A, \mathbf{v})$ ,  $\mathbf{x} = \lambda_0\mathbf{v} + \lambda_1 A\mathbf{v} + \dots + \lambda_{m-1} A^{m-1}\mathbf{v}$  with  $\lambda_k \in \mathbb{R}$ . For every basic iterative method we have  $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta \mathbf{d}_k$  where  $M\delta \mathbf{d}_k = \mathbf{r}_k$  with  $M$  an approximation of matrix  $A$ . Substitution leads to  $\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}\mathbf{r}_k$ . Thus starting with an arbitrary initial vector  $\mathbf{x}_0$  the following series can be deduced,

$$\mathbf{x}_0, \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0 \quad (4.17)$$

$$\mathbf{x}_1 = \mathbf{x}_0 + M^{-1}\mathbf{r}_0 \quad (4.18)$$

$$\mathbf{x}_2 = \mathbf{x}_1 + M^{-1}\mathbf{r}_1 \quad (4.19)$$

$$= \mathbf{x}_0 + M^{-1}\mathbf{r}_0 + M^{-1}\mathbf{r}_1 \quad (4.20)$$

$$= \mathbf{x}_0 + 2(M^{-1}\mathbf{r}_0) - M^{-1}A(M^{-1}\mathbf{r}_0) \quad (4.21)$$

Clearly  $\mathbf{x}_k$  can be written as a polynomial of  $M^{-1}\mathbf{r}_0$ . Moreover, all  $\mathbf{x}_i \in \mathbf{x}_0 + \text{span} \{ M^{-1}\mathbf{r}_0, M^{-1}A(M^{-1}\mathbf{r}_0), \dots, (M^{-1}A)^{i-1}(M^{-1}\mathbf{r}_0) \}$  which is in fact a Krylov subspace,  $\mathbf{x}_0 + \mathcal{K}_m(M^{-1}A, M^{-1}\mathbf{r}_0)$ . We observe that the solution is a linear combination of the start vector  $\mathbf{x}_0$  and  $(M^{-1}A)^{i-1}(M^{-1}\mathbf{r}_0)$ . The objective of a Krylov subspace method is to construct a basis  $\{ \mathbf{r}_0, A\mathbf{r}_0, \dots, A^{m-1}\mathbf{r}_0 \}$  to obtain a solution for  $A\mathbf{x} = \mathbf{b}$ . At the base of these techniques lies the Arnoldi's procedure [9] for building an orthonormal basis for a Krylov subspace. This Section will discuss the conjugated gradient method as well as the preconditioned conjugated gradient method.

#### 4.4.1 Conjugated gradient method

The conjugated gradient (CG) method is the most famous Krylov subspace method. The CG method is also well known for its performance with respect to sparse, symmetric positive definite problems. The idea behind the CG method is to construct a solution vector  $\mathbf{x}_k$  that minimizes the error  $\|\mathbf{x} - \mathbf{x}_k\|_2^2$ . An obvious way to minimize the latter is to start searching in the direction for which  $\|\mathbf{x} - \mathbf{x}_k\|_2^2$  is relatively small. We reformulate this into a proper minimization problem [14]. The first iterate  $\mathbf{x}_1$  can be written as  $\mathbf{x}_1 = \alpha_0\mathbf{r}_0$ . Hence we have the following minimization problem,

$$\min_{\alpha_0} \|\mathbf{x} - \mathbf{x}_1\|_2^2 = \quad (4.22)$$

$$\min_{\alpha_0} (\mathbf{x} - \alpha_0\mathbf{r}_0)^T (\mathbf{x} - \alpha_0\mathbf{r}_0) \quad (4.23)$$

Solve (4.22) analytically to obtain an exact solution for  $\alpha_0$ ,

$$\frac{\partial}{\partial \alpha_0} (\mathbf{x}^T \mathbf{x} - 2\alpha_0 \mathbf{r}_0^T \mathbf{x} + \alpha_0^2 \mathbf{r}_0^T \mathbf{r}_0) = 0 \iff \alpha_0 = \frac{\mathbf{r}_0^T \mathbf{x}}{\mathbf{r}_0^T \mathbf{r}_0} \quad (4.24)$$

The expression for  $\alpha_0$  does not provide us with a solution as the unknown vector  $\mathbf{x}$  is what we want to solve for. Hence, we introduce a different norm  $\|\cdot\|_A$  and inner product  $(\cdot, \cdot)_A$  which are defined as,

$$\|\mathbf{x}\|_A^2 = \mathbf{x}^T A \mathbf{x} \quad (4.25)$$

$$(\mathbf{x}, \mathbf{y})_A = \mathbf{x}^T A \mathbf{y} \quad (4.26)$$

Rewrite the minimization problem of (4.22) as,

$$\min_{\alpha_0} \|\mathbf{x} - \mathbf{x}_1\|_A^2 = \quad (4.27)$$

$$\min_{\alpha_0} (\mathbf{x} - \alpha_0 \mathbf{r}_0)^T A (\mathbf{x} - \alpha_0 \mathbf{r}_0) \quad (4.28)$$

So,

$$\alpha_0 = \frac{\mathbf{r}_0^T A \mathbf{x}}{\mathbf{r}_0^T A \mathbf{r}_0} = \frac{\mathbf{r}_0^T \mathbf{b}}{\mathbf{r}_0^T A \mathbf{r}_0} \quad (4.29)$$

Now  $\alpha_0$  provides us with a (local) minimum. Extend [9] this philosophy to a general iteration step,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \quad (4.30)$$

where  $\mathbf{p}_k = \mathbf{r}_k + \beta_{k-1} \mathbf{p}_{k-1}$  is the direction vector. In the first step of the CG algorithm  $\mathbf{p}_0 = \mathbf{r}_0$ , thus we start searching in the direction of the initial residual. In (7) we present the full CG algorithm.

---

**Algorithm 7** Conjugated gradient algorithm

---

Start with  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ ,  $\mathbf{p}_0 = \mathbf{r}_0$

$\mathbf{r}_k \neq 0$

**for**  $k = 0, 1, \dots$  **do**

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T A \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A \mathbf{p}_k$$

$$\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$


---

#### 4.4.2 Preconditioned conjugated gradient method

We have seen in Section (4.3) that for ill-conditioned systems of equations (basic) iterative methods can be improved by using a preconditioner. The same applies for the Krylov subspace methods. The reliability of the Krylov methods depends more on the quality of the preconditioner than on the Krylov subspace accelerators

used [9]. It is apparent that preconditioned conjugated gradient is likely to perform better with a preconditioner when solving ill conditioned systems.

The preconditioner for the CG method must meet a number of requirements. The preconditioner  $M$  must be a symmetric positive definite matrix, as the CG method is designed specifically for symmetric systems and performs best for positive definite matrices. Also, we must find a decent approximation of the system matrix  $A$  and it should be easy to solve  $M\mathbf{y} = \mathbf{s}$  as it will be solved every iteration step. Recall from Section (4.3) that the preconditioned system is of the form  $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$  and that in order to preserve symmetry we introduce the  $M$ -inner product. When we replace the Euclidean inner product with the  $M$ -inner product and rewrite the CG algorithm of table (7) we obtain the preconditioned CG method. This algorithm can be found in (8). Note that just one extra line of computation is added to the original algorithm,  $\mathbf{z}_{k+1} = M^{-1}\mathbf{r}_{k+1}$ , hence preconditioned CG is fairly easy to implement.

---

**Algorithm 8** Preconditioned conjugated gradient algorithm

---

Start with  $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ ,  $\mathbf{z}_0 = M^{-1}\mathbf{r}_0$  and  $\mathbf{p}_0 = \mathbf{z}_0$

$\mathbf{r}_k \neq 0$

**for**  $k = 0, 1, \dots$  **do**

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{z}_k}{\mathbf{p}_k^T A \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A \mathbf{p}_k$$

$$\mathbf{z}_{k+1} = M^{-1} \mathbf{r}_{k+1}$$

$$\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{z}_{k+1}}{\mathbf{r}_k^T \mathbf{z}_k}$$

$$\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$$


---

## 4.5 Multigrid

Multigrid has found its way into many engineering applications because of its agility and performance. In Chapter 2 we have derived the discretization of the virtual work equation based on a finite element approach. Where multigrid has become famous for its performance on solving discretized elliptic equations on structured grids, it is also suitable for solving complicated problems on unstructured grids.

The main philosophy behind multigrid is to solve a set of equations on a fine grid by using an approximation obtained on much coarser grids and transferring it back with some particular iterative process. This approach works two ways, first by reducing the number of equations that need to be solved and second by increasing the grid sizes that can be handled with the same amount of computing resources.

Unfortunately, constructing a working multigrid solver is somewhat more involved than it appears to be. Within the multigrid framework we distinguish four key components that determine the convergence speed, stability and robustness of the multigrid method. We have the coarse grid specification, smoother, restric-

tion operator and prolongation operator. As it is well known how to find the best operators for the discretized elliptic equations, its not so easy for general, linear equations on unstructured grids. After the introduction of multigrid basics, the right components for these type of problems are discussed.

For ease of understanding multigrid equidistant, structured grids are used in the examples.

#### 4.5.1 Basics multigrid (two-grid)

Consider the following discretization  $A_h$  of an arbitrary equation on a grid with spatial mesh size  $h$ ,  $\Omega_h$ :

$$A_h \mathbf{x}_h = \mathbf{b}_h \quad (4.31)$$

If the solution of this equation is approximated by  $\mathbf{x}_h^m$ , the error  $\delta \mathbf{x}_h^m$  and residual  $\mathbf{r}_h^m$  are as follows:

$$\begin{aligned} \delta \mathbf{x}_h^m &: = \mathbf{x}_h - \mathbf{x}_h^m \\ \mathbf{r}_h^m &: = \mathbf{b}_h - A_h \mathbf{x}_h^m \end{aligned}$$

This results in the defect equation which is equivalent to the original equation because  $\mathbf{x}_h = \delta \mathbf{x}_h^m + \mathbf{x}_h^m$ :

$$A_h \delta \mathbf{x}_h^m = \mathbf{r}_h^m$$

If a basic iterative method, like Jacobi or Gauss-Seidel, is used to solve the equation and the error is computed, then it appears that the error becomes smooth after several iteration steps. In that case, the iteration formula can be interpreted as an error averaging process. This error-smoothing is one of the two basic principles of the multigrid approach. The other principle is based on the fact that a quantity that is smooth on a certain grid can also be approximated on a coarser grid. So if the error of the approximation of the solution has become smooth after several relaxation sweeps, then this error can be approximated with a suitable procedure on a coarser grid.

Suppose that the matrix  $A_h$  can be approximated by a more easy to invert matrix  $\hat{A}_h$  then:

$$\hat{A}_h \delta \hat{\mathbf{x}}_h^m = \mathbf{r}_h^m \longrightarrow \mathbf{x}_h^{m+1} = \mathbf{x}_h^m + \delta \hat{\mathbf{x}}_h^m$$

The idea of multigrid is to approximately solve the defect equation on a coarser grid with spatial mesh size, e.g.  $H := 2h$ . Obviously, this will take less time and work than a conventional direct method on a grid with spatial mesh size  $h$ .

$$A_H \delta \hat{\mathbf{x}}_H^m = \mathbf{r}_H^m \quad (4.32)$$



Assume that  $A_H^{-1}$  exists. As  $\mathbf{x}_H^m$  and  $\delta\hat{\mathbf{x}}_H^m$  are grid functions operating on the coarser grid, we introduce two (linear) transfer operators:

$$I_h^H : \mathcal{G}(\Omega_h) \longrightarrow \mathcal{G}(\Omega_H), \quad I_H^h : \mathcal{G}(\Omega_H) \longrightarrow \mathcal{G}(\Omega_h)$$

These functions are necessary to restrict and prolongate the residuals and approximations of the error to different coarser and finer grids. This yields,

$$\begin{aligned} \mathbf{r}_H^m & : = I_h^H \mathbf{r}_h^m, \text{ restrict } \mathbf{r}_h^m \text{ to } \Omega_H \\ \delta\hat{\mathbf{x}}_h^m & : = I_H^h \delta\hat{\mathbf{x}}_H^m, \text{ prolongate } \delta\hat{\mathbf{e}}_H^m \text{ to } \Omega_h \end{aligned}$$

One choice for  $I_h^H$  can be the injection operator. For instance, the residual on a fine grid  $\Omega_h$  will be mapped directly to the coarser grid  $\Omega_H$ . No weighting has been applied. Other operators are based on (full) weighting ( $I_h^H$ ) and linear or bilinear interpolation for  $I_H^h$ . In section (4.5.2) these operators will be described in more detail.

Unfortunately coarse grid correction alone is not enough to obtain a good, smooth approximation of the solution on the fine grid. The operator  $A_h$  has different eigenmodes which correspond to low and high frequency error components in the solution. In general, where restriction reduces the low frequency error components, the prolongation of coarse grid corrections reintroduces high frequency error components on the fine grid [13]. One common approach to reduce the high frequency errors is applying one or more smoothing sweeps before and after the coarse grid correction. These sweeps are known as pre- and post-smoothing. When developing a multigrid method that needs to perform and is robust we need to take a closer look at the eigenmodes of the operator  $A_h$ . The analysis of the spectrum of  $A_h$  can give us an indication on how to choose the smoother as well as the restriction and prolongation operators.

#### 4.5.1.1 Multigrid cycle

The multigrid idea starts from the observation that in a well converged two-grid method (section 4.5.1) it is neither useful nor necessary to solve the coarse grid defect equation (4.32) exactly. Instead, without loss of convergence speed, one may replace  $\delta\hat{\mathbf{x}}_H^m$  by a suitable approximation. A natural way to obtain such an approximation is to apply the two-grid idea to (4.32) again, now employing an ever coarser grid than  $\Omega_H$ .

This is possible, as obviously the coarse grid equation (4.32) is of the same form as the original equation (4.31). If the convergence factor of the two-grid method is small enough, it is sufficient to perform only a few, say  $\gamma$ , two-grid iteration steps to obtain a good enough approximation to the solution of (4.32). This idea can, in a straightforward manner, be applied recursively, using coarser and coarser grids, down to some coarsest grid. On this coarsest grid any solution method may be used (e.g. a direct method or some relaxation-type method if it has sufficiently

good convergence properties on that coarsest grid). In ideal cases, the coarsest grid consists of just one grid point.

For a formal description of multigrid methods use a sequence of coarser and coarser grids  $\Omega_{h_k}$ , characterized by a sequence of mesh sizes  $h_k$ :

$$\Omega_{h_l}, \Omega_{h_{l-1}}, \dots, \Omega_{h_0}$$

The coarsest grid is characterized by the mesh size  $h_0$  whereas the index  $l$  corresponds to the finest grid  $\Omega_h : h = h_l$ . For simplicity, replace the index  $h_k$  by  $k$  in the following. For each  $\Omega_k$ , assume that linear operators

$$A_k : G(\Omega_k) \rightarrow G(\Omega_k), \quad S_k : G(\Omega_k) \rightarrow G(\Omega_k), \quad (4.33)$$

$$I_k^{k-1} : G(\Omega_k) \rightarrow G(\Omega_{k-1}), \quad I_{k-1}^k : G(\Omega_{k-1}) \rightarrow G(\Omega_k) \quad (4.34)$$

are given, where the  $A_k$  correspond to  $\Omega_k$  for  $k = l, \dots, 0$ , and where the original equation (4.31) reads

$$A_l \mathbf{x}_l = \mathbf{b}_l \quad (\Omega_l) \quad (4.35)$$

and is the discrete problem to solve. The operators  $S_k$  denote the linear iteration operator corresponding to given smoothing methods on  $\Omega_k$ . Performing  $\nu$  smoothing steps (applied to any discrete problem of the form  $A_k \mathbf{x}_k = \mathbf{b}_k$  with initial approximation  $\mathbf{x}_k^m$ ) resulting in the approximation  $\bar{\mathbf{x}}_k^m$  will denoted by

$$\bar{\mathbf{x}}_k^m = \text{SMOOTH}^\nu(\mathbf{x}_k^m, A_k, \mathbf{b}_k)$$

Now introduce multigrid cycle, more precisely an  $(l+1)$ -grid cycle, to solve (4.35) for a fixed  $l \geq 1$ . Using the operators  $A_k$  ( $k = l, l-1, \dots, 0$ ) as well as  $S_k, I_k^{k-1}, I_{k-1}^k$  ( $k = l, l-1, \dots, 1$ ), assuming parameters  $v_1, v_2$  (the number of pre- and postsmoothing iterations) and  $\gamma$  to be fixed and starting on the finest grid  $k = l$ , the calculation of a new iterate  $\mathbf{x}_k^{m+1}$  from given approximation  $\mathbf{x}_k^m$  to the solution  $\mathbf{x}_k$  proceed as presented in (9).

The different number of two-grid iterations steps determine the structure of a multigrid cycle. Possibilities are the V-cycle ( $\gamma = 1$ ), W-cycle ( $\gamma = 2$ ) or F-cycle ( $\gamma = \gamma_k$ ). The main differences between these approaches are the number of pre- and post-smoothing steps and the different number of coarser grids used.

### 4.5.2 Multigrid Components

As described above, the following multigrid components have to be chosen,

- Coarse grid specification
- Smoother

---

**Algorithm 9 Multigrid cycle**  $\mathbf{x}_k^{m+1} = \text{MGCYC}(k, \gamma, \mathbf{x}_k^m, A_k, \mathbf{b}_k, v_1, v_2)$ 


---

**1. Presmoothing**

- Compute  $\bar{\mathbf{x}}_k^m$  by applying  $v_1(\geq 0)$  smoothing steps to  $\mathbf{x}_k^m$   
 $\bar{\mathbf{x}}_k^m = \text{SMOOTH}^{v_1}(\mathbf{x}_k^m, A_k, \mathbf{b}_k)$

**2. Coarse grid correction**

- Compute the defect  $\bar{\mathbf{r}}_k^m := \mathbf{b}_k - A_k \bar{\mathbf{x}}_k^m$
- Restrict the defect  $\bar{\mathbf{r}}_{k-1}^m := I_k^{k-1} \bar{\mathbf{r}}_k^m$
- Compute an approximate solution  $\delta \bar{\mathbf{x}}_{k-1}^m$  of the defect equation on  $\Omega_{k-1}$   
 $A_{k-1} \delta \mathbf{x}_{k-1}^m = \bar{\mathbf{r}}_{k-1}^m$  (\*)  
 by

**If**  $k = 1$  **then** use a direct or fast iterative solver for (\*)

**If**  $k > 1$  **then** solve (\*) approximately by performing  $\gamma(\geq 1)$   $k$ -grid cycles using the zero grid function as a first approximation

$$\delta \mathbf{x}_{k-1}^m = \text{MGCYC}^\gamma(k-1, \gamma, 0, A_{k-1}, \bar{\mathbf{r}}_{k-1}^m, v_1, v_2)$$

- Interpolate the correction  $\delta \mathbf{x}_k^m := I_{k-1}^k \delta \mathbf{x}_{k-1}^m$
- Compute the corrected  
 approximation on  $\Omega_k$   $\mathbf{x}_k^{m, \text{after CGC}} = \bar{\mathbf{x}}_k^m + \delta \mathbf{x}_k^m$

**3. Postsmoothing**

- Compute  $\mathbf{x}_k^{m+1}$  by applying  $v_2(\geq 0)$  smoothing steps to  $\mathbf{x}_k^{m, \text{after CGC}}$   
 $\mathbf{x}_k^{m+1} = \text{SMOOTH}^{v_2}(\mathbf{x}_k^{m, \text{after CGC}}, A_k, \mathbf{b}_k)$
- 

- Restriction operator
- Prolongation operator

**4.5.2.1 Coarse grid specification**

For structured, equidistant grids it is straightforward how to find the coarser grid. Where the finest grid has mesh size  $h$ , we could have  $H := 2h$  as a logical choice. However, the problems that we observe within structural mechanics do not have structured grids due to the irregular shapes of the materials. In this case we have several options. The first option is to predetermine a sequence of coarser grids before any computations are done. Either construct a coarser grid by taking a subset of the original grid nodes as the coarse grid nodes, as illustrated in figure (3.5). Or use a mesh generator to obtain a coarser grid with much less grid nodes than the fine grid. These methods are easy to implement and allow for an easy choice of the restriction and prolongation operators. However, these methods can be a problem when the grid is cracking as the original cohesion will disappear. Note that for each load step a new sequence of coarse grids needs to be constructed.

The second option is to use algebraic multigrid (AMG). In this case we do not have to define the coarser grid explicitly. AMG does not distinguish between grid points but operates on subsets of the solution vector  $\mathbf{x}$ , and corresponding entries within  $A$ . Another advantage of AMG is that both the restriction and the prolongation operators are part of the AMG algorithm. But also AMG has tailored (grid) operators and is not a black-box multigrid solver.

#### 4.5.2.2 Smoother

The smoother has two vital tasks. First, it acts as a smoother. The high-frequency error components on the fine grids are smoothed down. Secondly, locally it computes a new approximation for the solution  $\mathbf{x}_H$  of  $A_H \mathbf{x}_H = \mathbf{b}_H$ , with  $H$  the coarsest spatial mesh size.

Multigrid methods are motivated by the fact that many iterative methods, especially if they are applied to elliptic problems, have a smoothing effect on the error between an exact solution and a numerical approximation. A smooth discrete error can be well represented on a coarser grid, where its approximation is much cheaper.

There are many iterative solvers that can act as a smoother. When we have large systems of equations we want the smoother to be cheap in terms of computing memory and CPU power. Basic iterative methods like Gauss-Seidel and Jacobi are an obvious choice but often lack efficiency because of the complexity or conditioning of the system. We need many smoothing sweeps to see some effect on the smoothness of the error. More sophisticated, krylov subspace, methods like CG and GMRES are a good choice for they are cheap and can handle a variety of problems. Finding the right smoother is a trial and error process but with the aid of standard software like LAPACK and matlab it is not hard to try different methods.

#### 4.5.2.3 Restriction operator

The restriction operator, together with the prolongation operator, is probably the most important multigrid component. The restriction operator has two tasks. First, it transfers the error from one grid to a coarser grid. Second, it reduces the low frequency error components which correspond to the slow converging parts of the solution. When using structured grids the restriction operator is often a weighting process. This is illustrated in figure (4.1) where the small black dots represent the nodes of the fine grids which are being restricted to the encircled nodes on the coarse grid. The distance between the grid nodes determines the weight of restriction. The weights at each coarse grid nodes must add up to one.

Similar techniques can also be applied at an unstructured FE grid.

#### 4.5.2.4 Prolongation operator

After computing the exact or approximate solution of the discrete equations on the coarse grids, the solutions need to be interpolated back to the fine grid and added to the fine-grid solutions. A natural prolongation operator is the scaled

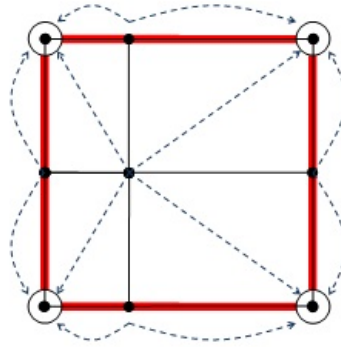


Figure 4.1: Restriction of neighbouring fine grid nodes to coarse grid nodes.

transpose of the restriction operator. This means that the same weights are applied as with the restriction operator. This operator is identical to constant interpolation in the coordinate direction of the component and bilinear interpolation in the other coordinate directions.

## 4.6 Domain decomposition

In preceding sections several techniques for handling large, sparse matrices that result from a finite difference or finite element discretization have been discussed. Also, it became clear that one of the bottlenecks of traditional direct solution methods and (basic) iterative methods lies within the hardware needed for computations instead of the mathematics behind it. As hardware is improving over the recent years memory size still dictates the maximum problem size that can be handled. Also due to the 'memory wall' and physical limitations, there is not enough bandwidth between the CPU(s) and memory to process the huge amounts of data needed for heavy computations. Unless groundbreaking new technology like MRAM (micro RAM) - which has significant higher read/write capabilities, bandwidth and storage size - becomes available, clusters of computing units shall be our method of choice. Hence, when handling large systems of equations on parallel machines mathematical methods that embed parallelism into its algorithms have to be developed. With the availability of parallel software packages like SCALAPACK the parallelization of linear algebra operations has become less difficult. So Krylov subspace methods like the conjugated gradient algorithm can be parallelized with relative ease but still need (parallel) preconditioners for good convergence rates.

More involved is the parallelization of direct solution methods, but as we have seen in Section (3.2) there are excellent parallel software packages available which can do the job.

In previous section we have also introduced the multigrid method, which is a very advanced tool to reduce the number of degrees of freedom in our system and still keep a decent level of accuracy. However, we have also seen that developing a

multigrid method is very involved and we have to overcome many difficulties like proper restriction and prolongation operators, coarse grid selection, good smoothers and a smart choice for the multigrid cycle. Also the finite element discretization can produce 'difficult' grids as (standard) multigrid operators are very sensitive to local grid refinements and grid stretching. With respect to the parallelisation of multigrid, the scattering of elements or slicing of the domain for assignment to the computing nodes is not a trivial operation.

The complexity of multigrid and the scaling issues with Krylov subspace algorithms leads us to domain decomposition methods. The domain decomposition methods combine techniques for solving PDE's, standard linear algebra, mathematical analysis and techniques from graph theory [9]. The idea behind domain decomposition is very straightforward. In the case of Section (2) we have a domain that consists of a number of elements that make up the mesh. Instead of using the full system matrix  $A$  of equation (4.1) the computations are only done on subdomains, i.e. submatrices of  $A$ . Each subdomains contains a set of elements and the unification of these subdomains (sets) returns the full domain (matrix). Clearly, mapping these subdomains to the computing nodes available is a trivial operation. And of course many techniques can be found that will ensure a proper workload for given subdomains and computing resources available. Domain decomposition is parallel by nature and is therefore an ideal candidate for preconditioning of for instance Krylov subspace algorithms.

This section will discuss the element-by-element domain decomposition where direction solution techniques as well as iterative methods will be taken into account.

#### 4.6.1 Block-Gaussian elimination

When introducing the domain decomposition method, visualization can clarify the discussion of the different mathematical choices. Hence, the same example will be referred to in upcoming sections. Consider the domain of figure (4.2). In this figure rectangular domain  $\Omega$  is given which consists of two subdomains,  $\Omega_1$  and  $\Omega_2$ . In this case  $\Omega = \Omega_i$  where  $\Omega_1 \cap \Omega_2 \neq \{j_{13}, j_{14}, j_{15}\}$  with  $j_k$  denotes the node  $k$  and in general  $\Omega = \Omega_i$  with  $s$  subdomains. Here domain  $\Omega$  holds 15 nodes and 8 elements. Each subdomain  $\Omega_i$  contains 9 nodes, hence they have an overlap of 3 nodes on border  $\Gamma_{12}$ .

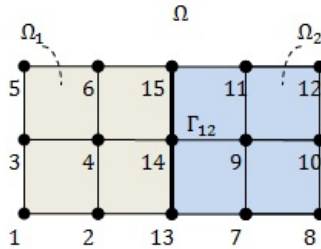


Figure 4.2: Domain decomposition rectangular domain  $\Omega$ .

When utilizing the standard finite element approach we would have a weak formulation (Section (2.1)) of certain unknown values  $\mathbf{x}$  on the domain  $\Omega$ . Each element contributes to the element stiffness matrix  $A$  and we would observe the following system of equations,

$$A\mathbf{x} = \mathbf{f} \quad (4.36)$$

Instead of using the whole domain  $\Omega$  we solve on the subdomains  $\Omega_1$  and  $\Omega_2$  or in general  $\Omega_i, \forall i \in \{1, 2, \dots, s\}$ . Rewrite equation (4.36) as,

$$\begin{pmatrix} B_1 & & & E_1 \\ & B_2 & & E_2 \\ & & \ddots & \vdots \\ & & & B_s & E_s \\ F_1 & F_2 & \dots & F_s & C \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_s \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_s \\ \mathbf{g} \end{pmatrix} \quad (4.37)$$

where  $\mathbf{x}_i$  is the subvector of unknowns that are interior to subdomain  $\Omega_i$  and the interface nodes are placed at the end of matrix  $A$ . For convenience simplify this system as,

$$A \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} \text{ with } A = \begin{pmatrix} B & E \\ F & C \end{pmatrix}. \quad (4.38)$$

Thus  $E$  represents the subdomain to interface coupling seen from the subdomains and  $F$  represents the interface to subdomain coupling seen from the interface nodes. Because the systems are linear independent solve (6.6) with different steps. First,

$$\mathbf{x} = B^{-1}(\mathbf{f} - E\mathbf{y}) \quad (4.39)$$

then,

$$\begin{aligned} F\mathbf{x} + C\mathbf{y} &= \mathbf{g} \\ FB^{-1}(\mathbf{f} - E\mathbf{y}) + C\mathbf{y} &= \mathbf{g} \\ (C - FB^{-1}E)\mathbf{y} &= \mathbf{g} - FB^{-1}\mathbf{f} \\ S\mathbf{y} &= \mathbf{g}'. \end{aligned} \quad (4.40)$$

Obviously if we solve for  $\mathbf{y}$ , by means of back substitution we obtain a solution for  $\mathbf{x}$ . In this way we only need to solve for the nodes on the interfaces to obtain a solution for the nodes within the inner of the subdomains. We call the matrix  $S$  the **Schur complement** of matrix  $A$ . This approach leads to the Block-Gaussian Elimination algorithm of table (10). Here  $E' = B^{-1}E$  and  $\mathbf{f}' = B^{-1}\mathbf{f}$ . In practice solve  $BE' = E$  and  $B\mathbf{f}' = \mathbf{f}$  by exploiting the structure of  $B$ , therefore solve for each subdomain  $\Omega_i$ ,  $B_iE'_i = E_i$  and  $B_i\mathbf{f}'_i = \mathbf{f}_i$ . However, there are two ways in which system  $S\mathbf{y} = \mathbf{g}'$  can be solved. Either by a direct solution method or with an iterative method. When using a direct method  $S^{-1}$  will have to be computed

explicitly. For Krylov subspace methods it is only necessary to perform matrix-vector multiplications, i.e.  $S\mathbf{v} = \mathbf{w}$ . Such operations can be evaluated in three steps, first compute  $\mathbf{v}' = E\mathbf{v}$ , then solve  $B\mathbf{z} = \mathbf{v}'$  and finally compute  $\mathbf{w} = C\mathbf{v} - F\mathbf{z}$ . Again, a linear system that involves matrix  $B$  translates into  $s$ -independent linear systems. It is also possible to apply a preconditioner to matrix  $S$ , in [9] several options for preconditioner  $M^{-1}$  can be found. One can also proof that a Krylov subspace method applied to the preconditioned reduced linear system  $M_s^{-1}S\mathbf{y} = M_s^{-1}\mathbf{g}'$  yields exact the same result as the Krylov subspace method applied to the full preconditioned system  $M_A^{-1}A(\mathbf{x} \ \mathbf{y})^T = M_A^{-1}(\mathbf{f} \ \mathbf{g})^T$  where both preconditioners are based on incomplete LU decomposition.

---

**Algorithm 10** Block-Gaussian elimination

---

Solve  $BE' = E$  and  $B\mathbf{f}' = \mathbf{f}$

Compute  $\mathbf{g}' = \mathbf{g} - F\mathbf{f}'$

Compute  $S = C - FE'$

Solve  $S\mathbf{y} = \mathbf{g}'$

Compute  $\mathbf{x} = \mathbf{f}' - E'\mathbf{y}$

---

#### 4.6.2 Multiplicative Schwarz

With the Block-Gaussian elimination algorithm of previous section we have introduced the Schur complement and proposed several methods of solving the resulting reduced system of equations. Another way of domain decomposition is to start from iterating over the subdomains in the first place. Recall the rectangular domain of figure (4.2). We could solve the weak formulation of the finite element discretization on each subdomain separately and use the most recent boundary values wherever possible. For a sequential algorithm this will implicate that we iterate over the subdomains one-by-one. When a solution for a subdomain is obtained we update the boundary values and proceed to the next (neighboring) subdomain. This method translates into the multiplicative Schwarz sweep. Before we look into the algorithm we have to introduce some (sub) domain operators first. The operators show a strong resemblance with the multigrid operators of Section (4.5.2). Recall the reordering of the original system resulting from an arbitrary finite element discretization that yielded equation (6.6). We now define restriction operator  $R_i$  which restricts global vector  $\mathbf{x}$  to subdomain  $\Omega_i$ . Restriction operator  $R_i$  is a  $n_i \times n$  matrix of zeros and ones. In algorithm (4.1) we present the operator  $R_1$  for the subdomain  $\Omega_1$  of the example of figure (4.2). In this example a vector holding all mesh nodes is mapped to a local vector holding only those nodes that lie in  $\Omega_1$ . The inverse operation is the prolongation of local vector  $x_i$  of subdomain  $\Omega_i$  to the domain  $\Omega$ , hence we introduce the prolongation operator  $R_i^T$  which is indeed the transpose of the restriction operator  $R_i$ .

The domain operators  $R_i$  and  $R_i^T$  can work on the matrix  $A$  as well yielding a Galerkin type operator  $A_i$  of dimension  $n_i \times n_i$ ,



$$\begin{pmatrix} 1 & \emptyset & 0 & & & \\ \emptyset & \ddots & \emptyset & \emptyset & & \\ 0 & \emptyset & 1 & & & \\ & & & 1 & 0 & 0 \\ & \emptyset & & 0 & 1 & 0 \\ & & & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_6 \\ x_7 \\ \vdots \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_6 \\ x_{13} \\ x_{14} \\ x_{15} \end{pmatrix}$$

Table 4.1: Restriction operator

$$A_i = R_i A R_i^T. \quad (4.41)$$

Now solve locally with  $A_i$  and prolongate the local solution to the whole domain and introduce the multiplicative Schwarz sweep in (11). In this algorithm  $\mathbf{x}^{m+1}$  represents the new global solution vector  $\mathbf{x}^m$  after solving for each subdomain  $i$ .

---

**Algorithm 11** Multiplicative Schwarz sweep

---

**for**  $i = 1, \dots, s$  **do**

$$\mathbf{x}^{m+1} = \mathbf{x}^m + R_i^T A_i^{-1} R_i (\mathbf{f} - A \mathbf{x})$$


---

#### 4.6.2.1 Multiplicative Schwarz preconditioning

Multiplicative Schwarz can also be utilized as a preconditioner for Krylov subspace methods. We observe that the error at each subiteration  $i$  equals  $\delta \mathbf{x}^i = \mathbf{x} - \mathbf{x}^i$  and the corresponding residual equals  $\mathbf{r}^i = \mathbf{b} - A \mathbf{x}^i$  where  $\mathbf{x}$  is the exact solution. Hence, the defect equation  $A \delta \mathbf{x}^i = \mathbf{r}^i$  yields,

$$\delta \mathbf{x}^{i+1} = \delta \mathbf{x}^i - R_i^T A_i^{-1} R_i A \delta \mathbf{x}^i. \quad (4.42)$$

Applying one multiplicative Schwarz sweep, start with given  $\mathbf{x}^0$  and  $P_i = R_i^T A_i^{-1} R_i A$  gives  $\delta \mathbf{x}^i = (I - P_i) \delta \mathbf{x}^{i-1}$ . After  $s$  subiterations we obtain,

$$\delta \mathbf{x}^s = (I - P_s) (I - P_{s-1}) \cdots (I - P_1) \delta \mathbf{x}^0. \quad (4.43)$$

For reasons of convenience let  $Q_s = (I - P_s) (I - P_{s-1}) \cdots (I - P_1)$ . Write  $\delta \mathbf{x}^s = Q_s \delta \mathbf{x}^0$  which is equivalent to  $\mathbf{x}^s - \mathbf{x} = Q_s (\mathbf{x}^0 - \mathbf{x})$  and can be recasted into,

$$\mathbf{x}^s = Q_s \mathbf{x}^0 + (I - Q_s) \mathbf{x}. \quad (4.44)$$

Moreover, observe that one multiplicative Schwarz sweep is in fact a global fixed point iteration,  $\mathbf{x}^{m+1} = G \mathbf{x}^m + \mathbf{f}$ . It is apparent that the expression of equation 4.44 can be plugged in and  $G = Q_s$  and  $\mathbf{f} = (I - Q_s) \mathbf{x}$  are obtained. Recall that  $\mathbf{x}^{m+1} =$

$G\mathbf{x}^m + \mathbf{f}$  is equal to solving the preconditioned system  $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$  with  $G = I - M^{-1}A$  and  $\mathbf{f} = M^{-1}\mathbf{b}$  hence,  $M^{-1}A = I - Q_s$  and  $M^{-1}\mathbf{b} = (I - Q_s)A^{-1}\mathbf{b}$ .

Introduce two algorithms in 12 and 13 for determining  $M^{-1}A\mathbf{v}$  and  $M^{-1}\mathbf{v}$ . Therefore use multiplicative Schwarz as preconditioner within Krylov subspace methods that solve  $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$ . To preserve symmetry, apply two sweeps over the subdomains, first  $i = 1, \dots, s$  and then  $i = s, s-1, \dots, 1$ .

---

**Algorithm 12** Multiplicative Schwarz preconditioner

---

Input:  $\mathbf{v}$ , Output:  $\mathbf{z} = M^{-1}\mathbf{v}$ .  
 $\mathbf{z}_1 = R_1^T A_1^{-1} R_1 \mathbf{v}$   
**for**  $i = 2, \dots, s$  **do**  
 $\mathbf{z}_i = \mathbf{z}_{i-1} + R_i^T A_i^{-1} R_i (\mathbf{v} - A\mathbf{z}_{i-1})$

---



---

**Algorithm 13** Multiplicative Schwarz preconditioner operator

---

Input:  $\mathbf{v}$ , Output:  $\mathbf{z} = M^{-1}A\mathbf{v}$ .  
 $\mathbf{z}_1 = R_1^T A_1^{-1} R_1 A \mathbf{v}$   
**for**  $i = 2, \dots, s$  **do**  
 $\mathbf{z}_i = \mathbf{z}_{i-1} + R_i^T A_i^{-1} R_i A (\mathbf{v} - \mathbf{z}_{i-1})$

---

### 4.6.3 Additive Schwarz

The additive Schwarz procedure is similar to a block-Jacobi iteration and consists of updating all the new (block) components from the same residual. It differs from multiplicative Schwarz because the components in each subdomain are not updated until the whole sweep over all subdomains has finished. In (14) we introduce the Additive Schwarz algorithm. In this algorithm  $\mathbf{x}^{m+1}$  represents the new global solution vector  $\mathbf{x}^m$  after solving for all subdomains  $i$ .

---

**Algorithm 14** Additive Schwarz sweep

---

**for**  $i = 1, \dots, s$  **do**  
 $\delta_i = R_i^T A_i^{-1} R_i (\mathbf{b} - A\mathbf{x})$   
 $\mathbf{x}^{m+1} = \mathbf{x}^m + \sum_{i=1}^s \delta_i$

---

Analogue to the multiplicative Schwarz define an additive Schwarz preconditioner. In (15) and (16) we introduce the two algorithms for determining  $M^{-1}A\mathbf{v}$  and  $M^{-1}\mathbf{v}$ .

## 4.7 Deflation

In preceding sections we have discussed several methods to reduce computation times and memory usage. All these methods use a mapping of the original grid to a subset of elements. With this subset of elements we try to find a solution for the

**Algorithm 15** Additive Schwarz preconditionerInput:  $\mathbf{v}$ , Output:  $\mathbf{z} = M^{-1}\mathbf{v}$ .**for**  $i = 1, \dots, s$  **do**

$$\mathbf{z}_i = R_i^T A_i^{-1} R_i \mathbf{v}$$

$$\mathbf{z} = \sum_{i=1}^s \mathbf{z}_i$$

**Algorithm 16** Additive Schwarz preconditioner operatorInput:  $\mathbf{v}$  Output:  $\mathbf{z} = M^{-1}A\mathbf{v}$ .**for**  $i = 1, \dots, s$  **do**

$$\mathbf{z}_i = R_i^T A_i^{-1} R_i A \mathbf{v}$$

$$\mathbf{z} = \sum_{i=1}^s \mathbf{z}_i$$

whole grid. The major benefit of working with smaller subsets is the reduction of the problem sizes. With smaller grids the resulting matrices become smaller and hence in general computation times and memory usage will decrease. Another advantage of smaller systems is that bigger problems can be handled with the same amount of CPU power and memory. Hence, methods like multigrid and domain decomposition are utilized when developing a solver for large, sparse systems of equations.

Despite the reduction of number of equations we do have to take into account the robustness of these numerical methods. For the ill-conditioned systems that we observe in structural mechanics two-level multigrid and domain decomposition are not sufficient. Unfavorable eigenvalues are causing these methods to break down or could slow down the iteration process. We have seen that the matrices resulting from the FE discretization will be ill-conditioned in the majority of cases that are being considered, especially when working with asphaltic materials. Therefore it is very important that the solver that we develop is numerically robust.

This section will introduce the deflation method. This method has been developed to effectively treat (extremely) unfavorable eigenvalues that delay the convergence of iterative methods [12]. By treating or filtering out these eigenvalues one hopes to achieve a better conditioned system and hence better convergence rates when using iterative solution methods. The deflation method is often used in combination with preconditioned Krylov subspace methods like CG which we have discussed in Section (4.4). Therefore, a feasible combination could be a Krylov subspace method applied to a deflated, preconditioned system where we could use domain decomposition or multigrid as preconditioner. This would unify the scalability properties of the multigrid and domain decomposition methods with the robustness of the deflation method. In later section we will explore these combinations of solvers.

**4.7.1 Deflation definitions**

Analogue to preceding chapters we want to solve a linear system of equations that may result from a FE discretization.

$$A\mathbf{x} = \mathbf{b} \quad (4.45)$$

The SPSD matrix  $A$  has dimensions  $n \times n$  and vectors  $\mathbf{x}, \mathbf{b}$  have dimension  $n$ . We assume that the matrix  $A$  has  $d \geq 0$  zero-eigenvalues. We introduce the following deflation operators.

$$\begin{aligned} Z &\in \mathbb{R}^{n \times k}, \quad k < n - d, \quad \text{deflation subspace matrix} \\ P &= I - AQ \in \mathbb{R}^{n \times n}, \quad \text{deflation matrix} \\ Q &= ZE^{-1}Z^T \in \mathbb{R}^{n \times n}, \quad \text{correction matrix} \\ E &= Z^T AZ \in \mathbb{R}^{k \times k}, \quad \text{inversion Galerkin matrix or coarse matrix} \end{aligned}$$

The deflation subspace matrix  $Z \in \mathbb{R}^{n \times k}$  is the key matrix within the deflation theory. The  $Z$  matrix has  $k$  columns which are the deflation vectors. The general idea behind deflation is choosing the eigenvectors corresponding to the unfavorable eigenvalues as deflation vectors. From the definition of the deflation operators it is clear that this will result into a mapping of matrix  $A$  onto a smaller matrix  $E$  that is based on the eigenvectors of the unfavorable spectrum. Return to the space of the original matrix  $A$  with the correction matrix  $Q$ . The choice of the deflation vectors determines the effectiveness of the deflation method. Hence, in the next section possible choices of  $Z$  are discussed. In general, choose  $Z$  such that  $\mathcal{N}(A) \subsetneq \mathcal{R}(Z)$  where  $\mathcal{N}(A)$  and  $\mathcal{R}(Z)$  represent the null space of  $A$  and column space of  $Z$  respectively. This implies that  $E$  is non-singular.

In [12] many properties of the deflation operators have been derived and proven. The relevant properties of deflation:

1.  $E^T = E$
2.  $Q^T = Q = QAQ$
3.  $QAZ = Z$
4.  $PAQ = 0_{n,n}$
5.  $P^2 = P$
6.  $AP^T = PA$
7.  $(I - P^T)\mathbf{x} = Q\mathbf{b}$
8.  $PAZ = 0_{n,k}$
9.  $P^T Z = 0_{n,k}$

We observe that  $E, Q$  are symmetric and although  $\mathbf{x}$  is unknown we can still compute  $(I - P^T)\mathbf{x}$ . Moreover,  $PA$  has  $k + d$  zero-eigenvalues and  $P$  has only zero and unit eigenvalues and therefore positive semi definite and hence  $PA$  is symmetric positive semi definite.

### 4.7.2 Deflated CG method

As the CG method is one of the most commonly used iterative methods for solving PSD systems we introduce the deflated CG method. We should emphasize however that the deflation method can be applied to any linear system and the same theory can therefore be extended to any iterative solver.

Solve  $x$  of equation 4.45 by using a splitting,

$$\mathbf{x} = (I - P^T) \mathbf{x} + P^T \mathbf{x} \quad (4.46)$$

The term  $(I - P^T) \mathbf{x}$  can be substituted by the expression  $Q\mathbf{b}$  according to property 7 of preceding Section. Rewrite equation 4.46 as follows,

$$\begin{aligned} \mathbf{x} &= Q\mathbf{b} + P^T \mathbf{x} \\ A\mathbf{x} &= AQ\mathbf{b} + AP^T \mathbf{x} \\ \mathbf{b} &= AQ\mathbf{b} + PA\mathbf{x} \\ P\mathbf{b} &= PA\mathbf{x} \end{aligned} \quad (4.47)$$

We call the solution  $\mathbf{x}$  of equation 4.47 the deflated solution as it could contain components of the null space of  $PA$ . Hence, it does not represent the real solution of equation 4.45. Compute the real solution with  $\mathbf{x} = Q\mathbf{b} + P^T \hat{\mathbf{x}}$ , where  $\hat{\mathbf{x}}$  is the deflated solution of equation 4.47.

In (17) we introduce the deflated CG algorithm for solving equation 4.45.

---

**Algorithm 17** Deflated CG solving  $A\mathbf{x} = \mathbf{b}$ 


---

Select  $\mathbf{x}_0$ . Compute  $\mathbf{r}_0 = (\mathbf{b} - A\mathbf{x}_0)$ , set  $\hat{\mathbf{r}}_0 = P\mathbf{r}_0$  and  $\mathbf{p}_0 = \hat{\mathbf{r}}_0$

**for**  $j = 0, 1, \dots$  until convergence **do**

$$\hat{\mathbf{w}}_j = PA\mathbf{p}_j$$

$$\alpha_j = \frac{(\hat{\mathbf{r}}_j, \hat{\mathbf{r}}_j)}{(\hat{\mathbf{w}}_j, \mathbf{p}_j)}$$

$$\hat{\mathbf{x}}_{j+1} = \hat{\mathbf{x}}_j + \alpha_j \mathbf{p}_j$$

$$\hat{\mathbf{r}}_{j+1} = \hat{\mathbf{r}}_j - \alpha_j \hat{\mathbf{w}}_j$$

$$\beta_j = \frac{(\hat{\mathbf{r}}_{j+1}, \hat{\mathbf{r}}_{j+1})}{(\hat{\mathbf{r}}_j, \hat{\mathbf{r}}_j)}$$

$$\mathbf{p}_{j+1} = \hat{\mathbf{r}}_{j+1} + \beta_j \mathbf{p}_j$$

$$\mathbf{x} = Q\mathbf{b} + P^T \hat{\mathbf{x}}_{j+1}$$


---

### 4.7.3 Deflated preconditioned CG method

Analogue to the deflated non-preconditioned CG method introduce the deflated preconditioned CG method. We introduce a SPD matrix  $M$  and rewrite equation 4.47 as,

$$M^{-1}PA\hat{\mathbf{x}} = M^{-1}P\mathbf{b} \quad (4.48)$$

In (18) we introduce the deflated preconditioned CG method for solving equation 4.48. Note that a deflation technique applied to a preconditioned system is equivalent to preconditioning of a deflated system (equation 4.48).

---

**Algorithm 18** Deflated preconditioned CG solving  $A\mathbf{x} = \mathbf{b}$ 


---

Select  $\mathbf{x}_0$ . Compute  $\mathbf{r}_0 = (\mathbf{b} - A\mathbf{x}_0)$ , set  $\hat{\mathbf{r}}_0 = P\mathbf{r}_0$  and  $\mathbf{p}_0 = \hat{\mathbf{r}}_0$

Solve  $M\mathbf{y}_0 = \hat{\mathbf{r}}_0$  and set  $\mathbf{p}_0 = \mathbf{y}_0$

**for**  $j = 0, 1, \dots$  until convergence **do**

$$\hat{\mathbf{w}}_j = PA\mathbf{p}_j$$

$$\alpha_j = \frac{(\hat{\mathbf{r}}_j, \hat{\mathbf{r}}_j)}{(\hat{\mathbf{w}}_j, \mathbf{p}_j)}$$

$$\hat{\mathbf{x}}_{j+1} = \hat{\mathbf{x}}_j + \alpha_j \mathbf{p}_j$$

$$\hat{\mathbf{r}}_{j+1} = \hat{\mathbf{r}}_j - \alpha_j \hat{\mathbf{w}}_j$$

Solve  $M\mathbf{y}_{j+1} = \hat{\mathbf{r}}_{j+1}$

$$\beta_j = \frac{(\hat{\mathbf{r}}_{j+1}, \mathbf{y}_{j+1})}{(\hat{\mathbf{r}}_j, \mathbf{y}_j)}$$

$$\mathbf{p}_{j+1} = \mathbf{y}_{j+1} + \beta_j \mathbf{p}_j$$

$$\mathbf{x} = Q\mathbf{b} + P^T \hat{\mathbf{x}}_{j+1}$$


---

#### 4.7.4 Deflation vectors

Choosing the right deflation vectors is difficult. We want the deflation subspace matrix  $Z$  to contain the eigenvectors that belong to the smallest eigenvalues as they correspond to the slow converging parts of the solution vector. When taken out of the system only the faster converging components of the solution vector remain and faster convergence speeds are expected. However, in most realistic applications the calculation of the eigenvectors is either not feasible or very time consuming. Another difficulty is that dense eigenvectors result in a full, coarse matrix  $E$  which becomes difficult to invert. Obviously, having real eigenvectors and corresponding eigenvalues as deflation vectors is not common practice. In summary, the deflation method should satisfy the next requirements in the ideal case [12],

- the deflation-subspace matrix  $Z$ , is sparse
- the deflation vectors approximate the eigenvectors corresponding to the unfavorable eigenvalues
- the cost of constructing deflation vectors is relatively low
- the method has favorable parallel properties
- the approach can easily be implemented in an existing PCG code

The choice of  $Z$  strongly depends on the application, therefore there is no optimal choice that leads to the best results for all applications. The deflation subspace matrix will have to be tailored to the specific situation. In [12] a number of choices for deflation vectors are being considered,

- Approximated eigenvector deflation
- Recycling deflation
- Subdomain deflation
- Multigrid and multilevel deflation vectors

Each method will not be discussed separately as it is too premature for this phase of the research. However, subdomain deflation appears to be the most feasible approach. The domain is divided into a number of subdomains and each subdomain corresponds to one or more deflation vectors. This approach shows a strong resemblance with the domain decomposition methods of Section 4.6. The deflation subspace matrix  $Z$  is sparse and consists of only ones and zeros. The number of deflation vectors is relatively small ( $k \ll n$ ) and they appear to approximate the eigenspace associated with the unfavorable eigenvalues. The deflation vectors correspond to the subdomain and are therefore easy to identify. Moreover, subdomain deflation is parallelizable as the deflation vectors are disjoint.

#### 4.7.5 Example of solving deflated virtual work equation

In order to demonstrate the potential power of deflation methods DCG and DPCG were applied to a very simple strain-stress test. Consider the two cubes of figure 4.4. The two cubes represent the same body but in different stages of a compression test. The left cube is undeformed and represents a body containing two materials. The blue material on the outside is rubber like and has only elastic material properties. The yellow material in the centre and core of the body is incompressible and one could think of steel or stone. A pressure is being applied at the top of the cube and is forcing the cube downwards within several load steps (static mechanics).

The right cube represents the body after the last load step. Clearly the elastic material has been pressed down and the incompressible material is still in place, undeformed.

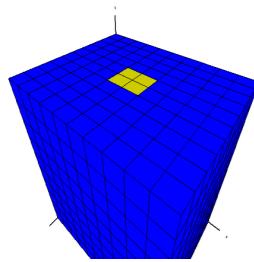


Figure 4.3: Cube in rest containing two materials.

The CAPA-3D software and Matlab are used to calculate the internal and external forces acting on the body during this experiment. The large differences in elasticity between the two materials yield a stiffness matrix with a very high condition number. In this experiment we used CG in combination with two different

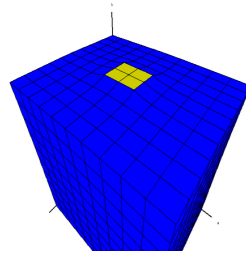


Figure 4.4: Deformed cube after compression test, containing two materials.

preconditioners - incomplete Cholesky and diagonal scaling - and deflation techniques. With Matlab we calculated the eigenvalues of the stiffnessmatrix and selected the eigenvectors corresponding to the two smallest eigenvalues as deflation vectors. Consider figure 4.5. The CG method with and without diagonal scaling is performing poorly and does not show any progress even after 200 iterations. When using the incomplete Cholesky we observe an increase in speed and the method converges in about 110 iterations. However, when deflation is being applied to CG with incomplete Cholesky we observe another increase in speed and the method converges in about 80 iterations. When using more eigenvectors as deflation vectors even better converge rates can be achieved but the deflation matrices become harder to compute.



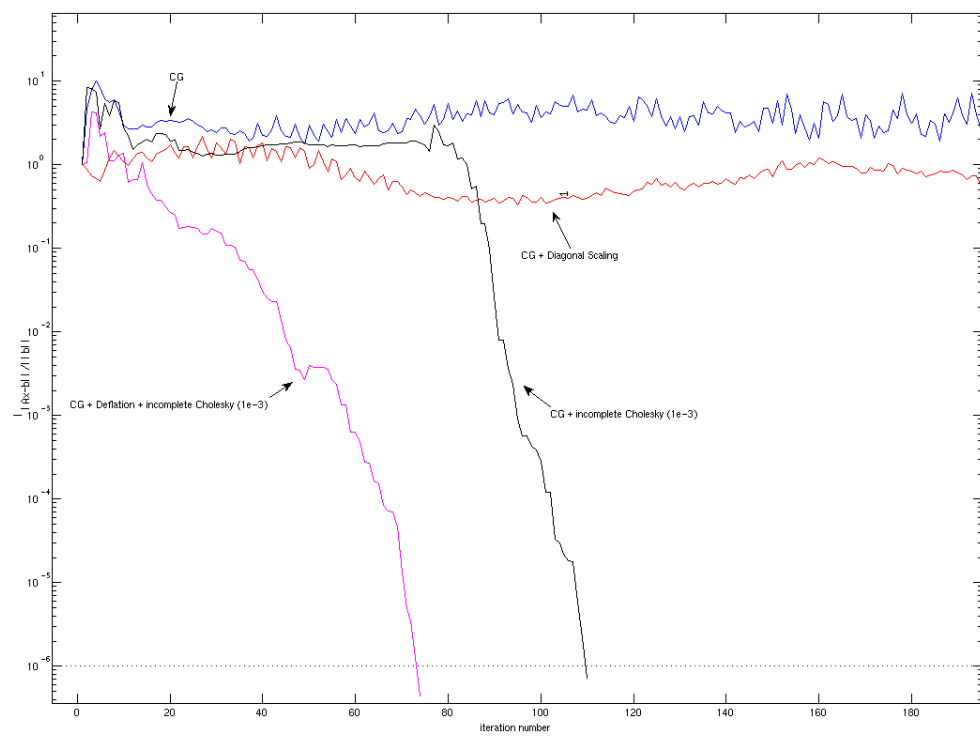


Figure 4.5: Compression test, two materials with deflated CG solver.



# Parallel direct solver

---

In preceding chapters it has become clear that most numerical methods will have to solve a (large) linear system somewhere in the algorithm. An obvious choice would be a parallel direct solver. Parallel direct solvers can handle large systems and divide the workload over a number of computing nodes. Moreover, adding more computing nodes will decrease computation times. Several open source implementations of parallel direct solvers are available, for example SuperLU [6] and MUMPS [4]. Both implementations are based on LU decomposition of the linear system. This research will focus on MUMPS. Next section will give a brief introduction on parallel computing,

## 5.1 Parallel computing

Parallel computing can be extremely difficult but also reasonably easy, it depends highly on the choices of the supporting packages of the parallel application. The open source community has developed many parallel software packages that can be invoked and glued together with a minimal knowledge of parallel programming. However, for hand tailored solutions of the parallelization of specific algorithms, some knowledge about parallel programming is mandatory. This section discusses the MPI language, which stands for Message Passage Interface and it is a protocol for communication between parallel threads. Also, an overview is given of several parallel open source packages and some tips and tricks on the architecture of a parallel cluster.

**MPI** Parallel programs can be observed as a collection of threads that run and finish independently but interact with each other during run time. In theory, when assigning each thread to one processing unit the parallel program should run faster than its serial counterpart. However, when threads are not independent and will have to wait for another thread to finish, parallel speed up will decrease. The Message Passing Interface is a platform independent protocol for communication between different threads of a parallel application. There are many MPI implementations but Open MPI [ ] and MPICH [ ] are used most.

Writing a parallel program with MPI is not complicated. Most MPI implementations are compatible with programming languages such as Fortran and the C-language and the compiler of choice can be used. A serial application can be parallelized by adding just a few extra commands. In the next example the Fortran programming language is used.

Consider the following simple 'Hello World' application. The output to the screen is the sentence 'Hello world in serial'.

```
program helloworld

write(*,*)'Hello world in serial'

end program
```

This program can be parallelized by adding the MPI commands. In the first line of the code the MPI header file, 'mpif.h' is included. In this file are references to MPI related functions are collected. To integers are added to store the rank number of the parallel thread and the error message. The first call is to 'mpi\_init'. It initializes the parallel program and it will create the number of threads indicated before the start of the program. The second call is to 'mpi\_comm\_rank', which is the communicator between the threads. The rank of the thread is stored in 'myrank'. The rank number is printed to the screen and the application is shutdown by killing all parallel threads with the last call 'mpi\_finalize'.

```
program helloparworld

include 'mpif.h'

integer myrank, ierr

call mpi_init(ierr)

call mpi_comm_rank(mpi_comm_world, myrank, ierr)

write(*,*)'Hello parallel world, from node ', myrank

call mpi_finalize(ierr)

end program
```

10

## 5.2 MUMPS

MUMPS is a public domain package and developed during the Esprit IV European project PARASOL (1996-1999) by CERFACS, ENSEEIHT-IRIT and RAL [4]. The MUMPS package computes a LU decomposition of any given matrix. An interface is available for *C* and *Fortran* software. The MUMPS software is parallel and should therefore be executed on parallel machines only. The MUMPS software offers an interface to *C* and *Fortran* software. Figure 6.3 displays a schematic overview of an external program invoking a MUMPS instance. There is no need for complicated MPI programming to embed MUMPS into existing software.

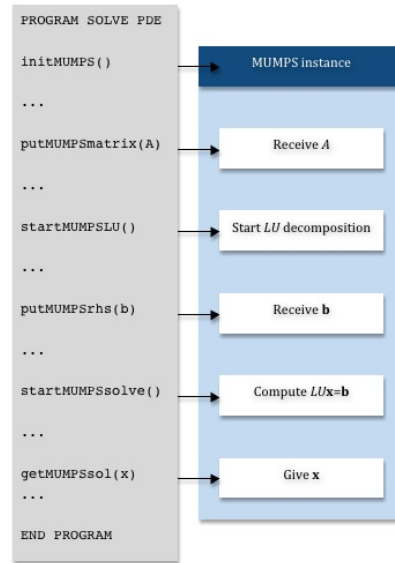


Figure 5.1: Invoking MUMPS from external program.

### 5.3 Test results

First test results show promising results for both the parallel direct solver and the preconditioned iterative solver. Two solvers have been implemented into the CAPA-3D software. The existing direct solver in algorithm 19 has been replaced by the parallel direct solver MUMPS [4] and Deflated preconditioned CG. The following experiments have been done on a cluster with four workstations carrying one Intel Xeon E5450 3.0 GHz Quad-core Duo and 16Gb of 800 MHz DDR2 memory each, yielding 32 processors and 64Gb of memory in total. The workstations were connected by an 1-Gigabyte ethernet network.

Figure 6.4 shows the test results for parallel LU decomposition of three different matrices. Five different configurations are considered with 2, 4, 8, 16 and 32 processors respectively. Each workstation only hosts 8 processors, hence the gray dotted vertical line indicates the transition to a cluster of workstations. Two and four workstations when requesting 16 processors and 32 processors respectively. Three matrices have been considered, each matrix with a different dimension and corresponding to a different material test. The bold lines represent the actual test results, the remaining black lines represent the theoretical linear speed up. An ideal parallel algorithm, with no communication overhead, should run twice as fast with twice as many processors involved. Apparently, this is not feasible when the communication between workstations, CPUs and memory is taken into account. The results are promising. For every test case reasonable speed up is observed. When the dimension of the matrices increases, MUMPS starts to perform better because initial communication overhead becomes negligible compared to the computation process. Only when more workstations become involved the speed up disappears because of network delays and latency. An infiniband network connection should

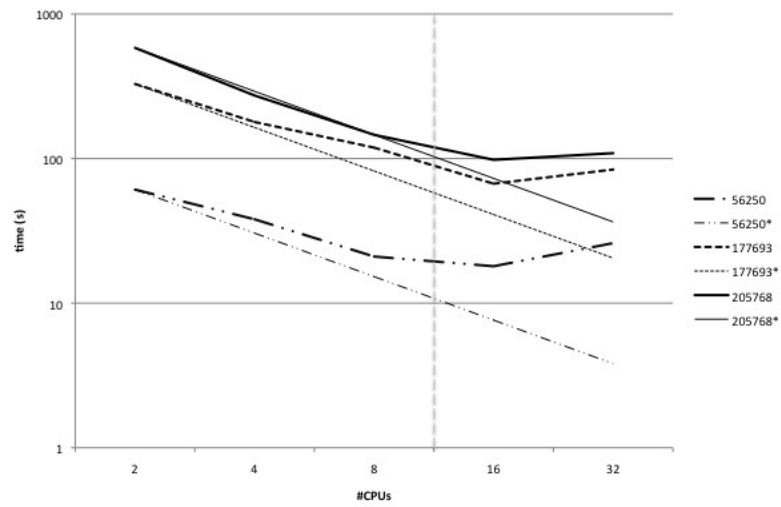


Figure 5.2: Parallel LU decomposition for different matrix dimensions.

overcome these problems.

# Summary

---

## Background structural mechanics

Within the field of structural mechanics, pavement engineering plays an important role in understanding and modeling the effects on heavy duty materials like asphalt and concrete when exposed to different kind of forces. Not only the appliance of force but also weather conditions and aging have to be taken into account. It is of crucial importance that the industry is able to predict how materials react to various circumstances under different time spans. Understanding these effects may result in more careful engineering of these materials which may replace the current trial and error design process. Moreover CAD methods will be more time and -cost efficient compared to laboratory tests.

The materials that are used in pavement engineering are one of the most difficult to model. Important material properties such as hyper-elasticity, viscosity and plasticity are non-linear phenomena and therefore hard to capture in equations. Decent algorithms are available but still under development. Especially the effects of water are not well understood and bring in an extra difficulties.

In real life applications, materials like asphalt and concrete are subjected to forces induced by humans, vehicles or planes that make use of the pavement. There are many cases that are interesting to investigate. Four phases of movement can be observed: acceleration, braking, constant speed and in rest. Each of these phases are important for explaining the deterioration of pavement. Obviously force has a direct relation with mass. Heavy vehicles like trucks or planes apply an enormous amount of pressure into the downward direction because of their sheer weight. Weight combined with speed can have massive impact on rubber like materials such as asphalt. Due to the effects of plasticity, roads nearby traffic lights are permanently deformed because of the braking effect and vehicles in rest. The same reason applies to vehicles in rest on parking lots. Permanent deformation of the pavement is observed also. These effects are magnified when trucks or planes are involved.

Common tests for examining the effect of pressure on pavement are compression and compaction tests. Columns of material are subjected to different loads and loading times. These experiments are vital for gathering data that can be used for benchmarking new material algorithms. The response of materials to distributed loads is captured in a stress-strain curve and is known as the constitutive relation. Different algorithms deliver different curves. A good fitting of the test data can justify the material response algorithms, Section1.4.

## Computational framework for pavement engineering

A framework for calculating material response, stresses and strains has been provided by dr. A. Scarpas *et al* [10],[8]. At the heart of this framework lies the virtual work equation that gives a relation between internal and external stresses,

$$\delta W(\mathbf{X}) = \delta W_{int}(\mathbf{X}) - \delta W_{ext}(\mathbf{X}) = 0 \quad (6.1)$$

where  $\delta W_{int}(\mathbf{X})$  and  $\delta W_{ext}(\mathbf{X})$  correspond to the energy induced by the internal and external forces respectively, Section 1.3.

The non-linear material response is an internal force and therefore equation 6.1 has to be linearized in order to solve it. Hence, a first order taylor expansion around configuration  $\mathbf{X}_0$  yields,

$$\delta W(\mathbf{X}_0) + D_{\Delta u}[\delta W(\mathbf{X}_0)] = 0 \quad (6.2)$$

After substitution of expressions for the internal and external energies and rewriting the following expression for the virtual work equation is obtained,

$$\begin{aligned} \int_V (\nabla_0 \Delta \mathbf{u} \cdot \mathbf{S}) : \nabla_0 \delta \mathbf{v} dV + \int_V (\nabla_0 \Delta \mathbf{u} : \mathbf{F} \cdot \mathbb{C} \cdot \mathbf{F}^T) : \nabla_0 \delta \mathbf{v} dV = \\ \delta \mathbf{v} \cdot \mathbf{f}_{ext} - \int_V \mathbf{P} : \nabla_0 \delta \mathbf{v} dV \end{aligned} \quad (6.3)$$

where  $\mathbf{F}$  is the deformation matrix, i.e. the deformation of the material compared to two subsequent configurations,  $\mathbf{P}$ ,  $\mathbf{S}$  and  $\mathbb{C}$  are functions of  $\mathbf{F}$  and  $\Delta \mathbf{u}$  represents the displacements.

## Discretization

The linearized virtual work equation is discretized by the finite element method, Section 2.1. A mesh is created to represent the body of the material, the mesh elements are either tetrahedrals or cubes. The resulting grids will be unstructured in most real life applications. Sophisticated mesh generators like Cubit [3], TetGen [5] or Amira [7] produce meshes that are congruent with the structure of the material. For example, a material like asphalt contains three basic ingredients, stones, bitumen and air. To obtain realistic simulations of the material each ingredient should be represented as accurate as possible. The mesh should follow the borders between the ingredients. In figure 6.1 a sample of asphalt has a 2D grid overlay. The stones (gray) and the bitumen (black) are identified easily. The red, triangular grids cells correspond to the stones and the blue grid cells correspond to the bitumen. Finer grids deliver more accurate and realistic simulations.

The finite element discretization uses second order shape functions, hence the unknowns lie on the corners and in the centre of the vertices of the elements. The linearized virtual work equation 6.3 has to be solved for the unknown displacement



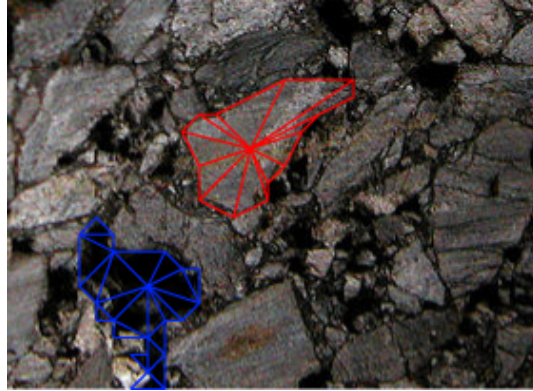


Figure 6.1: Asphaltic core with 2D grid overlay.

field  $\Delta \mathbf{u}$ . The displacement field has three dimensions, therefore the unknowns on the grid nodes have three components for the  $x$ ,  $y$  and  $z$  direction respectively.

The discretization of the linearized virtual work equation results into a short hand notation of the static system that has to be solved,

$$K \Delta \mathbf{u} = \Delta \mathbf{f} \quad (6.4)$$

where  $K$  is the stiffness matrix,  $\mathbf{u}$  the unknown incremental displacement field and  $\Delta \mathbf{f}$  the difference between the internal and external stresses.

Most simulations use either a prescribed displacement field for some elements of the mesh or separate load steps. The algorithm 19 to solve equation 6.4 is invariant under these two situations.

---

**Algorithm 19** Balancing of forces

---

```

for  $t = 0 \dots t_{\text{end}}$  do
  Compute external load  $\mathbf{f}_{\text{ext}}^t$ 
  for  $i = 0$  until convergence do
    Assemble stiffnessmatrix  $K^{t,i}$ 
    if  $i = 0$  then
       $\mathbf{f}_{\text{int}}^0 = 0$  and  $\Delta \mathbf{f}^0 = \mathbf{f}_{\text{ext}}^0 - \mathbf{f}_{\text{int}}^0$ 
    Solve system  $K^{t,i} \Delta \mathbf{u}^i = \Delta \mathbf{f}^i$ 
    Update displacements,  $\mathbf{u}^{i+1} = \mathbf{u}^i + \Delta \mathbf{u}^i$ 
    Compute internal force,  $\mathbf{f}_{\text{int}}^{i+1}$  and  $\Delta \mathbf{f}^{i+1} = \mathbf{f}_{\text{ext}}^{i+1} - \mathbf{f}_{\text{int}}^{i+1}$ 
    Test for convergence,  $\frac{\Delta \mathbf{f}^{i+1}}{\Delta \mathbf{f}^0} < \varepsilon$ 

```

---

Algorithm 19 has two important computation steps.

First step is the computation of the increment of the displacement field. This is equal to solving the linear system of equation 6.4. The stiffness matrix  $K$  is symmetric positive definite for all simulations, Section (). This is an important property as it is crucial for obtaining good convergence rates for many numerical solution methods. For small meshes the dimension of  $K$  allows for direct solution

methods if the matrix is non-singular. However, with the refinement of the meshes the dimension and complexity of the linear system increases significantly and other numerical solution methods need to be found.

Second step is the internal stress update. The non-linear equations of the material response algorithms are solved with the Newton-Raphson method and can be evaluated for each element separately, Section 2.2. With the refinement of the meshes and increasing complexity of the material response algorithms, a more efficient internal stress update needs to be devised.

Increasing the dimension of  $K$  induces great difficulties for direct solvers because computer memory and CPU power are limited. The condition number is defined as the quotient between the largest and smallest eigenvalues of a matrix. Large condition numbers yield ill-conditioned systems which are therefore difficult to solve. Not only grid refinement but also the non-linear material properties affect the solvability of system 6.4. When plasticity and viscosity builds up or hyper-elasticity applies, stiffness of the materials changes and stiffness matrix  $K$  will have to be reassembled. Large differences in stiffness between material ingredients, e.g. stone and bitumen, will result in large condition numbers and thus in slow converging solvers.

## Numerical methods for structural mechanics

The research will focus on the development of a fast, robust and scalable solver that can deal with a variety of constitutive material models (elasticity, plasticity, viscosity and cracking) and mesh sizes. The solver should be parallel to ensure the possibility of upscaling of the experiments, i.e. meshes, as single core computers lack memory and CPU power. Moreover, parallelization of the solver will be extended to parallelization of the internal stress updates as these will affect the performance of algorithm 19 significantly.

Finding a suitable numerical method for solving an arbitrary linear system  $A\mathbf{x} = \mathbf{b}$  can be difficult if the linear system is ill-conditioned. Neither direct solution methods nor iterative solvers will perform, if a solution can be computed at all. Preconditioning of the system can reduce the condition number of  $A$  and hence improve convergence rates significantly. Multiply matrix  $A$  with the inverse of a matrix  $M$  to obtain the better conditioned system,

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \quad (6.5)$$

Matrix  $M$  is the left preconditioner and is preferably an approximation of matrix  $A$ , Section 4.3. However, mostly matrix  $M$  is not known explicitly but the operation  $M^{-1}$  is replaced by solving  $v$  from system  $M\mathbf{v} = \mathbf{w}$  where  $M$  can be any linear solver.

The choice of numerical solvers can be overwhelming and there is still no standard recipe for solving an arbitrary set of equations. Many well known engineering problems like fluid dynamics, the Helmholtz and the Maxwell equations have their own, tailored solution methods. Multigrid, conjugate gradient (CG), deflation and

domain decomposition dominate the spectrum of modern solvers. It is common practice to combine these methods to take full advantage of their single properties. Figure 6.2 gives a schematic overview of combinations of different numerical methods for solving system 6.4.

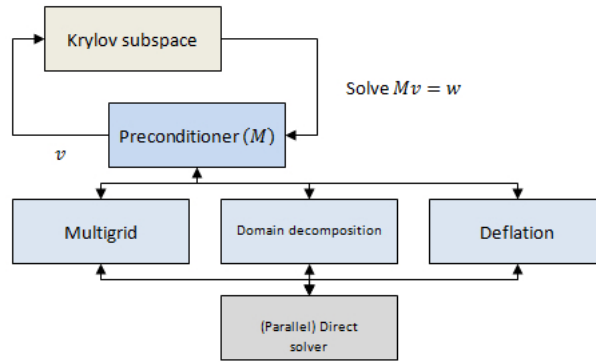


Figure 6.2: Schematic overview of the combination of different linear solvers into one numerical method.

The CG, Krylov subspace method is in the upper iteration loop and has its preconditioner in the lower iteration loop. Possible choices of preconditioners are either multigrid, domain decomposition or deflation. Because many preconditioners still involve direct solution of linear systems the parallel direct solver is added to the figure. The choice of the CG method is straightforward when the properties of the stiffness matrix of equation 6.4 are taken into account. The CG method is well known for its excellent performance on semi definite symmetric systems, Section 4.4. Moreover, preconditioned CG method only yields one extra computation step where the system  $M\mathbf{v} = \mathbf{w}$  needs to be solved. A sensible choice of the preconditioner will result in a stable numerical method.

Multigrid, domain decomposition and deflation can be preconditioners for CG. All three methods share the same principle: projection of the linear system onto smaller subspace where the reduced system will be solved and back propagation of the reduced solution to obtain a solution for the original system. Two requirements are satisfied, robustness and the ability of handling very large systems of equations, at the cost of speed per iteration. Obviously, bringing in extra linear solvers will increase the complexity of the solver and the computation times. However, when tailored in the right way, the reduction of the number of iterations due to stability will increase the overall speed of the solver.

**Multigrid** Multigrid is known for its performance with respect to solving elliptic differential equations on structured, equidistant grids, Section 4.5. The amount of work is of  $\mathcal{O}(N)$  with  $N$  the number of unknowns, linear convergence is observed and grid operators are easily defined. The main advantage of multigrid is the reduction of the number of unknowns that has to be solved. The linear system is solved on a much

coarser grid than the original domain. Grid operators interpolate the solution back to the finer grid where solution errors are smoothed out. Unfortunately, convergence rates similar to solving elliptic differential equations are not observed when solving other linear systems. Moreover, defining grid operators on unstructured grids is a complicated process. Adapted multigrid methods like algebraic multigrid (AMG) are designed for unstructured grids but more suited numerical methods are available.

**Domain decomposition** Domain decomposition methods are designed for numerical problems where subdomains can easily be identified, Section 4.6. The original linear system  $A\mathbf{x} = \mathbf{b}$  is split into  $s$  problems where  $s$  denotes the number of subdomains. The resulting system can be written as,

$$A \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} \text{ with } A = \begin{pmatrix} B & E \\ F & C \end{pmatrix}. \quad (6.6)$$

where the unknowns in the interior of the domains are collected in vector  $\mathbf{x}$  and the unknowns on the interfaces are collected in vector  $\mathbf{y}$ . This system can be solved for the interface elements  $\mathbf{y}$  solely from which  $\mathbf{x}$  can be obtained. Several algorithms, including preconditioned CG, are available to solve the linear system after domain decomposition. Choosing the domains however is complicated and not straightforward. When the subdomains are chosen optimally the slow converging components of the solution should be filtered out. These components often coincide with physical effects like large differences in stiffness between two neighboring materials. Much research is needed to find an algorithm for defining the subdomains without introducing numerical artifacts.

**Deflation** The deflation method has a strong resemblance with both multigrid and domain decomposition, Section 4.7. The philosophy behind deflation is filtering out the slow converging components of the solution. It has been proved in [11] that for CG these components correspond to the smallest eigenvalues in the spectrum of the linear system. If the original system has dimension  $n$  a smaller system is introduced with dimension  $n - k$  where  $k$  corresponds to the number of eigenvalues that have been filtered out. The reduced system is better conditioned and hence convergence rates improve. Both the stability and reduction requirements are satisfied. Unfortunately, calculation of the eigenvalues of large linear systems is both time and resource consuming. Approximations of the eigenvectors are used instead. The eigenvectors correspond to physical aspects of the problem. For example the density of water and air in bubbly flow, Section (). On the interface between water and air bubbles the difference between densities is big. These interfaces correspond to the slow converging components of the solution, i.e. to the smallest eigenvalues. Filtering out those interfaces better convergence rates for CG are observed [12]. The eigenvector approximations corresponding to the interfaces are easy to construct.

**Parallel direct solver** On the bottom of figure 6.2 a direct solver is displayed. All three preceding numerical methods will have to solve a (large) linear system. An

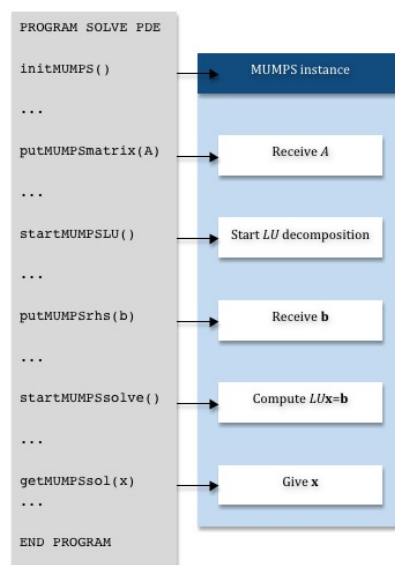


Figure 6.3: Invoking MUMPS from external program.

obvious choice would be a parallel direct solver. Parallel direct solvers can handle large systems and divide the workload over a number of computing nodes. The requirements of speed and scale are easily satisfied. Moreover, adding more computing nodes will decrease computation times. Several open source implementations of parallel direct solvers are available, for example SuperLU [6] and MUMPS [4]. Both implementations are based on LU decomposition of the linear system. This research will focus on MUMPS. The MUMPS software offers an interface to *C* and *Fortran* software. Figure 6.3 displays a schematic overview of an external program invoking a MUMPS instance. There is no need for complicated MPI programming to embed MUMPS into existing software.

## Test results

First test results show promising results for both the parallel direct solver and the preconditioned iterative solver. Two solvers have been implemented into the CAPA-3D software. The existing direct solver in algorithm 19 has been replaced by the parallel direct solver MUMPS [4] and Deflated preconditioned CG.

**MUMPS** MUMPS is a public domain package and developed during the Esprit IV European project PARASOL (1996-1999) by CERFACS, ENSEEIHT-IRIT and RAL [4]. The MUMPS package computes a LU decomposition of any given matrix. An interface is available for *C* and *Fortran* software. The MUMPS software is parallel and should therefore be executed on parallel machines only. The following experiments have been done on a cluster with four workstations carrying one Intel Xeon E5450 3.0 GHz Quad-core Duo and 16Gb of 800 MHz DDR2 memory

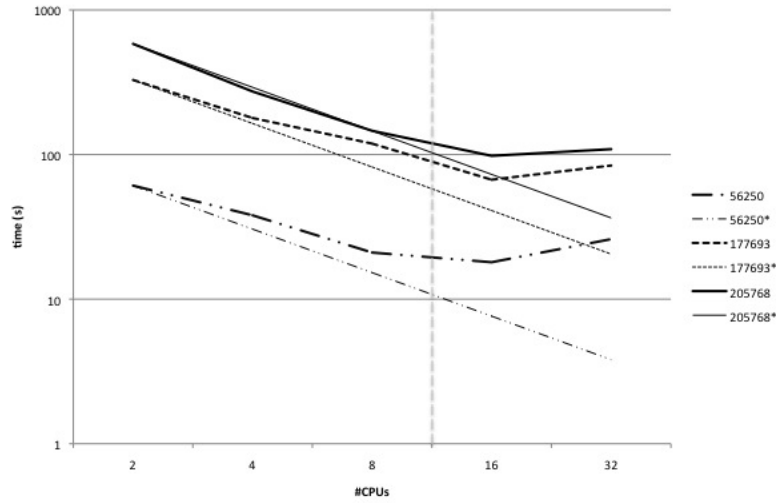


Figure 6.4: Parallel LU decomposition for different matrix dimensions.

each, yielding 32 processors and 64Gb of memory in total. The workstations were connected by an 1-Gigabyte ethernet network.

Figure 6.4 shows the test results for parallel LU decomposition of three different matrices. Five different configurations are considered with 2, 4, 8, 16 and 32 processors respectively. Each workstation only hosts 8 processors, hence the gray dotted vertical line indicates the transition to a cluster of workstations. Two and four workstations when requesting 16 processors and 32 processors respectively. Three matrices have been considered, each matrix with a different dimension and corresponding to a different material test. The bold lines represent the actual test results, the remaining black lines represent the theoretical linear speed up. An ideal parallel algorithm, with no communication overhead, should run twice as fast with twice as many processors involved. Apparently, this is not feasible when the communication between workstations, CPUs and memory is taken into account. The results are promising. For every test case reasonable speed up is observed. When the dimension of the matrices increases, MUMPS starts to perform better because initial communication overhead becomes negligible compared to the computation process. Only when more workstations become involved the speed up disappears because of network delays and latency. An infiniband network connection should overcome these problems.

**Deflated preconditioned CG** Deflation methods DCG and DPCG are applied to a very simple strain-stress test. A cube containing two materials is subjected to a load. The outside is rubber like and has only elastic material properties. The material within the centre and core of the cube is incompressible, one could think of steel or stone. A distributed load is being applied at the top of the cube and is forcing it downwards within several load steps (static mechanics). The CAPA-3D software and Matlab are used to calculate the internal and external forces acting

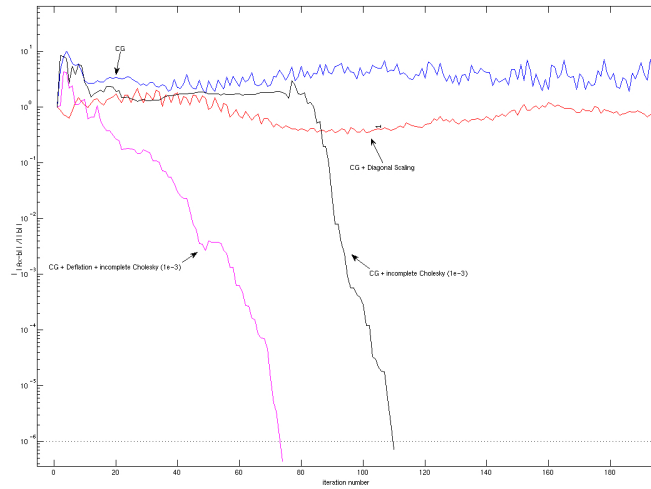


Figure 6.5: Compression test, two materials with deflated CG solver.

on the body during this experiment. The large differences in elasticity between the two materials yield a stiffness matrix with a very high condition number, hence bad convergence rates are expected. The experiment uses a combination of CG, two different preconditioners - incomplete Cholesky and diagonal scaling - and deflation. The eigenvalues and eigenvectors corresponding to the two smallest eigenvalues are selected as deflation vectors and have been computed with Matlab. Consider figure 6.5. The CG method with and without diagonal scaling is performing poorly and does not show any progress even after 200 iterations. Preconditioning with incomplete Cholesky yields an increase in speed and the method converges in about 110 iterations. However, when deflation is being applied to CG with incomplete Cholesky preconditioning, another increase in speed is observed and the method converges in about 80 iterations. It is to be expected that increasing the number of deflation vectors would yield even better converge rates but the application of the deflation operator becomes more expensive.





# Bibliography

- [1] K. J. Bathe. *Finite Element Procedures*. Prentice Hall, 1996. 21
- [2] Douglas J. Faires and Richard L. Burden. *Numerical Methods*. PWS-Kent Pub.Co., Boston, 1993. 17
- [3] Sandia Labs <http://cubit.sandia.gov/>. Cubit, geometry and mesh generation toolkit. 72
- [4] CERFACS <http://graal.ens-lyon.fr/MUMPS/>. Superlu sparse gaussian elimination on high performance computers. 67, 68, 69, 77
- [5] Hang <http://tetgen.berlios.de/>, Si. Tetgen, a quality tetrahedral mesh generator and three-dimensional delaunay triangulator. 72
- [6] Xiaoye Li <http://www.cs.berkeley.edu/~demmel/SuperLU.html>, Jim Demmel. Superlu sparse gaussian elimination on high performance computers. 67, 77
- [7] Mercury Computer Systems <http://www.tgs.com/>. Amira visualization software. 72
- [8] N. Kringos. *Modeling of combined physical-mechanical moisture induced damage in asphaltic mixes*. TU Delft, 2007. 15, 72
- [9] Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Mathematics, April 2003. 37, 45, 46, 47, 54, 56
- [10] A. Scarpas. *a Mechanics based computational platform for pavement engineering*. TU Delft, 2004. iii, 7, 14, 72
- [11] L.J. Sluys and R. de Borst. *Computational methods in non-linear solid mechanics*. TU Delft, 2001. 21, 76
- [12] Y.M. Tang. *Two-Level Preconditioned Conjugate Gradient Methods with Applications to Bubbly Flow Problems*. TU Delft, 2008. 59, 60, 62, 76
- [13] Ulrich Trottenberg, Cornelius W. Oosterlee, and Anton Schuller. *Multigrid*. Academic Press, December 2000. 37, 40, 49
- [14] C. Vuik. *Numerical methods for large algebraic systems*. TU Delft, 2004. 37, 38, 41, 45

