Universitetet i Oslo
Institutt for informatikk

Mixed Finite Element Method for Elasticity with Strongly Imposed Symmetry

Vera Louise Hauge

25th July 2005

Preface

This thesis is submitted for the degree of candidata scientarium at the Department of Informatics, University of Oslo.

I would like to thank my supervisor professor Ragnar Winther, for the idea of this thesis, for the time taken to answer all my questions, and for always giving patient and explanatorious answers.

Next, I owe Kent-André Mardal my greatest thanks and gratitude; For all his time spent on helping me, for answering all kinds of question and helping with all kinds of obstacles, for providing Diffpack version 4.1 and all necessary source code and none the least for his encouragement. Without his extensive help the implementation process had been far less complete and the work would have to be submitted unfinished.

I would also like to thank Yanqiu Wang, Texas A&M University, for sharing her knowledge of her implementation of the Arnold-Winther element with me.

Lastly, thanks to Per-Idar Evensen for proofreading the thesis.

25th July, 2005

Vera Louise Hauge

Contents

Chapter 1

Introduction

1.1 The purpose of the thesis

The initial purpose of this Cand.scient. thesis is to implement a new element for the mixed finite element method for the elasticity equation. The new element has strongly imposed symmetry in the stress and is presented by Douglas N. Arnold and Ragnar Winther in *Mixed finite elements for elasticity* [1]. We will refer to the new element as the Arnold-Winther element.

Before any implementation was possible, an understanding of the finite element methods and how elements are defined had to be obtained. Thus in the process of implementing the simulator and the elements, a great effort has also been put into acquiring an understanding and overview of the theory behind.

Implementation of a simulator using the mixed finite element method for the elasticity problem enables numerical simulations and experiments. In this thesis the numerical simulations will be concentrated around possible effects when the elasticity or Lamé constants are discontinuous. Thus we have the following research question and answering this question will be the second purpose of this thesis.

1.1.1 Research question

Will mixed finite element method reveal more details in the simulation of elasticity models when discontinuous Lamé constants are involved, than what the classical Galerkin finite element method does?

1.1.2 Short background to the research question

As will be described in the introductory example, mixed finite element method for the Poisson equation yields a better solution than standard finite element method when the problem has a discontinuous coefficient. Due to resemblance in the mathematical formulation of the Poisson equation and the elasticity equation, it is therefore interesting to see if the same type of differences in solutions of finite element method and mixed finite element method occur when we have discontinuous coefficients in the elasticity equation.

In addition, some of the background for using mixed finite element method for elasticity, thus the necessity for constructing a mixed element, might be found in physical application of elasticity: The mathematical models might be complex such that mixed method arises naturally when applying the Galerkin element method. Moreover, the locking phenomenon which might occur in Galerkin element method is avoided using mixed finite element method.

1.2 Outline of the thesis

The next chapter, Chapter 2, presents an introductory example, serving as a motivation for the research question of this thesis. Here both Galerkin finite element method and mixed finite element method are used without any further explanations. The purpose of this chapter is to motivate the implementation of the latter method, without going any deeper into the theory yet.

The following chapters then introduce the theory and background necessary for the implementation. Chapter 3 presents the theory of finite element methods and mixed finite element method. Chapter 4 introduces the new element which is to be implemented with details of the calculation, and finally the theory of elasticity is found in Chapter 5. The implemention of the new element is documented in Chapter 6, before numerical simulations and the subsequent results are presented in Chapter 7. Towards the end, in Chapter 8, further investigations and extensions are suggested.

The implementation is done using the software package Diffpack, version 4.1. The description of the implementation is based on already existing programs and existing classes. Thus the description will require some knowlegde of Diffpack.

None of the details of the implementation nor the actual numerical results will appear before respectively Chapter 6 and Chapter 7. The reader, familiar to finite element method and elasticity and interested in the implementation or the results of the simulations, is recommended to fast forward to these chapters.

These two chapters in additions to the calculation of the basis functions of the Arnold-Winther elements (Chapter 4) constitute the main parts of the work done and results in this thesis. The rest is meant to serve as a thorough background. Understanding this background has been a essential part in completing this thesis.

1.3 Abbreviations and notation

In the text the two element methods will often be abbreviated as FEM for finite element method and MxFEM or mixed FEM for mixed finite element method.

In the elasticity equation vectors and vector operators are denoted with a bar above the symbol, for instance $\bar{v}$. Tensors are denoted with a double bar above the symbol, $\bar{\bar{\tau}}$.

Chapter 2

An Introductory Example

2.1 A source-sink model

A simple model for fluid flow in porous media is given by the elliptic pressure equation

$$-\text{div } (\bar{\bar{A}}(\bar{x})\text{grad } p) = f \quad \text{in } \Omega \tag{2.1}$$

$$(\bar{\bar{A}}(\bar{x})\text{grad } p) \cdot \bar{n} = 0 \quad \text{on } \partial\Omega \tag{2.2}$$

where p is the pressure of the fluid in the domain Ω, $\bar{n}$ is the normal vector and $\bar{\bar{A}}(\bar{x})$ is the permeability tensor. The Neumann boundary condition (2.2) prescribes no flux of the fluid through the boundary. Due to this type of boundary conditions, the equation does not generally have a unique solution. The Neumann boundary condition makes the solution unique only up to an additive constant. This is described in the last part of Section 2.1.3.

The function f is specified to be equal to 0 all over the area, except in two corners where we have respectively a source and a sink, with f-values equal to respectively 100 and -100:

source	$f = 100$
sink	$f = -100$
otherwise	$f = 0$

Figure 2.1 illustrates the system. With the above function on the right hand side of equation (2.1), the boundary value problem might be considered as a source-sink model.

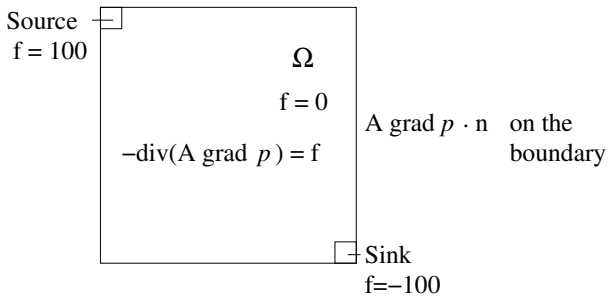


Figure 2.1: The boundary value problem for fluid flow.

2.1.1 Permeability

The media in which the fluid flows has a permeability; That is a property which describes the capability of a fluid to flow through the media. This permeability is given in the symmetric, positive definite tensor $\bar{\bar{A}}$ in our equation (2.1). The permeability might vary not only from one media to another, but also within one media. Such spatial variations are often discontinuous. Thus to reflect such spatial discontinuity, the tensor might be discontinuous over the area.

In our case a discontinuous tensor means that we have one tensor in one area and another one in the rest of the domain. Figure 2.2 shows a domain with two different permeability regions: The white area has one permeability tensor $\bar{\bar{A}}_1$ and the gray area has a different one, $\bar{\bar{A}}_2$.

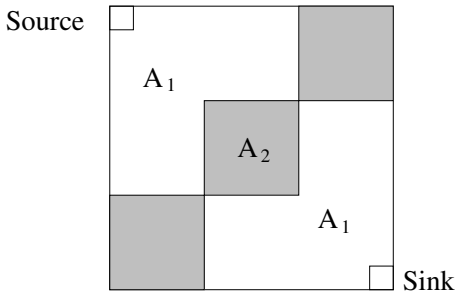


Figure 2.2: The discontinuous permeability field for our source-sink model.

Relating the permeability concept to a porous media, for instance sand, means that a high permeability (easy flow of fluid) occurs when the pore space available for flow is larger than in a low permeability region. For instance one can imagine that in a region with high permeability, the sand is loosely packed, while in a low permeability region the sand is compactly packed. In addition the structure of the order of sand particles might give an easier flow in one space direction than in the other.

More about the physical interpretation of both the source-sink system and permeability in porous media might be found in R. A. Klausen's *Introduction to the thesis "On Locally Conservative Numerical Methods for Elliptic Problems; Application to Reservoir Simulation"* in [9].

2.1.2 Expected behaviour of the fluid

The behaviour we can expect from the fluid in our setting, is a flow into the domain from the source and toward the sink, where it finally flows out of the domain. The Neumann boundary condition, making the area isolated, causes no fluid to flow in and out across the boundary.

When the permeability is homogeneous, logical reasoning tells us that the fluid will flow toward the sink and the pressure p will decrease smoothly from the source to the sink.

Adding the discontinuous permeability as sketched in Figure 2.2, and assigning the white region a lower permeability than the gray area, makes us expect the fluid to flow slower in the white low permeability region, than in the gray area.

Having such a field as in Figure 2.2, means in addition that there are sharp corners where the permeability changes. We can thus expect the fluid to flow slowly the whole way in the low permeability region, into the corners, and then abruptly change when it reaches the high permeability region. In a simulation we would thus expect to recognize the shape of the boundary between the different permeability fields.

2.1.3 Simulations of fluid flow using different numerical methods

Numerical simulations of the pressure of the fluid flow were done using two different numerical methods; Galerkin finite element method (FEM) and mixed finite element method (MxFEM). The technical description of how the element methods are used in this section requires some knowledge. Chapter 3 will explain these methods.

Comparison of the two methods is done only qualitatively, that is by comparing plots and observing visual differences in these plots. This is because the visual differences illustrate the necessary point.

The source-sink model used, is the one given in the first section. The elements used in FEM are standard linear elements. In MxFEM linear elements are used for the pressure, while Mini elements for the velocity. The Mini element is described in Kent-André Mardal's paper *Mixed Finite Elements* in [12]. Similar simulations of fluid flow are usually done with mixed finite elements such as Raviart-Thomas. However, the above mentioned paper shows that also the Mini element can be used for mixed models.

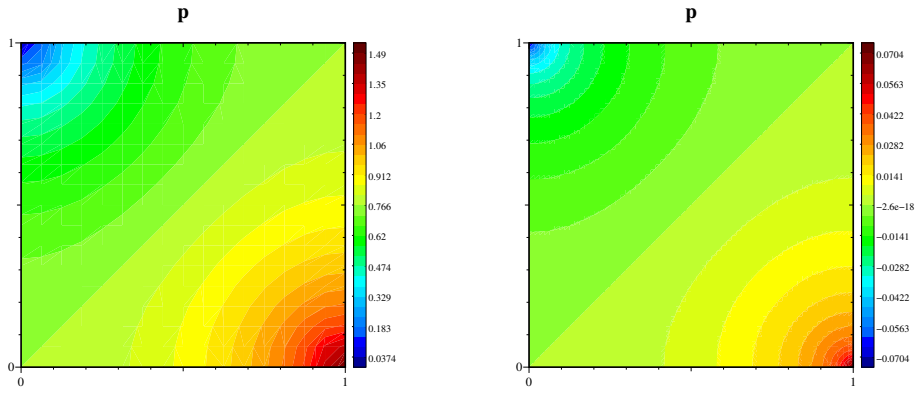
Firstly, simulations using a homogeneous permeability field are done. In this case the permeability tensor is defined to be

$$\bar{\bar{A}}_1 = \bar{\bar{A}}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Secondly, simulations with the discontinuous permeability field from Figure 2.2 are done, and the discontinuous permeability tensor $\bar{\bar{A}}$ is defined to be:

$$\bar{\bar{A}}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \bar{\bar{A}}_2 = \begin{bmatrix} 55 & 0 \\ 0 & 55 \end{bmatrix}$$

Homogeneous permeability field



(a) FEM (b) MxFEM

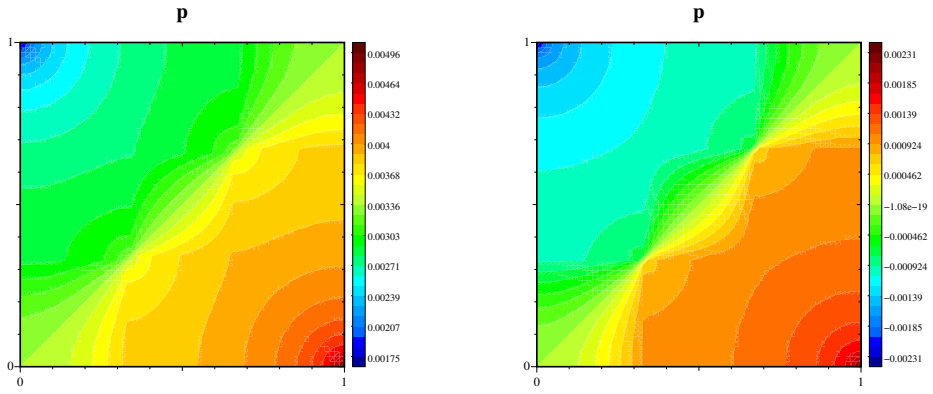
Figure 2.3: Fluid flow in homogeneous permeability field.

Figure 2.3 shows the pressure of the fluid when the permeability field is homogeneous. We see that in both plots the pressure is decreasing uniformly and there is no qualitative difference.

The only difference to point out is around the source and sink; Here the MxFEM figure seems to have a smaller area of high pressure around the source. However, this difference does not imply any difference in the behaviour of the fluid.

Thus in the case of homogeneous permeability, FEM and MxFEM yields similar results, with no decisive qualitative differences.

Discontinuous permeability field



(a) FEM (b) MxFEM

Figure 2.4: 2D plots of fluid flow in discontinuous permeability field.

Figure 2.4 and Figure 2.5 show the numerical results when using discontinuous permeability field. We see that the MxFEM plot shows the boundary between the low and high permeability region better, that is we can more easily recognize the underlying field. While in the FEM plot it seems to be more smeared out.

This smearing effect is precisely one special feature of FEM. R. A. Klausen points out this smearing effect in her thesis [9]. Particularly in the case of discontinuity, this smearing effect of FEM appears. If the point is to avoid this effect, the MxFEM could thus be a better choice.

Note on disagreeing scales

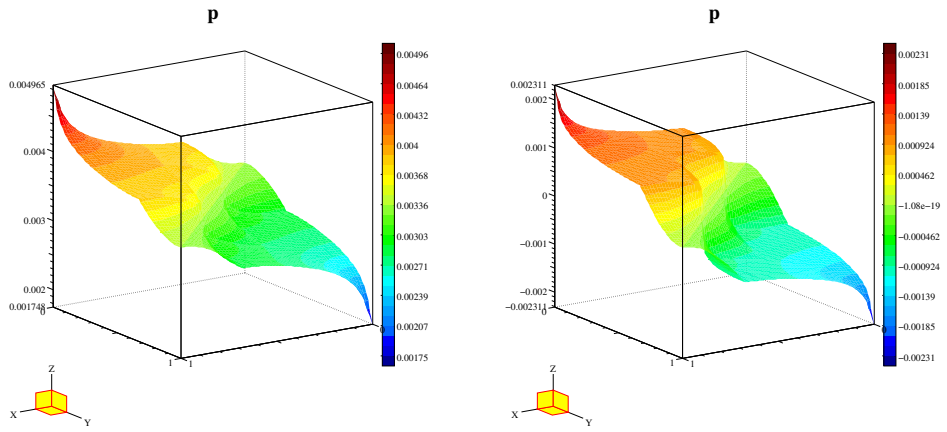
The observant reader will notice that the plots we compare have disagreeing scales. The reason for this is that solutions of equations with pure Neumann boundary conditions are unique only up to an additive constant. This means that if p is a solution, then also $q = p + c$ where $c \in \mathbb{R}$ is a constant, is a solution.

Our system is such a pure Neumann boundary condition problem. Solved by different numerical methods thus might result in differently scaled solutions.

Formulating the system such that it has a unique solution, could be achieved by adding a boundary condition to the system. Such a boundary condition might be one point along the boundary is set equal to a constant. For instance we can add the boundary condition

$$p(x, y) = c \quad (x, y) \in \partial\Omega.$$

However, as this source-sink example is only included to serve as an illustratory example motivating the research question, and since no other comparisons other than qualitative



(a) FEM (b) MxFEM

Figure 2.5: 3D plots of fluid flow in discontinuous permeability field.

are undertaken, the two applications computing the two solutions were not modified to yield solutions in precisely the same intervals.

2.2 Motivation for the research question

This source-sink example with discontinuous permeability serves as a motivation for using MxFEM for elliptic problems, in this case particularly the elasticity problem.

Even though FEM does apply to this source-sink example, some physical conditions make the MxFEM more favourable. We see from our numerical simulations in this chapter, that in the case of discontinuity the MxFEM simulates the details around the discontinuity better. This is a generally accepted fact.

In fluid flow we have discontinuous permeability field, while in elasticity we might have discontinuous material constants.

Secondly, MxFEM allows better accuracy for velocity in fluid flow and stress in elasticity, since numerical derivation of these quantities is avoided. In addition, approximation estimates will be in $H(\mathrm{div})$ in MxFEM, which is a natural choice.

Lastly, there are extensions to both mathematical models which cannot be reduced to *one* elliptic equation. For instance one variant of fluid flow models is the Navier-Stokes equations. In elasticity, we have for instance the system of visco-elasticity. These system cannot be reduced to one equation such that FEM can be applied. In these cases MxFEM is necessary if the idea is to solve the equations by some finite element method.

Thus both the case of discontinuity and extended systems, makes it interesting to apply MxFEM to elasticity.

Analogy to elasticity

The only analogy between the elliptic pressure model as described in this chapter and the elasticity equation which will be described later, is that both models are described

by a *elliptic* mathematical model. The mathematical formulation for both is

$$-\text{div } (A \text{ grad } u) = f$$

with some boundary conditions. However, for elasticity the unknown u is a vector, while for the pressure equation u is a scalar.

The analogy is not stronger than this mathematical resemblance. Physical interpretations of the models differ significantly.

With the numerical results for the pressure equation as we saw in this chapter, we have seen some results for one elliptic equation. It is therefore interesting to see how the results turn out for other elliptic equations as the elasticity equation.

Restatement of research question

The observations of better results with MxFEM in the discontinuous permeability field for the pressure equation, together with the analogy to elasticity, lead us to our research question for this thesis: Will mixed finite element method reveal more details in the simulation of elastic models when discontinuous coefficients are involved, than Galerkin finite element method?

Chapter 3

Finite Element Method

Finite element method (FEM) is a numerical method for solving partial differential equations. The method has a sound mathematical foundation in functional analysis. The following sections introduce the method both through a direct approach, developing the numerical method, and from a mathematical approach, where we also seek to establish existence and uniqueness of solutions.

These two approaches for describing the same method are used because they serve different purposes: The direct method provides an easier way of understanding *what* FEM does and how the numerical method is formulated, ready for implementation, while the mathematical approach helps understanding *why* FEM works. In addition the latter approach presents the notational conventions and terminology used in FEM.

The direct approach is to a large degree based on chapter 2 in *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*, [11]. The second approach is based on *The Mathematical Theory of Finite Element Method*, [3], and *Finite elements: Theory, fast solvers, and applications in solid mechanics*, [2], with some support from *The Finite Element Method* vol. 1 *The Basis*, [15].

As a later chapter will be concerned with the implementation of mixed finite element method for elasticity, some main principles of implementing both FEM and MxFEM will also be introduced in this chapter.

3.1 The direct approach to FEM

The main idea of the finite element method as a numerical method, is to approximate the unknown function in a partial differential equation (PDE) with a sum of prescribed functions, often referred to as basis functions. Let the PDE be

$$Lu = f \quad \text{in } \Omega. \tag{3.1}$$

The boundary conditions will be incorporated later in Section 3.1.2. L is a differential operator. An approximation $\hat{u}$ of the solution u is then

$$u \approx \hat{u} = \sum_{i=1}^{n} u_i N_i \tag{3.2}$$

where N_i are the basis functions and n is the number of such basis functions.

The PDE being solved, is then multiplied on both sides with a set of prescribed test functions W_j and then integrated over the prescribed domain Ω:

$$\int_\Omega L\hat{u} W_j \mathrm{d}\Omega = \int_\Omega f W_j \mathrm{d}\Omega \quad j = 1, \ldots, n. \tag{3.3}$$

This results in a $n \times n$ system of linear equations where the unknowns are the coefficients u_j in the approximating sum.

3.1.1 Weighted residual methods

More specifically, the general idea of FEM is actually to find the coefficients u_i in the approximated solution (3.2) by minimizing the error $u - \hat{u}$. The actual error $u - \hat{u}$ is itself an unknown. We therefore work with the *residual* R obtained by putting the approximated $\hat{u}$ into the PDE instead of u. Reformulating the PDE to

$$\mathcal{L}(u) = Lu - f = 0. \tag{3.4}$$

If the approximation is inserted into this PDE, we generally have that $\mathcal{L}(\hat{u}) \neq 0$. This error is referred to as residual R:

$$R = \mathcal{L}(\hat{u}). \tag{3.5}$$

Furthermore, we assume that a small error in the residual implies a small error in the error $u - \hat{u}$. We are now interested in determining $u_1, \ldots, u_n$ by requiring that $\mathcal{L}(\hat{u}) = R(u_1, \ldots, u_n; \boldsymbol{x})$ to be small in the following different senses:

Least-Squares Method: This method minimizes the average square of the residual with respect to the unknowns $u_1, \ldots, u_n$:

$$\frac{\partial}{\partial u_j} \int_\Omega R^2 \mathrm{d}\Omega = \int_\Omega 2R \frac{\partial R}{\partial u_j} \mathrm{d}\Omega = 0, \quad j = 1, \ldots, n.$$

Weighted Residual Method: This method minimizes a weighted mean of the residual

$$\int_\Omega R W_j \mathrm{d}\Omega = 0, \quad j = 1, \ldots, n.$$

The functions W_j are often referred to as test functions.

Galerkin: Letting the test functions W_j be the same functions as the basis functions used in approximating the solution, we have the Galerkin method:

$$\int_\Omega R N_j \mathrm{d}\Omega = 0, \quad j = 1, \ldots, n.$$

Petrov-Galerkin: For instance in convection-diffusion equations the different terms might be weighted by different weighting functions. Weighting the diffusion term by the basis functions N_j and the convective term by a perturbed weighting function

$$W_j = N_j + \tau N_j'$$

gives us the Petrov-Galerkin method.

These are some variants of the simplest type of FEM. Working with the residual results in an equivalent formulation as (3.3), with only the test functions varying.

These variants of FEM discretize the problems in space. Extending the method to be used for time dependent problems usually consists of approximating the time derivative by finite difference and some weighted residual method in space. For further description the reader is referred to [11] or other literature on the topic. One of further variants of FEM, the mixed finite element method, will be developed in a later section.

3.1.2 Boundary conditions

If the PDE (3.1) has homogeneous Dirichlet boundary conditions, that is $u = 0$ on $\partial\Omega$, then we can require that the basis functions, by which we approximate the solution, also are equal to 0 on the boundary, $N_j = 0$ on $\partial\Omega\ \forall j$. In the case of non-homogeneous Dirichlet boundary conditions, $u = \tilde{g}(x) \neq 0$ on $\partial\Omega$, we can introduce a function $g(x)$ which equals $\tilde{g}(x)$ on $\partial\Omega$ and then use the expansion

$$\hat{u}(x) = g(x) + \sum_{i=1}^{n} N_i u_i$$

to approximate the solution, still requiring that $N_i = 0$ on the boundary. $g(x)$ is not uniquely defined. It might be replaced by another function which equals the correct boundary value $\tilde{g}$ on the boundary.

Anyhow, the boundary conditions nicely come into the formulation when integrating by parts. For the purpose of illustrating integration by parts we let the differential operator L in (3.1) be the Laplace operator. We therefore have the Poisson equation

$$-\Delta u = f \quad \text{in } \Omega. \tag{3.6}$$

For the purpose of illustration, we set the boundary conditions to be a combination of Dirichlet and Neuman boundary conditions:

$$u = g_1 \quad \text{on } \partial\Omega_E \tag{3.7}$$

$$\frac{\partial u}{\partial n} = g_2 \quad \text{on } \partial\Omega_N \tag{3.8}$$

where $\partial\Omega_E$ and $\partial\Omega_N$ are disjoint parts of the boundary. Now the basis functions N_i used in approximating the solution, are required to be equal to 0 only on $\partial\Omega_E$.

We restate the Galerkin finite element method

$$-\int_\Omega \Delta\hat{u} N_j \mathrm{d}\Omega = \int_\Omega f N_j \mathrm{d}\Omega, \quad j = 1, \ldots, n.$$

Applying integration by parts (Green's theorem) on the left hand side gives

$$\int_\Omega \nabla\hat{u} \cdot \nabla N_j \mathrm{d}\Omega - \int_{\partial\Omega} \frac{\partial\hat{u}}{\partial n} N_j \ \mathrm{d}\Gamma = \int_\Omega f N_j \ \mathrm{d}\Omega, \quad j = 1, \ldots, n.$$

Taking a closer look at the boundary term, we see that on the part $\partial\Omega_N$, our boundary condition (3.8), the normal derivative, fits right in. The basis functions N_j are not required to be 0 here, so the boundary term does not immediately vanish. However, as we required the N_j functions to be equal to 0 on the boundary $\partial\Omega_E$, where we have Dirichlet boundary conditions, the boundary term for this part vanishes.

We are then left with the boundary term

$$\int_{\partial\Omega_N} g_2 N_j \ \mathrm{d}\Gamma, \quad j = 1, \ldots, n.$$

A convention for naming boundary conditions is that boundary conditions involving normal derivatives are called natural boundary conditions, therefore the subscript N in $\partial\Omega_N$. While for boundary conditions like $u = g_1$, the convention is to call it essential boundary condition, thus $\partial\Omega_E$.

3.1.3 The final system of linear equations

As we saw in Section 3.1, FEM gives a system of linear equations. Incorporating the boundary conditions and applying integration by parts, give us the final system of equations for the Galerkin method:

$$\sum_{i=1}^{n} \int_{\Omega} \nabla N_i \cdot \nabla N_j \; u_i \; \mathrm{d}\Omega = \int_{\Omega} f N_j \; d\Omega + \int_{\partial\Omega_N} g_2 N_j \; \mathrm{d}\Gamma, \quad j = 1, 2, \ldots, n$$

where $u_i,\ i = 1, \ldots, n$ are the unknowns. Here the formula is based on the illustrating Poisson problem in the previous Section 3.1.2. We thus have a linear system $\boldsymbol{Au} = \boldsymbol{b}$, where

$$\boldsymbol{A}_{i,j} = \int_{\Omega} \nabla N_i \cdot \nabla N_j \; d\Omega, \quad \boldsymbol{b}_j = \int_{\Omega} f N_j \; \mathrm{d}\Omega + \int_{\partial\Omega_N} g_2 N_j \; \mathrm{d}\Gamma.$$

From linear algebra we have several methods for solving linear systems on the form $\boldsymbol{Au} = \boldsymbol{b}$, from the simple Gauss elimination to more advanced iterative solvers.

The section on implementation of FEM will restate the system of linear equations, as well as describing characteristic features of the system arising from FEM.

3.2 The mathematical approach to FEM

As mentioned introductorily, the mathematical foundation of FEM consists to a large degree of using functional analysis. In the following sections the method will be approached through functional analysis. Definitions and theorems stated are taken from Evans, [7], or Brenner and Scott, [3].

The following presentation is not meant to be a complete mathematical background to FEM, however, it is meant to give a more solid approach to FEM, showing that a solution to FEM indeed exists.

3.2.1 Weak solutions

In Evans (page 7) we read the following definition: A partial differential equation is well-posed if

i) the problem has a solution

ii) the solution is unique

iii) the solution depends continuously on the data given in the problem.

This definition requires also a precise definition of what we mean by "solution" and "depends continuously": For instance should the solution be analytic or infinitely differentiable or only k times differentiable for a PDE of order k. When requiring the solution to depend continuously on the data given, we also need to specify the norm.

A solution of such a well-posed PDE of order k, which is at least k times differentiable, is referred to as a *classical solution*. Classical solutions often exist if the boundary of the underlying domain is sufficiently smooth and if necessary boundary conditions are given.

However, not all PDEs do have classical solutions. For instance the conservation law has no classical solution, but is well-posed if allowing wider solution space. In such cases we may search for solutions in a wider class of candidates for solutions by relaxing the requirements on regularity (smoothness) and finding so-called *weak solutions*.

The idea of a weak solution starts with the weak partial derivative and we also need to have proper function spaces in place before moving on to the definition of weak solution:

Definition of weak partial derivative: Suppose $u, v \in L^1_{loc}(\Omega)$, α being a multi-index. We say that v is the α^{th}-weak partial derivative of u written

$$D^\alpha u = v$$

provided

$$\int_\Omega u D^\alpha \phi \mathrm{d}x = (-1)^{|\alpha|} \int_\Omega v\phi \mathrm{d}x$$

for all test functions $\phi \in C_c^\infty(\Omega)$.

With the introduction of weak partial derivatives, we can now introduce wider spaces of functions, the Sobolev spaces, in which we seek the weak solutions.

Definition of Sobolev spaces: The Sobolev space

$$W^{k,p}(\Omega)$$

consists of all locally summable functions $u : \Omega \to \mathbb{R}$ such that for each multi-index α with $|\alpha| \leq k$, $D^\alpha u$ exits in the weak sense and belongs to $L^p(\Omega)$.

Moreover, if $p = 2$, a usual convention is to write

$$H^k(\Omega) = W^{k,2}(\Omega) \quad (k = 0, 1, \dots).$$

The letter H for Hilbert space is used in honour of David Hilbert. Such Sobolev spaces are the wider spaces in which we seek weak solutions.

In the following we will assume

$$a^{ij}, b^i, c \in L^\infty(\Omega) \quad (i, j = 1, \dots, n)$$

and

$$f \in L^2(\Omega).$$

We now extend the concept of weak partial derivative to PDEs: Considering again the general elliptic PDE from Section 3.1:

$$Lu = f \text{ in } \Omega \tag{3.9}$$

$$u = 0 \text{ on } \partial\Omega \tag{3.10}$$

where Ω is an open, bounded subset of $\mathbb{R}^n$ and u is the unknown function and L is a second-order partial differential operator. This operator might be on either divergence or non-divergence form. We will in the following consider the divergence form:

$$Lu = -\sum_{i,j=1}^{n} (a^{ij}(x)u_{x_i})_{x_j} + \sum_{i=1}^{n} b^i(x)u_{x_i} + c(x)u \tag{3.11}$$

where a^{ij}, b^i and c are given coefficient functions.

We define the *bilinear form* $B[\ ,\ ]$ associated with the divergence form of L as

$$B[u, v] := \int_\Omega \sum_{i,j=1}^{n} a^{ij} u_{x_i} v_{x_j} + \sum_{i=1}^{n} b^i u_{x_i} v + cuv \ \mathrm{d}x \tag{3.12}$$

for $u, v \in H_0^1(\Omega)$. Note that this form comes from multiplying the left hand side of the PDE Lu by a smooth test function $v \in C_c^\infty(\Omega)$ and integrating over the domain Ω and

finally integrating by parts the first term. The boundary condition $u = 0$ on $\partial\Omega$ causes the boundary term to vanish.

Moreover, we note that one derivative has "moved over" to the test function v. This makes it possible to seek solutions by relaxing the requirement on smoothness, since we now are approximating a function which has one less derivative.

Since $u, v \in H_0^1(\Omega)$ we might say that the bilinear form $B[\ ,\ ]$ belongs to the normed, linear space $H_0^1(\Omega) = H_0^1$. Thus the bilinear form takes the property of being bounded (continuous):

$$|B[u,v]| \leq \alpha ||u||_{H_0^1} ||v||_{H_0^1}, \quad \forall u, v \in H_0^1$$

with $\alpha < \infty$, and the property of being coercive on $H_0^1(\Omega)$

$$B[u,u] \geq \beta ||u||_{H_0^1}^2 \quad \forall u \in H_0^1$$

with $\beta > 0$.

Finally we can state the following definition:

Definition of a weak solution: $u \in H_0^1(\Omega)$ is a weak solution of the boundary value problem (3.9) and (3.10) if

$$B[u,v] = (f,v) \tag{3.13}$$

for all $v \in H_0^1(\Omega)$, where $(\ ,\ )$ denotes the inner product in $L^2(\Omega)$. The identity (3.13) is often referred to as the *weak* or *variational* formulation of the original boundary problem. "Weak" comes assumingly from the use of "weak derivative", while the "variational" concept comes from the fact that the function v is allowed to vary arbitrarily.

3.2.2 Existence and uniqueness of weak solutions

The existence and uniqueness of weak solutions are ensured by the following theorem:

Lax-Milgram theorem: Assume H is a real Hilbert space, with norm $||\cdot||$ and inner product $(\ ,\ )$. Assume that

$$B : H \times H \to \mathbb{R}$$

is a bilinear mapping for which there exist constants $\alpha, \beta > 0$ such that

$$|B[u,v]| \leq \alpha ||u||\ ||v|| \quad (u, v \in H)$$

and

$$\beta ||u||^2 \leq B[u,v] \quad (u \in H).$$

In other words, B is a continuous and coercive bilinear form.

Finally, let $f : H \to \mathbb{R}$ be a bounded linear functional on H. Then there exists a unique element $u \in H$ such that

$$B[u,v] = \langle f, v \rangle \tag{3.14}$$

for all $v \in H$. $\langle\ ,\ \rangle$ denotes the pairing of the dual space $H^{-1}(\Omega)$ and $H_0^1(\Omega)$.

This theorem thus ensures our boundary value problem from the previous section to have a unique solution. Formal proofs of theorems and lemmas might be found in several books on FEM.

3.2.3 Variational formulation of elliptic equations

Having the definition of bilinear form and knowing that there exists unique solution to formulations such as equation (3.13), leads us to formulating our mathematical models in a weak or variational formulation.

Revisiting our illustratory Poisson equation:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g_1 \quad \text{on } \Gamma_1 \\ \frac{du}{dn} &= g_2 \quad \text{on } \Gamma_2 \end{aligned} \tag{3.15}$$

The associated bilinear form to the Poisson equation is

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \; d\Omega.$$

Finding the solution then consists of finding $u \in H(\Omega)$ such that

$$u \mid_{\Gamma_1} = g_1$$

and

$$a(u,v) = (f,v)$$

for all $v \in V$, where

$$(f,v) = \int_\Omega fv \; d\Omega + \int_{\Gamma_2} g_2 v \; d\Gamma$$

and

$$V = \{v \in H(\Omega) \; : \; v \mid_{\Gamma_1} = 0\}.$$

Such variational formulation is a usual formulation of and notation in FEM.

3.2.4 Discretization of the variational formulation

In the above sections we have seen how PDEs might be formulated in the weak or variational sense. We have been working with basis and test functions as if they were in a set of infinite functions. However, this is not the case unless we only theoretically analyse the PDEs. In any practical situation, each function space we are working with needs to be *finite* - hence the name finite element method.

We thus need discretized formulations; We discretize the solution u to the discrete u_h and our infinite space H is made finite such that we have $u_h \in V_h \subset H$. A later chapter, Chapter 4 will give examples of finite element spaces. How to actually construct such a finite element space is more complex and thus out of the scope of this thesis.

Now our variational formulation of the Poisson equation becomes: Find $u_h \in V_h$ such that

$$a(u_h, v) = (f,v) \quad \text{for all } v \in V_h. \tag{3.16}$$

Since we now are working with a subset V_h of H, this subset inherits all the properties from the theory above and thus this theory still applies to our discrete formulation (3.16).

We only need to be convinced that our discrete solution u_h does not deviate unsatisfactorily much from u. Cea's lemma ensures this by giving an error estimate on u_h.

Cea's lemma: Suppose the bilinear form B is coercive and continuous, and $V_h \subset H$. If u and u_h are solutions of (3.13) and (3.16) respectively, then

$$||u - u_h||_{H_0^1} \leq C \inf_{v \in V_h} ||u - v||_{H_0^1} \tag{3.17}$$

where $C = \alpha/\beta$ only depends on the coercivity and continuity constants in the bilinear form from the Lax-Milgram theorem.

3.3 Mixed finite element method

In mixed finite element method (MxFEM) different unknowns are approximated by different basis functions. Such a method has its origins in solid mechanics where it often has been desirable to have a more accurate approximation of *derivatives* of the displacement, the primary unknown. Instead of first solving for the primary unknown and then carrying out numerical derivation with the possibility of introducing additional numerical errors, MxFEM allows us to find the desired unknown directly.

As mentioned in the motivation of the research question in Section 2.2, some elasticity models appear in mixed form when applying standard Galerkin FEM. Likewise, systems as the Navier-Stokes equations for viscous fluid flow also result in the mixed method when natural Galerkin approximation is applied. Considering the stationary Stokes equations for steady flow of a viscous fluid:

$$\begin{aligned} -\Delta\bar{u} + \bar{\text{grad}}\, p &= \bar{f} \\ \text{div}\ \bar{u} &= 0 \end{aligned}$$

on the domain Ω with for instance Dirichlet boundary conditions $\bar{u} = 0$ on the boundary $\partial\Omega$. Unlike the Poisson equation which might be written as *one* equation, the Stokes equations are a *system* which is difficult to reduce to one equation.

The unknown $\bar{u}$ represents the fluid velocity, while p denotes the pressure. Both of these unknowns need to be approximated. However, since $\bar{u}$ is a vector and p is a scalar, we obviously need different approximation spaces. Thus natural Galerkin in this system of equations means approximating the two different unknowns by different function spaces; In other words natural Galerkin leads to MxFEM.

Mixed Poisson

We now write our well-known Poisson equation as $-\text{div grad}\ u = f$, with Dirichlet boundary conditions, $u = 0$ on $\delta\Omega$. This can equivalently be written as

$$\begin{aligned} -\text{div}\ \bar{\sigma} &= f \\ \bar{\sigma} &= \text{grad}\ u \end{aligned}$$

on Ω. As we will see later, the essential boundary conditions in the original formulation appear as natural boundary conditions in the mixed formulation.

In the original equation, the primary unknown is u. In the mixed formulation we have two primary unknown simultaneously; u and $\bar{\sigma}$.

These two different unknowns are approximated in different spaces. One obvious difference between the unknowns, is that $\bar{\sigma}$ is a vector, while u is a scalar.

Once we have determined the approximation spaces, we proceed in a similar way as for the Galekin FEM: We approximate the unknowns with a sum of basis functions. In the case of mixed Poisson, let the approximating space for u be V and for $\bar{\sigma}$ we have Σ. We let $v \in V$ and $\bar{\tau} \in \Sigma$:

$$u \approx \hat{u} = \sum_{i=1}^{n} u_i v_i, \qquad \bar{\sigma} \approx \hat{\bar{\sigma}} = \sum_{j=1}^{m} \sigma_j \bar{\tau}_j.$$

Next we multiply each of the equations with the sets of basis functions and integrate over the domain. The fact that we have both a function space of vector functions Σ and scalar function space V, comes in conveniently:

$$\begin{aligned} -\int_\Omega \text{div}\ \hat{\bar{\sigma}}\ v_k\ \mathrm{d}\Omega &= \int_\Omega f v_k\ \mathrm{d}\Omega, \quad \text{for } k = 1, \ldots, n \\ \int_\Omega (\hat{\bar{\sigma}} - \bar{\text{grad}}\ \hat{u}) \cdot \bar{\tau}_l\ \mathrm{d}\Omega &= 0, \quad \text{for } l = 1, \ldots, m. \end{aligned}$$

We apply integration by parts (Green's theorem) on the second term of the last equation, and due to the Dirichlet boundary conditions the boundary term vanishes:

$$-\int_\Omega \bar{\text{grad}}\ \hat{u} \cdot \bar{\tau}_l \ \mathrm{d}\Omega = \int_\Omega \hat{u}\ \text{div}\ \bar{\tau}_l \ \mathrm{d}\Omega - \int_{\delta\Omega} \hat{u}\ \bar{\tau}_l \cdot \bar{n} \ \mathrm{d}\Gamma.$$

Here we see that our originally essential boundary conditions fits right into the boundary integral, thus appears as natural boundary conditions.
Inserting the approximated solutions leads to

$$\sum_{j=1}^{m} \int_\Omega (\text{div}\ \bar{\tau}_j) v_k \ \sigma_j \ \mathrm{d}\Omega = -\int_\Omega f v_k \ \mathrm{d}\Omega, \quad \text{for } k = 1, \ldots, n$$

$$\sum_{j=1}^{m} \int_\Omega \bar{\tau}_j \bar{\tau}_l \ \sigma_j \ \mathrm{d}\Omega + \sum_{i=1}^{n} \int_\Omega v_i \text{div}\ \bar{\tau}_l \ u_i \ \mathrm{d}\Omega = 0, \quad \text{for } l = 1, \ldots, m.$$

The linear system we get

$$\mathcal{A} \begin{bmatrix} \boldsymbol{\sigma} \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\sigma} \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{b} \end{bmatrix} \tag{3.18}$$

where the entries into the block matrices are

$$\begin{aligned} \boldsymbol{A}_{j,l} &= \int_\Omega \bar{\tau}_j \bar{\tau}_l \ \ \mathrm{d}\Omega \\ \boldsymbol{B}^T_{j,k} &= \int_\Omega (\text{div}\ \bar{\tau}_j) v_k \ \mathrm{d}\Omega \\ \boldsymbol{b}_k &= -\int_\Omega f v_k \ \mathrm{d}\Omega \end{aligned} \tag{3.19}$$

and the unknowns are

$$\begin{aligned} \boldsymbol{\sigma} &= [\sigma_1, \ldots, \sigma_m]^T \\ \boldsymbol{u} &= [u_1, \ldots, u_n]^T. \end{aligned}$$

$\boldsymbol{A}$ is a $m \times m$-matrix, while $\boldsymbol{B}^T$ is a $m \times n$-matrix, with $\boldsymbol{B}$ as $n \times m$, and $\mathcal{A}$ is of dimensions $(m+n) \times (m+n)$. The linear system in (3.18) is what is solved to find the solutions of the mixed formulation. As the equation (3.18) shows, the element matrix in the mixed case has a block structure.

3.3.1 Mixed variational formulation of elliptic equations

The essence of mixed variational formulation is that the mathematical model is treated with *two* bilinear forms. Finding the solution thus means finding a pair of solution functions.

For our Poisson equation with Dirichlet boundary conditions we now have the following variational formulation: Given the function $f \in L^2(\Omega)$, find the functions $\bar{\sigma} \in \bar{\Sigma}(\Omega)$ and $u \in V(\Omega)$ such that

$$\begin{aligned} a(\bar{\sigma}, \bar{\tau}) + b(u, \bar{\tau}) &= 0 \qquad \forall\ \bar{\tau} \in \bar{\Sigma}(\Omega), \\ b(v, \bar{\sigma}) &= \langle f, v \rangle \qquad \forall\ v \in V(\Omega) \end{aligned} \tag{3.20}$$

The bilinear forms $a(\cdot,\cdot)$ and $b(\cdot,\cdot)$ are defined as

$$a(\bar{\sigma},\bar{\tau}) := \int_\Omega \bar{\sigma}\cdot\bar{\tau}\ d\Omega$$
$$b(v,\bar{\sigma}) := \int_\Omega (\text{div}\ \bar{\sigma})\ v\ d\Omega$$

and

$$\langle f, v\rangle := \int_\Omega fv\ d\Omega.$$

Same discretizing of the function spaces as in the case of classical FEM leads us to finding $\bar{\sigma}_h \in \bar{\Sigma}_h \subset \bar{\Sigma}(\Omega)$ and $u_h \in V_h \subset V(\Omega)$ such that

$$\begin{aligned} a(\bar{\sigma}_h,\bar{\tau}) + b(u_h,\bar{\tau}) &= 0 \qquad \forall\ \bar{\tau}\in\bar{\Sigma}_h(\Omega),\\ b(v,\bar{\sigma}_h) &= \langle f, v\rangle \qquad \forall\ v\in V_h(\Omega). \end{aligned}$$

This is the variational formulation for the mixed system.

3.3.2 Solvability requirements

In Section 3.2.2 we saw that the classical Galerkin FEM has ensured the existence and uniqueness of a weak solution through the Lax-Milgram theorem. For our MxFEM, the question is when the system (3.18) has a solution, that is when $\mathcal{A}$ is invertible. This requirement is closely related to the approximation spaces. Not all sets of approximating spaces will lead to $\mathcal{A}$ being invertible and thus convergent approximations. There are two conditions which need to be fulfilled to ensure success in solvability of MxFEM, of which the latter is often referred to as the *inf-sup*-condition or the Babuška-Brezzi condition. Before we go deeper into the conditions, we first consider the system (3.18).

As indicated, the matter of the mixed system having a solution depends on the matrix $\mathcal{A}$. This matrix depends in turn on $\boldsymbol{A}$ and $\boldsymbol{B}$. Considering the first equation in the equation set:

$$\boldsymbol{A\sigma} + \boldsymbol{B}^T\boldsymbol{u} = 0 \quad \Rightarrow \quad \boldsymbol{\sigma} = -\boldsymbol{A}^{-1}\boldsymbol{B}^T\boldsymbol{u}$$

Inserting this equation into the second one gives

$$\boldsymbol{B}^T\boldsymbol{\sigma} = -\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B}^T\boldsymbol{u} = \boldsymbol{b}$$

Solving for $\boldsymbol{u}$ require that $-\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B}^T$ is invertible and this is not obvious. The matrix $-\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B}^T$ is often called the pressure Schur complement. If both unknowns are approximated with the same type of element, the pressure Schur complement can become singular. Thus a vital point in the solvability condition for the mixed formulation is that the approximation spaces are different.

As we are so far dealing with the linear system, we introduce the algebraic conditions ensuring an invertible matrix $\mathcal{A}$:

$$\boldsymbol{\sigma}^T\boldsymbol{A\sigma} > 0, \quad \forall\boldsymbol{\sigma}\in\ker(\boldsymbol{B}) \quad (\boldsymbol{A}\text{ sufficiently invertible}) \tag{3.21}$$

and

$$\ker(\boldsymbol{B}^T) = \{0\}, \quad (\boldsymbol{B}^T\boldsymbol{A}^{-1}\boldsymbol{B}^T \text{ is invertible}). \tag{3.22}$$

These solvability conditions are however implied by the stronger Babuška-Brezzi conditions which will be presented in the following.

Since our bilinear forms $a(\cdot,\cdot)$ and $b(\cdot,\cdot)$ are bounded from the definitions of the function spaces Σ and V, we have that

$$a(\bar{\sigma},\bar{\tau}) \leq C||\bar{\sigma}||_{\bar{\Sigma}}||\bar{\tau}||_{\bar{\Sigma}} \quad \forall \bar{\sigma},\bar{\tau} \in \bar{\Sigma}$$
$$b(u,\bar{\tau}) \leq C||u||_V||\bar{\tau}||_{\bar{\Sigma}} \quad \forall \bar{\tau} \in \bar{\Sigma}, u \in V.$$

The norms are the norms associated to the function spaces.

The two following conditions are necessary for solvability of the mixed system. These conditions are not generally fulfilled and the function spaces must be "designed" to meet them.

Condition 1: There exists a constant α independent of h such that $a(\cdot,\cdot)$ is coercive on $\bar{Z}_h$, where $\bar{Z}_h = \{\bar{\tau} \in \bar{\Sigma}_h \ : \ b(v,\bar{\tau}) = 0 \quad \forall v \in V_h\}$:

$$a(\bar{\tau},\bar{\tau}) \geq \alpha||\bar{\tau}||^2_{\bar{\Sigma}} \quad \forall \bar{\tau} \in \bar{Z}_h. \tag{3.23}$$

Condition 2 (The inf-sup condition): There exists a constant β independent of h such that

$$0 < \beta := \inf_{u_h \in V_h} \sup_{\bar{\tau}_h \in \bar{\Sigma}_h} \frac{b(u,\bar{\tau})}{||\bar{\tau}||_{\bar{\Sigma}}||u||_V}. \tag{3.24}$$

These conditions are, as mentioned, the Babuška-Brezzi conditions.

3.4 Main principles in implementation of FEM

This section is based on *Computational Partial Differential Equations - Numerical Methods and Diffpack programming* [11], chapter 2 and 3, which introduces both FEM and Diffpack. The software package Diffpack which will be used later for implementation, uses the below mentioned implementation principles. Therefore these principles are included.

The main principles in implementation of FEM are best described using an example. For simplicity the example will be in one dimension. Let us again assume that the differential operator is the Laplace operator and that we have the Poisson equation in 1D:

$$-u_{xx} = f \quad \text{in } \Omega = (0,1)$$

For the sake of illustration we have a combination of essential and natural boundary conditions:

$$u(0) = g_1 \quad \text{(essential)}$$
$$u_x(1) = g_2 \quad \text{(natural)}$$

In the numerical approach in Section 3.1 we saw that the finite element method resulted in a system of linear equations, equation (3.3). Furthermore we saw how the boundary conditions were incorporated when applying integration by parts. We therefore apply Galerkin finite element method with the basis functions N_j and integrate by parts:

$$\int_\Omega (\sum_{i=1}^n N_i' u_i) N_j' dx - (\sum_{i=1}^n u_i N_i') N_j|_0^1 = \int_\Omega f N_j \ \mathrm{d}x, \quad j = 1,\ldots,n.$$

The indices i and j run through all the indices of the basis functions.

At the boundary node $x = 1$ we have natural boundary conditions. This means that $\sum_{i=1}^n u_i N_i'(1) = g_2$. At the essential boundary node $x = 0$, and due to a desired local support of the basis functions, we require that $N_j(0) = 0$ for all j. Inserting the boundary conditions in the boundary term gives

$$\int_\Omega (\sum_{i=1}^n N_i' u_i) N_j' \ \mathrm{d}x - (g_2 N_j(1) - 0) = \int_\Omega f N_j \ \mathrm{d}x, \quad j = 1,\ldots,n.$$

This gives us a system of linear equations $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$, where

$$\boldsymbol{A}_{i,j} = \int_\Omega N_i' N_j' \, \mathrm{d}x, \quad \boldsymbol{b}_j = \int_\Omega f N_j \, \mathrm{d}x + g_2 N_j(1).$$

and $u_1, u_2, \ldots, u_n$ are the unknowns. $\boldsymbol{A}$ is often referred to as the stiffness matrix. Computation of this matrix is a central part in implementing FEM.

At this point we have arrived at the same stage as we ended up at in Section 3.1.3 and we have a system of equations ready to be implemented and solved by a suitable method for solving linear equations. Such an implementation will be rather direct and we have not reached any of the special principles when implementing FEM.

However, when specializing the problem further with respect to which elements used, that is which basis and test functions used, we might optimize the implementation by knowing that the matrix $\boldsymbol{A}$ might have a special structure. Let us therefore introduce linear elements. A more thorough treatment of different elements will be given in the next chapter.

Linear elements now make the approximated solution a piecewise linear function. Let us take 8 subdomains of $(0, 1)$. That is we divide our domain into 8 subdomains and between these subdomains we have element nodes. These nodes are a nodal basis for the 8 linear basis functions N_j, $j = 1, \ldots, 8$ by which we approximate the solution. Figure 3.1 shows 5 of the 8 linear basis functions in one dimension. Each index j might now also be associated to an element or a subdomain of $\Omega = (0, 1)$.

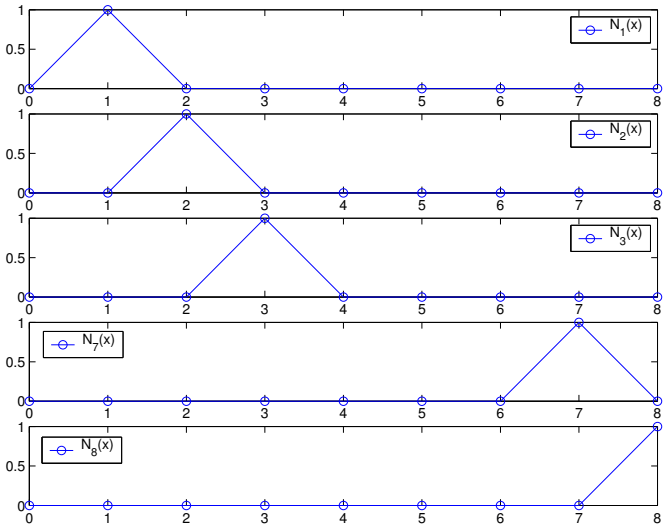


Figure 3.1: Five linear basis functions in 1D. The numbers on the x-axis are the numbers of the element nodes in the domain.

Note also from Figure 3.1 that no linear basis function is non-zero at $x = 0$ (node number 0), but that N_8 take the value 1 at node number 8. This is consistent with our requirements above on the basis functions.

Formula for the basis function $N_j(x)$ is

$$N_j(x) = \begin{cases} 0, & x \leq x^{[j-1]} \\ \frac{1}{h}(x - x^{[j-1]}), & x^{[j-1]} \leq x \leq x^{[j]} \\ -\frac{1}{h}(x - x^{[j+1]}), & x^{[j]} \leq x \leq x^{[j+1]} \\ 0, & x \geq x^{[j+1]} \end{cases} \tag{3.25}$$

where h is the length of each subdomain.

Due to the local support of N_j, we see that N_j is different from 0 only on the elements number j and $j+1$. This makes only a few combinations of N_j's and N_i's in the stiffness matrix non-zero. The combinations of j with $i = j$ and $i = j \pm 1$ will give non-zero entries:

$$\begin{aligned}\boldsymbol{A}_{j,j-1} &= \int_0^1 N'_{i-1}N'_i dx = -\frac{1}{h}\\ \boldsymbol{A}_{j,j} &= \int_0^1 N'_i N'_i dx = \frac{2}{h}\\ \boldsymbol{A}_{j,j+1} &= \int_0^1 N'_i N'_{i+1} dx = -\frac{1}{h}\end{aligned}$$

for $i = 2, \ldots, n-1$. The end elements need special treatment and give

$$\begin{aligned}\boldsymbol{A}_{1,1} = \frac{1}{h}, \quad \boldsymbol{A}_{1,2} = -\frac{1}{h}\\ \boldsymbol{A}_{n-1,n} = -\frac{1}{h}, \quad \boldsymbol{A}_{n,n} = \frac{1}{h}.\end{aligned}$$

Hence we see that the matrix $\boldsymbol{A}$ becomes tri-diagonal and we need only find the entries at the three main diagonals, which saves us a lot of computation.

On the right hand side we have

$$\begin{aligned}\boldsymbol{b}_j &= \int_0^1 f(x)N_j(x)\ \mathrm{d}x, \quad j = 2, \ldots, n-1\\ \boldsymbol{b}_n &= \int_0^1 f(x)N_n(x)\ \mathrm{d}x + g_2 N_n(1).\end{aligned}$$

In all the integrals we need to use some numerical integration rule, known to be exact for the necessary order.

Remembering the essential boundary conditions at $x = 0$ allows us to set the unknown u_1 directly. This gives $u_1 = g_1$. For $i = 2, \ldots, n-1$ we use the computed matrix entries:

$$\frac{1}{h}\begin{bmatrix} 1 & 0 & 0 & & & \ldots & & 0\\ -1 & 2 & -1 & & & & & \\ 0 & -1 & 2 & -1 & & \ddots & & \vdots\\ & & -1 & 2 & -1 & & & \\ \vdots & & \ddots & -1 & 2 & -1 & & \\ & & & & -1 & 2 & -1 & \\ & & & & & -1 & 2 & -1\\ 0 & & \ldots & & & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} u_1\\ u_2\\ \vdots\\ \vdots\\ \vdots\\ u_7\\ u_8 \end{bmatrix} = \begin{bmatrix} g_1\\ b_2\\ \vdots\\ \vdots\\ \vdots\\ b_7\\ b_8 \end{bmatrix}$$

The use of linear elements in FEM results in the same linear system as the finite difference method. However, this coincidence does not occur generally.

3.4.1 Elementwise computation

There is one more modification of the computation of entries to matrix $\boldsymbol{A}$, which makes up an important principle in implementation of FEM.

We observe that the integrands for $\boldsymbol{A}_{j,j(\pm 1)}$ are quite similar when using a regular subdivision of the domain; Actually they are the same except for the fact that the part of

the domain which gives a contribution to the integral, varies with a horizontal shift along the x-axis. We can thus take a closer look at one subdomain and see that all information we need for computing $\boldsymbol{A}_{i,j} = \int_\Omega N_i' N_j'$ is provided by the basis functions defined over this subdomain.

Thus we introduce a reference element with local coordinates ξ and introduce element-wise computation. The reference element for the linear element is shown in Figure 3.2.

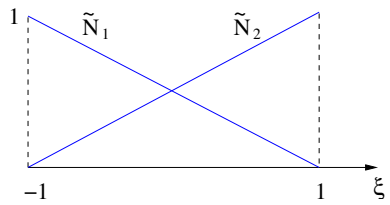


Figure 3.2: Linear reference element in 1D.

We map each subdomain $\Omega_e = [x^{[e]}, x^{[e+1]}]$ into the reference element $[-1, 1]$, by the mapping

$$x^{(e)}(\xi) = \frac{1}{2}\left(x^{[e]} + x^{[e+1]}\right) + \frac{1}{2}\xi\left(x^{[e+1]} - x^{[e]}\right).$$

This is the relation between the global coordinate x and the local reference coordinate ξ. The expression for the two local basis functions are

$$\tilde{N}_1(\xi) = \frac{1}{2}(1-\xi)$$
$$\tilde{N}_2(\xi) = \frac{1}{2}(1+\xi)$$

Now the idea is to compute the stiffness matrix and right hand side for the reference element, that is compute the so-called element matrix, $\boldsymbol{A}^{(e)}_{r,s}$, $r, s = 1, 2$ and element vector $\boldsymbol{b}^{(e)}_r$, $r = 1, 2$. We start with an arbitrary element

$$\int_{x^{[e]}}^{x^{[e+1]}} N_i'(x) N_j'(x)\ \mathrm{d}x.$$

By change of variables we have that

$$\frac{dN_i}{dx} = \frac{d\tilde{N}_r}{d\xi}\frac{d\xi}{dx} = J^{-1}\frac{d\tilde{N}_r}{d\xi}, \quad i = q(e, r).$$

where J is the Jacobian matrix. Thus transforming the integration into reference element coordinates gives

$$\int_{x^{[e]}}^{x^{[e+1]}} N_i'(x) N_j'(x)\ dx = \int_{-1}^{1} J^{-1}\frac{d\tilde{N}_r(\xi)}{d\xi} J^{-1}\frac{d\tilde{N}_s(\xi)}{d\xi}\det J\ \mathrm{d}\xi.$$

Now the explicit expression for each entry in the element matrix for linear elements for one dimension becomes

$$\boldsymbol{A}^{(e)}_{r,s} = \int_{-1}^{1} \frac{2}{h}\tilde{N}_r'(\xi)\frac{2}{h}\tilde{N}_s'(\xi)\frac{h}{2}\mathrm{d}\xi, \quad r, s = 1, 2.$$

We do the same transformation to a reference element for the right hand side, giving us the element vector:

$$\boldsymbol{b}^{(e)}_r = \int_{-1}^{1} f(x^{(e)}(\xi))\tilde{N}_r(\xi)\frac{h}{2}\mathrm{d}\xi.$$

Finally some special treatment of the first and last equations in the system, ensures incorporation of essential boundary conditions.

The next point is to assemble the element matrix and element vector. The entry (r, s) in the element matrix corresponds to the coupling of local node r and s in the reference element. Hence the entry is a contribution to the coupling of basis functions in global node $(i, j) = (q(e, r), q(e, s))$, where q is the mapping from local to global node numbers. In other words, the assembly process takes for each entry in the global system a sum of entries from the element matrix which "belong" to the global entry by the nodal mapping.

Summing up, the main principle in implementing FEM lies in the computation of the stiffness matrix: Calculate the element matrix and element vector and assemble the global system.

With the assembly process, we calculate the integrands only a few times and use the result several times. This requires considerably less computation than calculating several integrals, and the assembly process is therefore a computational advantage.

3.4.2 Extension to non-constant element size

In some problems the domain might be subdivided into elements of different sizes. In one dimension we might have a partition like in Figure 3.3 if the problem requires more finely calculated values near the boundaries.

The elements thus have unequal lengths. From the previous expression, we see that the element matrix and vector only involve the length of the current element. Thus in the assembly process we only have to replace the general h by a element specific $h_e = x^{[e+1]} - x^{[e]}$ in the expression for the element matrix and vector.

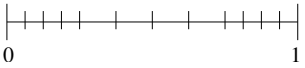


Figure 3.3: Partitioning of the 1D domain (0,1) into elements of unequal length.

3.4.3 Extension to higher order elements

We introduced linear elements with constant element size for the purpose of illustrating the main principle of implementing FEM. Again for the sake of illustrating how we may extend the element matrix to higher order elements, we shortly mention higher order elements:

Quadratic elements consist of basis functions which are second-order polynomials. In the one dimensional case we have three nodes per element and following three basis functions per element. For each element, there are now three basis functions which contribute to the integral in the stiffness matrix. Thus the local element matrix becomes a 3×3-matrix.

Similarly, cubic elements defined with four nodes in the one dimensional reference element and thus four basis functions of order three, have a local element matrix of size 4×4.

From this we see that in computing the element matrix, the number of basis functions in a reference element determines the size of the matrix. Higher order elements result in larger element matrices and thus slightly more computation.

However, the assembly process for higher order elements follows the same strategy. But the mapping between local and global nodes does not necessarily have an easy and explicit formula as in the case of linear elements in one dimension.

Generally, the higher degree of the elements, the better the solution is approximated.

3.4.4 Main principles of implementing the mixed FEM

Since implementing the mixed FEM for elasticity in Diffpack is the main objective of this thesis, a later chapter is owed completely to the details of this implementation. In this section we will only outline the main principles of such an implementation to show the similarity to the standard FEM. The later chapter will take these implementation principles as an undelying basis and build the implementation of the mixed elasticity on top of this basis.

As described above, computation of the element matrix in FEM is a central part of the implementation of this method. Likewise, in mixed FEM the computation of the element matrices is central. Section 3.3 showed that the element matrix in the mixed system has a block structure. Thus this computation in the mixed case is slightly different.

For the linear element in FEM we pointed out the local support of the basis functions which made the element matrix sparse. In addition we saw that regular division of the domain made it possible to restrict computation of integrals to the element matrix, by using a reference element and then undertaking an assembly process. The sparsity of the element matrix depends of course on the function space and we might lack such favourable properties with complicated elements also in the FEM version. But for the mixed method, the element function spaces are usually slightly more complex, such that local support of basis functions and the sparse structure of the matrix are usually not present. However, the element-wise computation with respect to local coordinates applies to mixed FEM and with the following assembling of the element matrices to make up the global linear system. Thus we see that the mixed FEM keeps the same main principles of implementation.

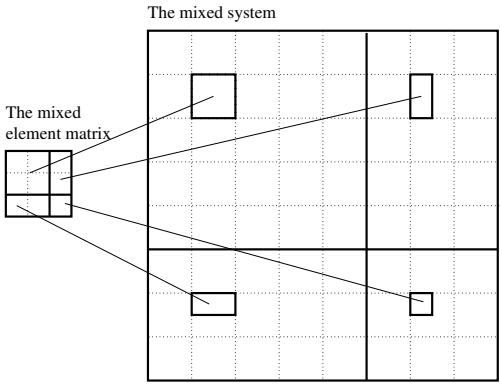


Figure 3.4: The assembly process of the mixed system.

As mentioned, the element matrix in the mixed system has a block structure. The matrix in equation (3.18) reflects this. According to the expressions for the three non-zero blocks, the entries in the element matrices are computed. This is not different from the standard FEM case.

However, the mapping of the element matrix entries to the global matrix entries in the assembly process is not that easy and intuitive. Although the mapping might still be explicitly expressed. Figure 3.4 shows visually the mapping process.

How to enforce the essential boundary condition in FEM was briefly mentioned as setting directly the unknowns associated with the boundary condition. In MxFEM likewise the essential boundary conditions are enforced directly.

As the later Chapter 6 will reveal, there are several more considerations to take when implementing MxFEM in for instance Diffpack. The above mentioned principles are thus only the main and general ones, serving as a basis.

3.5 Strength of FEM

The finite element method has several advantages as a numerical method. Firstly, the method is applicable to almost any field of problems: Heat transfer, stress analysis, magnetic fields and so on.

Furthermore, the strength of FEM appears when complicated geometry is present. Complicated geometry, with curved boundaries, is approximated better by triangles than for instance rectangles. The FEM methods might be applied on any underlying grid, thus might approximate difficult geometry.

As the domain of the problem to be solved is subdivided into elements or subdomains, varying properties or behaviour of the domain might thus be incorporated.

Some drawbacks with the finite element methods are that the methods are relatively complicated to comprehend and not as straightforward as for instance finite difference method. Thus implementation of the methods is not straightforward except in the easiest cases.

Chapter 4

Finite Element Spaces

In the previous chapter the approximation space used to approximate the solution in the finite element method, was simply presented as an available element, the linear element. Furthermore, the extension to higher order elements was just touched upon in the connection to implementation of FEM. In MxFEM, with two unknowns, the idea was said to use a pair of elements. Then each unknown is approximated by a different function space.

In all these cases, no attention is given to why the specific elements are chosen or how these elements are created or found. Which elements used in the finite element methods is actually not a trivial matter; The elements need to be designed such that certain convergence conditions are fulfilled. Hence, choosing proper elements and designing proper elements are important for the finite element methods.

However, these topics are out of the scope of this thesis. In this chapter we will rather give some examples of finite element function spaces. Next we will present and calculate for implementational purposes, the Arnold-Winther element. This part will be given the greatest focus. The tedious computation of the basis functions will be explained in detail. This is because finding the explicit expressions for the basis functions is a central part to the implementation done in Diffpack, as Diffpack requires explicit expressions of the basis functions. This calculation has also been a central part of the work of this thesis. Finally, using the Arnold-Winther element requires some special care. Thus the last sections are dedicated to the special considerations needed to be taken when implementing the basis function in Diffpack.

4.1 Definition of a finite element space

Above in the first sentence of this chapter, the terms "element" and "approximation space" were used to denote the same concept. The word "element" is often used for the function space or approximation space.

A formal definition of a finite element, given by Ciarlet [4] and restated in Brenner and Scott [3] page 67, is given as: Let

i) $K \subset \mathbb{R}^n$ be a domain with piecewise smooth boundary (the element domain),

ii) $\mathcal{P}$ be a finite-dimensional space of functions on K (the shape functions) and

iii) $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ be a basis for $\mathcal{P}'$ (the nodal variables).

Then $(K, \mathcal{P}, \mathcal{N})$ is called a finite element.

A more tangible way of considering an element is to consider a partition of the domain Ω into non-overlapping subdomains Ω_e, e being the index running over the number of such subdomains. In each subdomain we define a set of basis functions (shape functions) N_i with support only in the given subdomain and in its neighbouring subdomains.

These basis functions are in turn associated with a number of local degrees of freedom. When finding the approximated solution based on these basis functions, the global degrees of freedom appear as coefficients in the expansion. The combination of a subdomain, the basis functions in this element and the degrees of freedom, defines a finite element.

By convention, finite elements are given in a local reference subdomain. For instance if the partitioning is a triangulation, a grid of triangles, the reference subdomain is a local triangle with vertices $(0,0)$, $(1,0)$ and $(0,1)$. For each physical subdomain there is a mapping between the physical subdomain and the reference subdomain.

The local evaluation is particularly convenient for implementational purposes and is described in the previous chapter. Diffpack uses such local evaluation of the basis functions.

The choice of basis functions is problem-dependent and they are chosen on basis of regularity requirements of the PDE.

Especially in elasticity, when applying MxFEM, there has been few known suitable stable pair of elements to choose among. One stable pair of elements has now been designed by Douglas Arnold and Ragnar Winther and is presented in their article *Mixed finite elements for elasticity*, [1]. As stated earlier, the implementation of this pair of elements is the aim of this thesis and the basis for being able to undertake the desired numerical experiments and simulations.

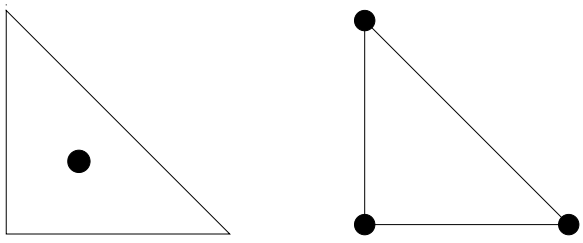
4.1.1 Examples of finite element spaces

Before we present the Arnold-Winther element and explain how we found the basis functions to implement, we consider some easier examples of elements.

Piecewise constant elements

With the piecewise constant element space, each basis function is a constant over a subdomain. There is only one such basis function over the subdomain and the number of basis function nodes is thus one. The basis function is given as

$$N(\xi_1, \xi_2) = 1 \quad \text{for all } \xi_1, \xi_2 \in \Omega_i.$$



(a) Constant element

(b) Linear element

Figure 4.1: Examples of lower order elements.

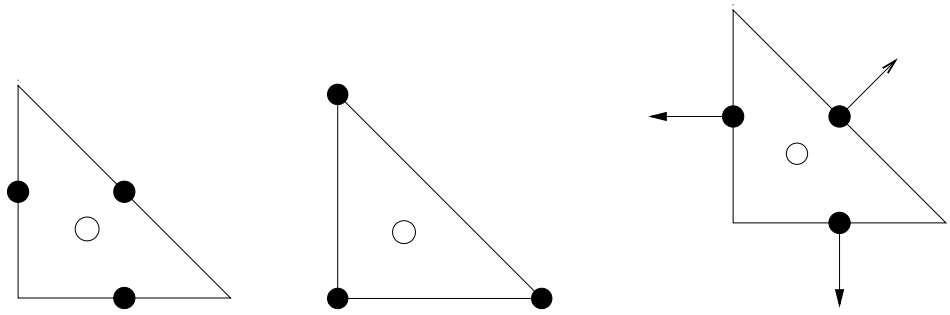
However, we still have three geometry nodes defining the subdomain, or the triangle. ξ refers to the reference coordinates. The location of the basis function node is arbitrary. A usual choice is the node $(1/3, 1/3)$. As the basis function node is an interior node and is not common for two or more subdomains, there is no continuity requirement across the boundary of the subdomains. Figure 4.1(a) shows the degrees of freedom for the element.

Piecewise linear elements

Piecewise linear elements have three linear basis functions over the subdomain.

$$N_1(\xi_1, \xi_2) = 1 - \xi_1 - \xi_2,$$
$$N_2(\xi_1, \xi_2) = \xi_1,$$
$$N_3(\xi_1, \xi_2) = \xi_2,$$

for all ξ_1 and ξ_2 in the subdomain Ω_i. These basis functions are defined from three basis function nodes. The basis function nodes coincides with the geometry nodes, that is the corner nodes of the triangle. Since a basis function node might be shared between two or more subdomains, this element has continuity in the nodal values. Figure 4.1(b) shows the degrees of freedom of the element.



(a) Crouzeix-Raviart element

(b) Mini element

(c) Raviart-Thomas element

Figure 4.2: Examples of mixed elements.

Crouzeix-Raviart element

Moving over to mixed element pairs, we have two sets of basis functions. Each set represents different function spaces and each approximates different unknowns. Often, as in the cases below there is one element particularly designed to meet certain conditions and this element is combined with a well-known or typical element to make up a mixed element pair.

The Crouzeix-Raviart element is linear and continuous at the midpoints of the triangle edges. The three basis functions are

$$N_1(\xi_1, \xi_2) = 1 - 2\xi_2,$$
$$N_2(\xi_1, \xi_2) = 2(\xi_1 + \xi_2) - 1,$$
$$N_3(\xi_1, \xi_2) = 1 - 2\xi_1.$$

This element is combined with discontinuous pressure elements, that is the constant element described above. Figure 4.2(a) shows the degrees of freedom of the element.

Mini element

In our introductory example of the source-sink model, we used the Mini element when solving the model by mixed element method. The Mini element is composed of linear elements both for the pressure and velocity, that is for both unknowns we are solving for. The velocity space is in addition expanded by the "bubble" function to meet the Babuška-Brezzi condition. The bubble function is defined as

$$N_4(\xi_1, \xi_2) = 27(1 - \xi_1 - \xi_2)\xi_1\xi_2,$$

while the three first basis functions are the same as for the linear element. Figure 4.2(b) shows the element.

Raviart-Thomas element

A frequently used mixed element for scalar elliptic problems is the Raviart-Thomas element. Having $\Omega \subset \mathbb{R}^2$ then the function space is defined as

$$RT_h := \{v \in L_2(\Omega);\ v|_T = \begin{pmatrix} a_T \\ b_T \end{pmatrix} + c_T \begin{pmatrix} x \\ y \end{pmatrix} \text{ for } T \in \mathcal{T}_h \text{ with } a_T, b_T, c_T \in \mathbb{R},$$
$$v \cdot n \text{ is continuous on the inter-element boundaries}\}.$$

The constants a_T, b_T, c_T are determined such that $v \cdot n$ is continuous on the inter-element boundaries. In other words this means that the normal components on the edges are the degrees of freedom and the continuity is preserved in these degrees of freedom. Figure 4.2(c) shows this element.

Since the Raviart-Thomas element is a mixed element, we also need another element to pair up with Raviart-Thomas element in the mixed method. Usually Raviart-Thomas is combined with discontinuous piecewise constants for the scalar unknown.

4.2 Arnold-Winther element, the new mixed element for elasticity with strongly imposed symmetry

Arnold and Winther have presented in their paper *Mixed finite elements for elasticity* [1], a new mixed finite element. We have called this the Arnold-Winther element. The element is based on piecewise linear displacement and piecewise quadratic stresses, augmented by some cubic shape functions. The finite elements on a single triangle $T \subset \Omega$ is defined as

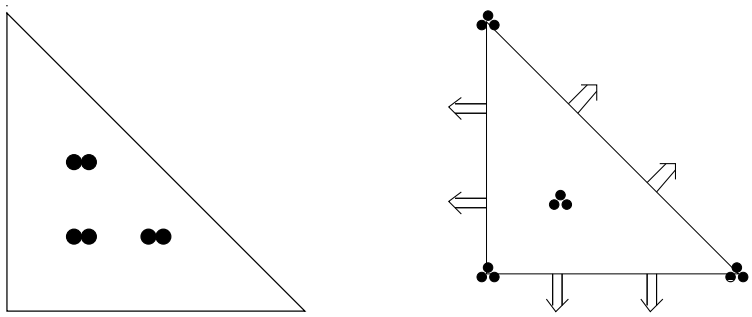
$$\begin{aligned} \Sigma_T &= \mathcal{P}_2(T, \mathbb{S}) + \{\tau \in \mathcal{P}_3(T, \mathbb{S}) \mid \operatorname{div} \tau = 0\} \\ &= \{\tau \in \mathcal{P}_3(T, \mathbb{S}) \mid \operatorname{div} \tau \in \mathcal{P}_1(T, \mathbb{R}^2)\}, \\ V_T &= \mathcal{P}_1(T, \mathbb{R}^2). \end{aligned}$$

The space V_T has dimension 6 and the complete set of degrees of freedom are given by the values of the two components at the three interior nodes in T. The space Σ_T has dimension 24. The degrees of freedom for Σ_T are

- the nodal values of the three components of $\tau(x, y)$ at each vertex (x, y) of T (9 degrees of freedom)

- the values of the moments of degree 0 and 1 of the two normal components of τ on each edge e of T (12 degrees of freedom)
- the value of the moment of degree 0 of the three components of τ on T (3 degrees of freedom).

Figure 4.3 shows the degrees of freedom of the two elements.



(a) Displacement element (b) Stress element

Figure 4.3: Mixed Arnold-Winther elasticity elements.

Computation of the basis functions

We want to compute the basis functions spanning the function spaces V_T and Σ_T. These function spaces will in turn be implemented in Diffpack.

For V_T this means that we want to identify the six basis functions $v_1, v_2, \ldots, v_6$, each with two components. For Σ_T this means that we want to identify the 2×2 symmetric matrices τ, where the elements are functions in P_3. There will be 24 of these.

How the Arnold-Winther function space actually is designed to meet the special stability requirements, is out of the scope of this thesis and will not be described. Likewise, the approximation properties are also out of the scope. Interested readers are referred to *Mixed finite elements for elasticity*, [1]. Error estimates for the elements will be mentioned in Chapter 7 in connection with verification of the elasticity simulator. However, no more details around the error estimates will be given.

4.2.1 Piecewise linear displacement

The function space for the displacement is, as already stated

$$V_T = \mathcal{P}_1(T, \mathbb{R}^2).$$

Let $v \in V_T$ be a basis function and we define the form of the basis function to be

$$v(x, y) = \begin{bmatrix} a + bx + cy \\ d + ex + fy \end{bmatrix}. \tag{4.1}$$

We define the basis function nodes to be

$$q_1 = (1/4, 1/4),$$
$$q_2 = (2/4, 1/4),$$
$$q_3 = (1/4, 2/4).$$

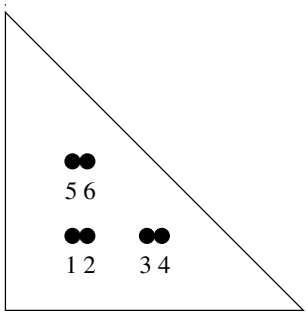


Figure 4.4: Numbering of displacement basis functions.

Figure 4.3(a) shows the degrees of freedom. From this figure we see that the basis function nodes are *internal* nodes and not the same as the geometry nodes. The element is thus non-isoparametric. Figure 4.4 shows the numbering of the degrees of freedom.

To determine the coefficients a, b, c, d, e, f in one of the basis functions, we insert the basis functions nodes into the form of the basis functions, equation (4.1). Then we vary the right hand side to be equal to either $[1,0]^T$, $[0,1]^T$ or $[0,0]^T$. We number the functions according to the degree of freedom number.

The six sets of equations are as follows:

$$\begin{aligned}
v_1(q_1) &= [1,0]^T, \quad v_1(q_2) = [0,0]^T, \quad v_1(q_3) = [0,0]^T \\
v_2(q_1) &= [0,1]^T, \quad v_2(q_2) = [0,0]^T, \quad v_2(q_3) = [0,0]^T \\
v_3(q_1) &= [0,0]^T, \quad v_3(q_2) = [1,0]^T, \quad v_3(q_3) = [0,0]^T \\
v_4(q_1) &= [0,0]^T, \quad v_4(q_2) = [0,1]^T, \quad v_4(q_3) = [0,0]^T \\
v_5(q_1) &= [0,0]^T, \quad v_5(q_2) = [0,0]^T, \quad v_5(q_3) = [1,0]^T \\
v_6(q_1) &= [0,0]^T, \quad v_6(q_2) = [0,0]^T, \quad v_6(q_3) = [0,1]^T
\end{aligned}$$

Each line defines one basis function. We get the following basis functions for the displacement:

$$v_1(x,y) = \begin{bmatrix} 3-4x-4y \\ 0 \end{bmatrix}, \quad v_2(x,y) = \begin{bmatrix} 0 \\ 3-4x-4y \end{bmatrix}, \quad v_3(x,y) = \begin{bmatrix} -1+4x \\ 0 \end{bmatrix},$$
$$v_4(x,y) = \begin{bmatrix} 0 \\ -1+4x \end{bmatrix}, \quad v_5(x,y) = \begin{bmatrix} -1+4y \\ 0 \end{bmatrix}, \quad v_6(x,y) = \begin{bmatrix} 0 \\ -1+4y \end{bmatrix}.$$

4.2.2 Piecewise quadratic stresses, augmented by cubic shape functions

In this subsection we will give the sets of equations defining the basis functions for the stress element. It will be rather long and technical, but none the less necessary for implementation in Diffpack. A lot of work was spent to understand how to calculate the correct basis functions and making all steps clear. This work is thus documented in this chapter.

In the article *FIAT: A new paradigm for computing finite element basis functions*, [8], a new approach for computing a finite element basis is presented. The approach allows us to construct a general class of finite element basis functions from orthonormal polynomials and evaluate and differentiate them at any points.

This approach is aimed at complicated elements such as the stress element of Arnold-Winther – making it easier to test such elements in practice, not only in theory. The

method consists of code for tabulating arbitrary order basis for nearly arbitrary finite elements. "FIAT" in the title of the article, stands for Finite Element Automatic Tabulator.

However, this FIAT approach was not explored in this thesis because Diffpack requires explicit formulas for the basis functions.

We repeat the function space for the stress element:

$$\begin{aligned}\Sigma_T &= \mathcal{P}_2(T,\mathbb{S}) + \{\tau \in \mathcal{P}_3(T,\mathbb{S}) \mid \operatorname{div}\tau = 0\} \\ &= \{\tau \in \mathcal{P}_3(T,\mathbb{S}) \mid \operatorname{div}\tau \in \mathcal{P}_1(T,\mathbb{R}^2)\}\end{aligned}$$

We define the form of a basis function to be

$$\tau = \begin{bmatrix} p(x,y) & q(x,y) \\ q(x,y) & r(x,y) \end{bmatrix} \tag{4.2}$$

where we have the cubic functions

$$\begin{aligned}p(x,y) &= a_p + b_p x + c_p y + d_p x^2 + e_p y^2 + f_p xy + g_p x^3 + h_p y^3 + i_p x^2 y + j_p xy^2, \\ q(x,y) &= a_q + b_q x + c_q y + d_q x^2 + e_q y^2 + f_q xy + g_q x^3 + h_q y^3 + i_q x^2 y + j_q xy^2, \\ r(x,y) &= a_r + b_r x + c_r y + d_r x^2 + e_r y^2 + f_r xy + g_r x^3 + h_r y^3 + i_r x^2 y + j_r xy^2.\end{aligned}$$

The subscript p, q and r in respectively $a_p,\ldots,j_p$, $a_q,\ldots,j_q$ and $a_r,\ldots,j_r$ denote that the coefficients belong to respectively the functions p, q and r. For each basis function τ we therefore have to determine $3\cdot 10 = 30$ coefficients.

However, we know that $\operatorname{div}\tau \in P_1(T,\mathbb{R}^2)$, that is $\operatorname{div}\tau$ is linear:

$$\begin{aligned}\operatorname{div}\tau &= \begin{bmatrix} \frac{\partial p}{\partial x} + \frac{\partial q}{\partial y} \\ \frac{\partial q}{\partial x} + \frac{\partial r}{\partial y} \end{bmatrix} = \begin{bmatrix} p_x(x,y) + q_y(x,y) \\ q_x(x,y) + r_y(x,y) \end{bmatrix} \\ &= \begin{bmatrix} b_p + 2d_p x + f_p y + 3g_p x^2 + 2i_p xy + j_p y^2 + c_q + 2e_q y + f_q x \\ \qquad + 3h_q y^2 + i_q x^2 + 2j_q xy \\ b_q + 2d_q x + f_q y + 3g_q x^2 + 2i_q xy + j_q y^2 + c_r + 2e_r y + f_r x \\ \qquad + 3h_r y^2 + i_r x^2 + 2j_r xy \end{bmatrix}\end{aligned} \tag{4.3}$$

Hence we set the coefficients in front of the second and third order terms in $\operatorname{div}\tau$ equal to zero, and get the following equations

$$\begin{aligned} 3g_p + i_q = 0 \quad &\Rightarrow \quad i_q = -3g_p \\ 2i_p + 2j_q = 0 \quad &\Rightarrow \quad i_p = -j_q = 3h_r \\ j_p + 3h_q = 0 \quad &\Rightarrow \quad j_p = -3h_q \\ \\ 3g_q + i_r = 0 \quad &\Rightarrow \quad i_r = -3g_q \\ 2i_q + 2j_r = 0 \quad &\Rightarrow \quad j_r = -i_q = 3g_p \\ j_q + 3h_r = 0 \quad &\Rightarrow \quad j_q = -3h_r \end{aligned} \tag{4.4}$$

These six equations constitute the six linear constraints.

These equations will be used for each basis function we want to find. Alternatively, we can reduce our originally 30 unknown coefficients in each basis function to 24, by eliminating i_p, j_p, i_q, j_q, i_r and j_r.

We repeat the degrees of freedom for Σ_T:

- the nodal values of the three components of $\tau(x, y)$ at each vertex (x, y) of T (9 degrees of freedom)
- the values of the moments of degree 0 and 1 of the two normal components of τ on each edge e of T (12 degrees of freedom)
- the value of the moment of degree 0 of the three components of τ on T (3 degrees of freedom).

From these degrees of freedom we get for each basis function 24 equations. Together with the six linear constraints above, we get for all 24 basis functions the 30 necessary equations to solve for the 30 unknowns in each basis function.

The numbering strategy of the basis functions is indicated in Figure 4.5: At each vertex we determine three basis functions, on the edges we determine four basis functions and finally three basis functions are determined in the midpoint of the triangle.

At the vertices the basis function of the lowest number, corresponds to the basis function with continuity in the entry (1,1) in the tensor. The next number, corresponds to continuity in components (1,2) and (2,1) (due to symmetry) and the highest number corresponds to continuity in the component (2,2).

At the edges the lowest two numbers corresponds to the moments of degree 0, while the two highest numbers are to the moments of degree 1. Of the two lowest numbers, the first one corresponds to continuity in the first component and the next number corresponds to continuity in the second component. For the moments of degree 1, the interior numbering is similar.

The numbers in the figure equal the subscripts of τ, denoting the number of the basis functions.

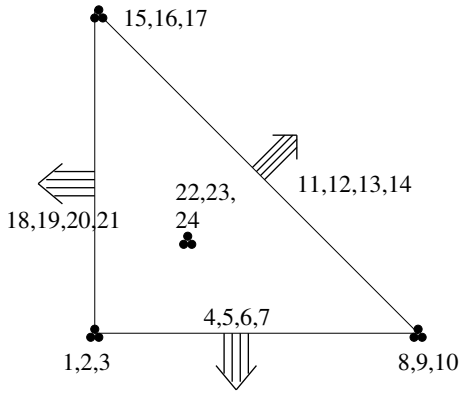


Figure 4.5: Numbering of stress basis functions.

Equations arising from the degrees of freedom at the vertices

We now find the "nodal values of the three components of $\tau(x, y)$ at each vertex (x, y) of T (9 degrees of freedom)". As usual, we let the vertices in the reference triangle be

$$q_1 = (0, 0), \quad q_3 = (1, 0), \quad q_5 = (0, 1).$$

(The symbol q with a subscript indicates a point, $q(x, y)$ without any subscript indicates a function in P_3.)

By inserting the points q_1, q_4 and q_7 into the expression of τ in equation (4.2), we have for any basis function τ that:

$$\tau(0,0) = \begin{bmatrix} p(0,0) & q(0,0) \\ q(0,0) & r(0,0) \end{bmatrix} = \begin{bmatrix} a_p & a_q \\ a_q & a_r \end{bmatrix}$$
$$\tau(1,0) = \begin{bmatrix} p(1,0) & q(1,0) \\ q(1,0) & r(1,0) \end{bmatrix} = \begin{bmatrix} a_p + b_p + d_p + g_p & a_q + b_q + d_q + g_q \\ a_q + b_q + d_q + g_q & a_r + b_r + d_r + g_r \end{bmatrix}$$
$$\tau(0,1) = \begin{bmatrix} p(0,1) & q(0,1) \\ q(0,1) & r(0,1) \end{bmatrix} = \begin{bmatrix} a_p + c_p + e_p + h_p & a_q + c_q + e_q + h_q \\ a_q + c_q + e_q + h_q & a_r + c_r + e_r + h_r \end{bmatrix}$$

The following equations defines 9 of the 24 equations for the basis functions:

$$\tau_1(0,0) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \tau_2(0,0) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \tau_3(0,0) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$
$$\tau_8(1,0) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \tau_9(1,0) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \tau_{10}(1,0) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$
$$\tau_{15}(0,1) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \tau_{16}(0,1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \tau_{17}(0,1) = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that each of these equations listed, gives only three equations (the three components in the symmetric tensor) in a set of equations for determining the unknown coefficients of the basis function with that specific number.

Also note that the numbering of the functions given above corresponds with the numbering in the Figure 4.5.

Equations arising from the degrees of freedom on the edges

Next we want to find "the values of the moments of degree 0 and 1 of the two normal components of τ on each edge e of T (12 degrees of freedom)". We are thus defining the twelve basis functions $\tau_4, \ldots, \tau_7$, $\tau_{11}, \ldots, \tau_{14}$ and $\tau_{18}, \ldots, \tau_{21}$ (see Figure 4.5).

The computation of these basis functions associated with moments of degree 0 and 1, is slightly more complicated than for the nodal values.

First we multiply a general basis function with the unit normal vector on each edge. Then we integrate over the domain, here in each case reduced to the edge, to find the moments of degree 0 and 1.

When finding the moment of degree 0, we just integrate each element in the tensor over the domain (which is only an edge). When finding the moment of degree 1, we first multiply each element in the matrix with the variable we are integrating over, then integrate over the edge.

Throughout all the computations, we will use unit normal vectors pointing *outwards* of the reference triangle.

On the edge $y = 0$: First we multiply with the unit normal vector $[0, -1]^T$ and use the fact that $y = 0$ on this edge:

$$\tau \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -q(x,y) \\ -r(x,y) \end{bmatrix} =_{(y=0)} \begin{bmatrix} -(a_q + b_q x + d_q x^2 + g_q x^3) \\ -(a_r + b_r x + d_r x^2 + g_r x^3) \end{bmatrix}$$

To find the moment of degree 0:

$$\int_0^1 \tau \begin{bmatrix} 0 \\ -1 \end{bmatrix} dx = \begin{bmatrix} -\int_0^1 q(x,0)\ dx \\ -\int_0^1 r(x,0)\ dx \end{bmatrix} = \begin{bmatrix} -(a_q + \frac{1}{2}b_q + \frac{1}{3}d_q + \frac{1}{4}g_q) \\ -(a_r + \frac{1}{2}b_r + \frac{1}{3}d_r + \frac{1}{4}g_r) \end{bmatrix}$$

The moment of degree 1:

$$\int_0^1 \tau \begin{bmatrix} 0 \\ -1 \end{bmatrix} x\ dx = \begin{bmatrix} -\int_0^1 q(x,0)x\ dx \\ -\int_0^1 r(x,0)x\ dx \end{bmatrix} = \begin{bmatrix} -(\frac{1}{2}a_q + \frac{1}{3}b_q + \frac{1}{4}d_q + \frac{1}{5}g_q) \\ -(\frac{1}{2}a_r + \frac{1}{3}b_r + \frac{1}{4}d_r + \frac{1}{5}g_r) \end{bmatrix}$$

Now we set the expressions of the moments of degree 0 and 1 equal to $[1,0]^T$ and $[0,1]^T$ to get the definition of four basis functions:

$$\int_0^1 \tau_4(x,0) \begin{bmatrix} 0 \\ -1 \end{bmatrix} dx = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^1 \tau_5(x,0) \begin{bmatrix} 0 \\ -1 \end{bmatrix} dx = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$\int_0^1 \tau_6(x,0) \begin{bmatrix} 0 \\ -1 \end{bmatrix} x\ dx = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^1 \tau_7(x,0) \begin{bmatrix} 0 \\ -1 \end{bmatrix} x\ dx = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

For each τ_i the rest of the degrees of freedom are set to equal 0.

On the edge $(x, 1-x)$*:* First we multiply with the unit outward-pointing normal vector $1/\sqrt{2}[1,1]^T$ and use the fact that $y = 1 - x$:

$$\begin{aligned} \tau \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} &= \frac{1}{\sqrt{2}} \begin{bmatrix} p(x,1-x) + q(x,1-x) \\ q(x,1-x) + r(x,1-x) \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} a_p + c_p + e_p + h_p + (b_p - c_p - 2e_p + f_p - 3h_p + j_p)x \\ + (d_p + e_p - f_p + 3h_p + i_p - 2j_p)x^2 + (g_p - h_p - i_p + j_p)x^3 \\ + a_q + c_q + e_q + h_q + (b_q - c_q - 2e_q + f_q - 3h_q + j_q)x \\ + (d_q + e_q - f_q + 3h_q + i_q - 2j_q)x^2 + (g_q - h_q - i_q + j_q)x^3 \\ a_q + c_q + e_q + h_q + (b_q - c_q - 2e_q + f_q - 3h_q + j_q)x \\ + (d_q + e_q - f_q + 3h_q + i_q - 2j_q)x^2 + (g_q - h_q - i_q + j_q)x^3 \\ + a_r + c_r + e_r + h_r + (b_r - c_r - 2e_r + f_r - 3h_r + j_r)x \\ + (d_r + e_r - f_r + 3h_r + i_r - 2j_r)x^2 + (g_r - h_r - i_r + j_r)x^3 \end{bmatrix} \end{aligned}$$

Then we find the moments of degree 0 and 1.

Moment of degree 0:

$$\begin{aligned} \int_0^1 \tau \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} dx &= \frac{1}{\sqrt{2}} \int_0^1 \begin{bmatrix} p(x,1-x) + q(x,1-x) \\ q(x,1-x) + r(x,1-x) \end{bmatrix} dx \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} a_p + \frac{1}{2}b_p + \frac{1}{2}c_p + \frac{1}{3}d_p + \frac{1}{3}e_p + \frac{1}{6}f_p + \frac{1}{4}g_p + \frac{1}{4}h_p + \frac{1}{12}i_p + \frac{1}{12}j_p \\ + a_q + \frac{1}{2}b_q + \frac{1}{2}c_q + \frac{1}{3}d_q + \frac{1}{3}e_q + \frac{1}{6}f_q + \frac{1}{4}g_q + \frac{1}{4}h_q + \frac{1}{12}i_q + \frac{1}{12}j_q \\ a_q + \frac{1}{2}b_q + \frac{1}{2}c_q + \frac{1}{3}d_q + \frac{1}{3}e_q + \frac{1}{6}f_q + \frac{1}{4}g_q + \frac{1}{4}h_q + \frac{1}{12}i_q + \frac{1}{12}j_q \\ + a_r + \frac{1}{2}b_r + \frac{1}{2}c_r + \frac{1}{3}d_r + \frac{1}{3}e_r + \frac{1}{6}f_r + \frac{1}{4}g_r + \frac{1}{4}h_r + \frac{1}{12}i_r + \frac{1}{12}j_r \end{bmatrix} \end{aligned}$$

Moment of degree 1:

$$\begin{aligned} \int_0^1 \tau \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} x\ dx &= \frac{1}{\sqrt{2}} \int_0^1 \begin{bmatrix} (p(x,1-x) + q(x,1-x))x \\ (q(x,1-x) + r(x,1-x))x \end{bmatrix} dx \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2}a_p + \frac{1}{3}b_p + \frac{1}{6}c_p + \frac{1}{4}d_p + \frac{1}{12}e_p + \frac{1}{12}f_p + \frac{1}{5}g_p + \frac{1}{20}h_p + \frac{1}{20}i_p + \frac{1}{30}j_p \\ + \frac{1}{2}a_q + \frac{1}{3}b_q + \frac{1}{6}c_q + \frac{1}{4}d_q + \frac{1}{12}e_q + \frac{1}{12}f_q + \frac{1}{5}g_q + \frac{1}{20}h_q + \frac{1}{20}i_q + \frac{1}{30}j_q \\ \frac{1}{2}a_q + \frac{1}{3}b_q + \frac{1}{6}c_q + \frac{1}{4}d_q + \frac{1}{12}e_q + \frac{1}{12}f_q + \frac{1}{5}g_q + \frac{1}{20}h_q + \frac{1}{20}i_q + \frac{1}{30}j_q \\ + \frac{1}{2}a_r + \frac{1}{3}b_r + \frac{1}{6}c_r + \frac{1}{4}d_r + \frac{1}{12}e_r + \frac{1}{12}f_r + \frac{1}{5}g_r + \frac{1}{20}h_r + \frac{1}{20}i_r + \frac{1}{30}j_r \end{bmatrix} \end{aligned}$$

We set these two expressions equal to $[1,0]^T$ and $[0,1]^T$ and get the definitions of four new basis functions

$$\int_0^{\sqrt{2}} \tau_{11}(x,1-x) \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} dx = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^{\sqrt{2}} \tau_{12}(x,1-x) \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} dx = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$
$$\int_0^{\sqrt{2}} \tau_{13}(x,1-x) \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} x\ dx = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^{\sqrt{2}} \tau_{14}(x,1-x) \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} x\ dx = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

On the edge $x = 0$*:* Multiplying by the outward unit normal vector $[-1,0]^T$ and inserting $x = 0$:

$$\tau \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -p(x,y) \\ -q(x,y) \end{bmatrix} =_{(x=0)} \begin{bmatrix} -(a_p + c_p y + e_p y^2 + h_p y^3) \\ -(a_q + c_q y + e_q y^2 + h_q y^3) \end{bmatrix}$$

Finding the moment of degree 0:

$$\int_0^1 \tau \begin{bmatrix} -1 \\ 0 \end{bmatrix} dy = \begin{bmatrix} -\int_0^1 p(0,y)\ dy \\ -\int_0^1 q(0,y) dy \end{bmatrix} = \begin{bmatrix} -(a_p + \frac{1}{2}c_p + \frac{1}{3}e_p + \frac{1}{4}h_p) \\ -(a_q + \frac{1}{2}c_q + \frac{1}{3}e_q + \frac{1}{4}h_q) \end{bmatrix}$$

Moment of degree 1:

$$\int_0^1 \tau \begin{bmatrix} -1 \\ 0 \end{bmatrix} y\ dy = \begin{bmatrix} -\int_0^1 p(0,y) y\ dy \\ -\int_0^1 q(0,y) y\ dy \end{bmatrix} = \begin{bmatrix} -(\frac{1}{2}a_p + \frac{1}{3}c_p + \frac{1}{4}e_p + \frac{1}{5}h_p) \\ -(\frac{1}{2}a_q + \frac{1}{3}c_q + \frac{1}{4}e_q + \frac{1}{5}h_q) \end{bmatrix}$$

Again we set the two expressions equal to $[1,0]^T$ and $[0,1]^T$ and get the four definitions:

$$\int_0^1 \tau_{18}(0,y) \begin{bmatrix} -1 \\ 0 \end{bmatrix} dy = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^1 \tau_{19}(0,y) \begin{bmatrix} -1 \\ 0 \end{bmatrix} dy = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$
$$\int_0^1 \tau_{20}(0,y) \begin{bmatrix} -1 \\ 0 \end{bmatrix} y\ dy = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^1 \tau_{21}(0,y) \begin{bmatrix} -1 \\ 0 \end{bmatrix} y\ dy = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Equations arising from the moments of degree 0 on the triangle

Finally we find "the value of the moment of degree 0 of the three components of τ on T (3 degrees of freedom)". Finding the moment of degree 0 over the triangle means integrating the basis function over the whole triangle. The three degrees of freedom appear from setting this integral equal to different tensors, reflecting the different components being continuous.

Firstly, writing out the integral expression

$$\int_0^1 \int_0^{1-x} \tau(x,y) = \begin{bmatrix} \int_0^1 \int_0^{1-x} p(x,y) & \int_0^1 \int_0^{1-x} q(x,y) \\ \int_0^1 \int_0^{1-x} q(x,y) & \int_0^1 \int_0^{1-x} r(x,y) \end{bmatrix}$$
$$= \begin{bmatrix} \frac{1}{2}a_p + \frac{1}{6}b_p + \frac{1}{6}c_p + \frac{1}{12}d_p + \frac{1}{12}e_p & a_q + \frac{1}{6}b_q + \frac{1}{6}c_q + \frac{1}{12}d_q + \frac{1}{12}e_q \\ +\frac{1}{24}f_p + \frac{1}{20}g_p + \frac{1}{20}h_p + \frac{1}{60}i_p + \frac{1}{60}j_p & +\frac{1}{24}f_q + \frac{1}{20}g_q + \frac{1}{20}h_q + \frac{1}{60}i_q + \frac{1}{60}j_q \\ a_q + \frac{1}{6}b_q + \frac{1}{6}c_q + \frac{1}{12}d_q + \frac{1}{12}e_q & a_r + \frac{1}{6}b_r + \frac{1}{6}c_r + \frac{1}{12}d_r + \frac{1}{12}e_r \\ +\frac{1}{24}f_q + \frac{1}{20}g_q + \frac{1}{20}h_q + \frac{1}{60}i_q + \frac{1}{60}j_q & +\frac{1}{24}f_r + \frac{1}{20}g_r + \frac{1}{20}h_r + \frac{1}{60}i_r + \frac{1}{60}j_r \end{bmatrix}$$

Then setting this as the left hand side with the different tensors for the right hand side:

$$\int_0^1 \int_0^{1-x} \tau_{22}(x,y)\ dy\ dx = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix},$$
$$\int_0^1 \int_0^{1-x} \tau_{23}(x,y)\ dy\ dx = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$
$$\int_0^1 \int_0^{1-x} \tau_{24}(x,y)\ dy\ dx = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

These are thus the three sets of equations defining the three last basis functions.

Altogether we have now identified all of the 30 equations needed to find the basis functions. That is, we have found 24 sets of 30 equations, each set describing one basis function when we set the right hand side appropriately. For every definition, all the other equations are set equal to 0, such that for all of the 24 sets of equations, only one entry in the right hand side is 1.

Solving the linear systems give us the coefficients in the three cubic functions of τ. This gives us the explicit expressions for the 24 tensor basis functions. Before listing all the basis functions in their explicit form in Section 4.3.2 on page 42, some more considerations have to be included.

4.3 Using the Arnold-Winther basis functions

As mentioned in the beginning of this chapter, the use of the Arnold-Winther basis functions is not trivial. Special care has to be taken since normal vector components and tensors are involved. The following two sections will present how to handle the normal vectors and how to calculate correctly with tensors in the element matrix. This includes orientation of normal vectors and edges, and geometric mapping from the reference element to the actual mesh.

4.3.1 Orientation of edges and normal vectors

In practical computation with Arnold-Winther stress basis functions, we need to take special care of the basis functions associated with the edges of the elements. As pointed out in the previous section, when calculating the basis functions with respect to the reference triangle, we used outward pointing normal vectors.

When combining two such reference triangles, one triangle is kept in the same orientation while the other is rotated 180 degrees. Figure 4.6 shows this situation. The outward pointing normal vectors on the common edge are marked for both of the triangles. We see that these normal vectors point in opposite directions. The orientation of the common edge is also opposite for the two triangles.

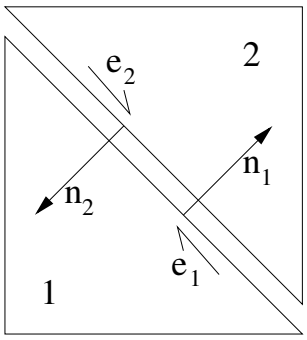


Figure 4.6: The first two subdomains in a mesh, with the direction of the common edge and the normal vectors.

We thus see that using the stress basis functions on an arbitrary triangle in the mesh with no modification concerning the normal vectors, results in the use of basis functions with no common definition for the global normal vectors. Continuity will not be preserved in such a case.

What we want, is a global definition of the normal vectors. We have degrees of freedom associtated to these normal vectors and we need to make sure that these normal vectors are unique for each edge (that means not having two normal vectors for one edge) such that the continuity across the edges exists. The global definition of normal vectors might easily be interpreted to the case that every other element triangle has outward and inward pointing normal vectors. Figure 4.7 illustrates this.

From Figure 4.7 we see that a regularly triangulated mesh will have every other triangle with outward and inward pointing normal vectors. For the stress basis function this means they are computed differently for every other element, with only the direction of the normal vector varying. In addition, the direction of the edge we integrate over, has to be accounted for.

There are two strategies to overcome the challenge of the normal vectors which will be briefly described.

Operate with two normal vectors: This means we have two sets of stress basis functions: One set of functions computed with an outward pointing normal vector and the other set computed with the inward pointing normal vector.

Instead of actually calculating two sets of basis functions, this means that for some integral expressions in the calculations we operate with an extra minus sign. We can thus use the basis function calculated with for instance the outward pointing normal vectors, then add a minus sign in front of the ones assosciated with normal vectors when using the basis function for a 180-degree rotated triangle. This will correspond to the same basis functions just computed with inward pointing normal vectors.

The other strategy for having a common definition of the normal vectors consists of having *orientated edges*. This means that we orient each edge and this orientation will apply globally. With reference to Figure 4.6, this means that for element 1 the global definition of the edge will coincide with the element edge. For element 2, the edge will get the opposite direction, which means an added negative sign. Again we achieve the same effect of the adding of a negative sign, which is that the normal vector has switched direction. This strategy has been implemented in *Overlapping Schwarz preconditioner for the mixed formulation of plane elasticity*, [14], by Yangiu Wang and shown in her documentation [13].

In the case of orientated edges we would have to modify the equations from earlier calculations for the basis functions for moments of degree 1 on the egdes to the following equations:

$$\int_0^1 \tau_6(x,0)\begin{bmatrix} 0 \\ -1 \end{bmatrix}(x-\tfrac{1}{2})\,dx = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^1 \tau_7(x,0)\begin{bmatrix} 0 \\ -1 \end{bmatrix}(x-\tfrac{1}{2})\,dx = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\int_0^{\sqrt{2}} \tau_{13}(x,1-x)\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}(x-\tfrac{\sqrt{2}}{2})\,dx = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^{\sqrt{2}} \tau_{14}(x,1-x)\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}(x-\tfrac{\sqrt{2}}{2})\,dx = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\int_0^1 \tau_{20}(0,y)\begin{bmatrix} -1 \\ 0 \end{bmatrix}(y-\tfrac{1}{2})\,dy = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \int_0^1 \tau_{21}(0,y)\begin{bmatrix} -1 \\ 0 \end{bmatrix}(y-\tfrac{1}{2})\,dy = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Basically this modification is to replace the extra integration variable which comes in with the definitions of moment, that is x with a oriented variable as $(x-\frac{1}{2})$.

However, this latter strategy is not chosen in the implementation. The strategy of adding negative signs in front of appropriate basis functions is chosen. In the chapter on implementation the functions dealing with this will be shown.

In the chosen strategy, in addition to the orientation of the normal vectors, both the direction of the edge we integrate over, and the definition of moment in the expression for the moment of degree, have to be accounted for. Integrating over an edge in a 180-degree rotated triangle means integrating over the edge in the opposite direction than for the

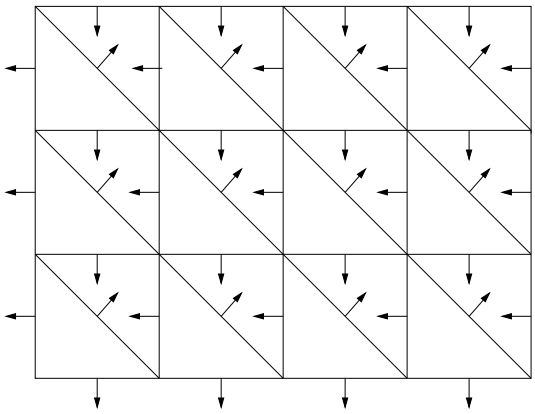


Figure 4.7: Triangulation with global definition of normal vectors.

standard reference element. In case of moment of degree 1, according to the direction of the edge we integrate over, we have to account for possible signs in front of the moment parameter.

Summary of negative signs

In summary, we have that basis functions based on nodal values are used unmodified with respect to signs on all triangles in the global mesh.

Basis functions based on moments of degree 0, when used on a rotated triangle, get a negative sign because of the orientation of the normal vector. The orientation of the egde we integrate over introduces another negative sign to the expression. These two negative signs cancel each other, so in effect we do not add any negativ sign when using the moments of degree 0 basis function on rotated triangles.

This applies also for basis functions based on moments of degree 1. However, in this case we also get a negative sign from the definition of moment (the moment parameter). This negative sign is not cancelled by any other sign, so in effect for these moment of degree 1 basis functions we do need to include a negative sign when using them on rotated triangles in the global mesh.

4.3.2 The matrix Piola transform

For a basis function on a reference element to correspond to the basis function on a triangle element in the actual mesh, the matrix Piola transform is needed. This is basically a geometric mapping.

Let T be a triangel in the domain where the problem is specified and let the global coordinates be x. Let $\hat{T}$ be the reference triangle with the local coordinates $\hat{x}$. Let F be the linear transformation such that

$$x = F\hat{x} = B\hat{x} + b$$

where B is a 2×2 matrix and b is a vector. Figure 4.8 shows the transformation.

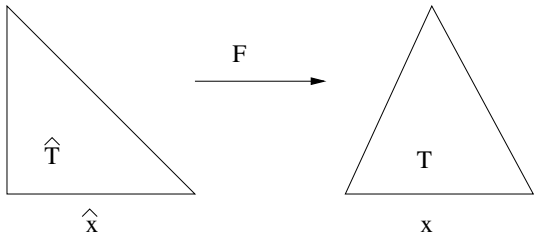


Figure 4.8: The element transformation.

The matrix Piola transformation for matrix functions is defined as

$$\tau(x) = B\hat{\tau}(\hat{x})B^T$$

and by the definition it follows that

$$\text{div } \tau(x) = B \text{ div } \hat{\tau}(\hat{x}).$$

These are the geometric mapping transformations needed for our implementation. As we will only consider regular triangulations in the coming implementation, the matrix B will be

$$B = \begin{bmatrix} \pm h & 0 \\ 0 & \pm h \end{bmatrix}$$

where h is the mesh size. For triangles in the same orientation as the reference triangle, the sign is posivite. For triangles rotated 180 degrees with respect to the reference triangle, the sign is negative.

With the introduction of this matrix Piola transformation, we add further possible negative signs to the implementation of the basis functions. For the mapping for the tensor itself, as the matrix B appear twice in the expression, the two possible negative signs cancel each other. However, this cancellation does not happen for the divergence expression. Thus in the divergence expression, all basis functions get an added negative sign to the expressions when used on a 180-degree rotated triangle.

The 24 Arnold-Winther basis functions for the stress element

$$\sigma_1 = \begin{bmatrix} 1-9\,y-3\,x^2+18\,y^2+9\,x^2y+2\,x^3-10\,y^3 & 6\,xy-6\,x^2y-9\,xy^2 \\ 6\,xy-6\,x^2y-9\,xy^2 & -3\,y^2+6\,xy^2+3\,y^3 \end{bmatrix}$$

$$\sigma_2 = \begin{bmatrix} 9\,x-36\,xy-24\,x^2+45\,x^2y+30\,xy^2+15\,x^3 & 1-9\,x-9\,y+48\,xy+18\,x^2 \\ & +18\,y^2-45\,x^2y-45\,xy^2-10\,x^3-10\,y^3 \\ 1-9\,x-9\,y+48\,xy+18\,x^2 & 9\,y-36\,xy-24\,y^2+30\,x^2y+45\,xy^2+15\,y^3 \\ +18\,y^2-45\,x^2y-45\,xy^2-10\,x^3-10\,y^3 & \end{bmatrix}$$

$$\sigma_3 = \begin{bmatrix} -3\,x^2+6\,x^2y+3\,x^3 & 6\,xy-9\,x^2y-6\,xy^2 \\ 6\,xy-9\,x^2y-6\,xy^2 & 1-9\,x+18\,x^2-3\,y^2+9\,xy^2-10\,x^3+2\,y^3 \end{bmatrix}$$

$$\sigma_4 = \begin{bmatrix} 3\,x^2-9\,x^2y-2\,x^3 & -6\,xy+6\,x^2y+9\,xy^2 \\ -6\,xy+6\,x^2y+9\,xy^2 & 3\,y^2-6\,xy^2-3\,y^3 \end{bmatrix}$$

$$\sigma_5 = \begin{bmatrix} 9\,x^2-18\,x^2y-9\,x^3 & 3\,x-18\,xy-12\,x^2+27\,x^2y+18\,xy^2+10\,x^3 \\ 3\,x-18\,xy-12\,x^2+27\,x^2y+18\,xy^2+10\,x^3 & -3\,y+24\,xy+9\,y^2-30\,x^2y-27\,xy^2-6\,y^3 \end{bmatrix}$$

$$\sigma_6 = \begin{bmatrix} 0 & 0 \\ 0 & 3\,x-12\,x^2+10\,x^3 \end{bmatrix}$$

$$\sigma_7 = \begin{bmatrix} 3\,y-12\,y^2+10\,y^3 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\sigma_8 = \begin{bmatrix} -3\,x+24\,xy+9\,x^2-27\,x^2y-30\,xy^2-6\,x^3 & 3\,y-18\,xy-12\,y^2+18\,x^2y+27\,xy^2+10\,y^3 \\ 3\,y-18\,xy-12\,y^2+18\,x^2y+27\,xy^2+10\,y^3 & 9\,y^2-18\,xy^2-9\,y^3 \end{bmatrix}$$

$$\sigma_9 = \begin{bmatrix} 3\,x^2-6\,x^2y-3\,x^3 & -6\,xy+9\,x^2y+6\,xy^2 \\ -6\,xy+9\,x^2y+6\,xy^2 & 3\,y^2-9\,xy^2-2\,y^3 \end{bmatrix}$$

$$\sigma_{10} = \begin{bmatrix} -6\,x+12\,xy+6\,x^2 & 6\,x-12\,xy-6\,x^2 \\ 6\,x-12\,xy-6\,x^2 & -6\,y+12\,xy+6\,y^2 \end{bmatrix}$$

$$\sigma_{11} = \begin{bmatrix} 0 & 0 \\ 0 & 6\,x-6\,y-6\,x^2+6\,y^2 \end{bmatrix}$$

$$\sigma_{12} = \begin{bmatrix} 24\,x-24\,xy-144\,x^2+180\,x^2y+120\,x^3 & -60\,x+240\,xy+180\,x^2-360\,x^2y-180\,xy^2-120\,x^3 \\ -60\,x+240\,xy+180\,x^2-360\,x^2y-180\,xy^2-120\,x^3 & 60\,y-360\,xy-120\,y^2+360\,x^2y+360\,xy^2+60\,y^3 \end{bmatrix}$$

$$\sigma_{13} = \begin{bmatrix} -36\,x^2+72\,x^2y+36\,x^3 & 72\,xy-108\,x^2y-72\,xy^2 \\ 72\,xy-108\,x^2y-72\,xy^2 & -60\,x+24\,y-48\,xy+180\,x^2-48\,y^2+108\,xy^2-120\,x^3+24\,y^3 \end{bmatrix}$$

$$\sigma_{14} = \begin{bmatrix} -6\sqrt{2}x+12\sqrt{2}xy+6\sqrt{2}x^2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\sigma_{15} = \begin{bmatrix} 0 & 0 \\ 0 & -6\sqrt{2}y+12\sqrt{2}xy+6\sqrt{2}y^2 \end{bmatrix}$$

$$\sigma_{16} = \begin{bmatrix} 24\sqrt{2}xy+24\sqrt{2}x^2-108\sqrt{2}x^2y-24\sqrt{2}x^3 & -72\sqrt{2}xy+72\sqrt{2}x^2y+108\sqrt{2}xy^2 \\ -72\sqrt{2}xy+72\sqrt{2}x^2y+108\sqrt{2}xy^2 & 36\sqrt{2}y^2-72\sqrt{2}xy^2-36\sqrt{2}y^3 \end{bmatrix}$$

$$\sigma_{17} = \begin{bmatrix} -36\sqrt{2}x^2+72\sqrt{2}x^2y+36\sqrt{2}x^3 & 72\sqrt{2}xy-108\sqrt{2}x^2y-72\sqrt{2}xy^2 \\ 72\sqrt{2}xy-108\sqrt{2}x^2y-72\sqrt{2}xy^2 & -24\sqrt{2}xy-24\sqrt{2}y^2+108\sqrt{2}xy^2+24\sqrt{2}y^3 \end{bmatrix}$$

$$\sigma_{18} = \begin{bmatrix} -6\,x+6\,y+6\,x^2-6\,y^2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\sigma_{19} = \begin{bmatrix} -6\,x+12\,xy+6\,x^2 & 6\,y-12\,xy-6\,y^2 \\ 6\,y-12\,xy-6\,y^2 & -6\,y+12\,xy+6\,y^2 \end{bmatrix}$$

$$\sigma_{20} = \begin{bmatrix} -24\,x+60\,y+48\,xy+48\,x^2-180\,y^2-108\,x^2y-24\,x^3+120\,y^3 & -72\,xy+72\,x^2y+108\,xy^2 \\ -72\,xy+72\,x^2y+108\,xy^2 & 36\,y^2-72\,xy^2-36\,y^3 \end{bmatrix}$$

$$\sigma_{21} = \begin{bmatrix} -60\,x+360\,xy+120\,x^2-360\,x^2y-360\,xy^2-60\,x^3 & 60\,y-240\,xy-180\,y^2+180\,x^2y+360\,xy^2+120\,y^3 \\ 60\,y-240\,xy-180\,y^2+180\,x^2y+360\,xy^2+120\,y^3 & -24\,y+24\,xy+144\,y^2-180\,xy^2-120\,y^3 \end{bmatrix}$$

$$\sigma_{22} = \begin{bmatrix} 12\,x-12\,xy-12\,x^2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\sigma_{23} = \begin{bmatrix} 12\,x-24\,xy-12\,x^2 & 12\,xy \\ 12\,xy & 12\,y-24\,xy-12\,y^2 \end{bmatrix}$$

$$\sigma_{24} = \begin{bmatrix} 0 & 0 \\ 0 & 12\,y-12\,xy-12\,y^2 \end{bmatrix}$$

Chapter 5

Elasticity

Before moving over to the implementation of the Arnold-Winther element from the previous chapter, we develop some more expressions for the mixed formulation of the elasticity partial differential equation.

First some basic theory of elasticity will be presented; The first subsection states the mathematical model, while the second develops the model based on physical laws such as Newton's second law and a constitutive relation between the stresses and deformations. The theory is mainly based on *Handbook of Numerical Analysis - Numerical Methods for Solids (Part 1)*, [5], and *The Mathematical Theory of Finite Element Methods*, [3].

Finally, the mixed variational formulation of elasticity is presented. From this formulation the expressions to be implemented are developed. This serves as the explanation to the formulas which are implemented in Chapter 6 on implementation in Diffpack.

Introduction

Finite elasticity models deformation or changes in the geometry of solid bodies. The central part of elasticity consists of finding the equilibrium position of the elastic body when it is subjected to a given applied force.

In standard solid mechanics, the deformations are very small – hardly detectable by the human eye. Thus it makes sense to consider the configuration of the bodies as fixed and neglect any changes in geometry. This assumption is the starting point of linear elasticity. In this small-strain case, the applied stress has been below a characteristic critical limit for the material, and the deformation in which the body returns to its original shape, is referred to as elastic deformation.

In contrast, other situations involve large deformations. A body might be elastic up to a critical limit. However, if the applied stress reaches beyond a characteristic limit, the body undergoes plastic deformation – the body changes shape. This is the case for instance for elasto-viscoplastic solids. In such cases the system becomes non-linear. Further extensions of elasticity are situations where the internal forces depend on both the deformation of the body as well as the history of the body – thus time dependency.

We will in the following only consider linear elasticity.

5.1 The mathematical model

Considering an elastic material, we define the configuration space $\Omega \subset \mathbb{R}^2$. The displacement of the body is represented by $\bar{u}(x)$ and the body force acting on the body is $\bar{f}(x)$.

Then $\bar{u}$ satisfies the equilibrium equation

$$-\bar{\text{div}}\ \bar{\bar{\sigma}}(\bar{u}) = \bar{f} \quad \text{in } \Omega \tag{5.1}$$

where $\bar{\bar{\sigma}}(\bar{u})$ is the stress tensor and is defined by the most common constitutive law, also called Hooke's law

$$\bar{\bar{\sigma}}(\bar{u}) = 2\mu\bar{\bar{\epsilon}}(\bar{u}) + \lambda \text{tr}(\bar{\bar{\epsilon}}(\bar{u}))\bar{\bar{\delta}} \tag{5.2}$$

where μ and λ are the so-called Lamé constants. $\bar{\bar{\epsilon}}(\bar{u})$ is the strain tensor and is related to the displacement by

$$\bar{\bar{\epsilon}}(\bar{u}) = \frac{1}{2}\left(\bar{\text{grad}}\ \bar{u} + (\bar{\text{grad}}\ \bar{u})^T\right). \tag{5.3}$$

The Lamé constants

The Lamé constants λ and μ appear as we have seen, in the relationship between strain (relative displacement) and stress. The constant λ describes the stresses due to change in density. The constant μ, often referred to as shear modulus or rigidity modulus, is a measure of the resistance of a body to shearing strain (deformation without change in volume). The shear modulus of liquids or gases is zero.

λ approaching infinity means we are dealing with a body which is nearly incompressible. In addition when λ approaches infinity, the performance of finite element methods deteriorates. This is known as the locking phenomenon and will be described shortly in Subsection 5.6.

The elasticity constants are given in other forms as well. For instance we have the modulus of elasticity E where the relation is

$$E = \frac{\mu(3\lambda + 2\mu)}{\lambda + \mu}$$

and we have the Poisson ratio ν where the connection is

$$\nu = \frac{\lambda}{2(\lambda + \mu)}.$$

From physical considerations we get the following requirements on the elasticity constants:

$$\lambda > 0, \quad \mu > 0, \quad E > 0, \quad 0 < \nu < 1/2.$$

For many materials we have $\nu \approx 1/3$. For nearly incompressible materials we have $\lambda >> \mu$, that is the Poisson ratio ν is very close to $1/2$. As stated above, this results in locking and that the stiffness matrix in element methods becomes ill-conditioned.

Discontinuous Lamé constants, which our later numerical simulations will be concerned with, means that we have a body of different materials, that is different parts of the body have different material properties.

An example of a body with different material properties and with a sharp boundary where the material properties change, is an object with a "sandwich" structure. That means that the top and bottom layer of the object is made of stiff or hard material, while the inside is made of a light (and not as stiff) material. Such a sandwich structure is shown in Figure 5.1.

Boundary Conditions

Generally in elasticity, there are two possible sets of boundary conditions on the boundary $\partial\Omega$:

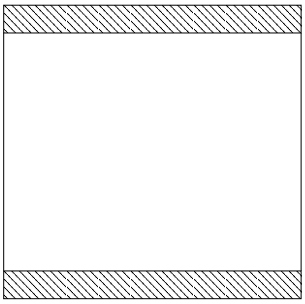


Figure 5.1: Example of a sandwich structure of a material.

- *Displacement* boundary conditions: The displacement $\bar{u}(x)$ is imposed to a prescribed value on a part of the boundary. This condition requires that we know the new displacement after an applied force. For instance if parts of the boundary is clamped, then there is no displacement. In linear elasticity, the whole boundary is clamped, thus there is no displacement.

- *Traction* boundary conditions: The traction force $\bar{t}$ (push or pull) is prescribed on a part of the boundary. This condition is used when the force used to pull or push parts of the body, is known.

Defining Γ_1 and Γ_2 to be open disjoint subsets of the boundary $\partial\Omega$. That is $\Gamma_1 \subset \partial\Omega$ and $\Gamma_2 \subset \partial\Omega$, $\partial\Omega = \Gamma_1 \cap \Gamma_2$ and $\Gamma_1 \cup \Gamma_2 = \emptyset$. Then assigning Γ_1 displacement conditions means

$$\bar{u}|_{\Gamma_1} = \bar{g}.$$

Traction boundary conditions on Γ_2 takes the form

$$(\bar{\bar{\sigma}}(\bar{u})\bar{n})|_{\Gamma_2} = \bar{t}.$$

$\bar{n}$ being the unit outer normal.

Figure 5.2 illustrates an example boundary value problem; On the boundary Γ_1 the object is clamped, that is $\bar{u}|_{\Gamma_1} = 0$. On the boundary Γ_2 there is traction boundary conditions with the traction force $\bar{t}$.

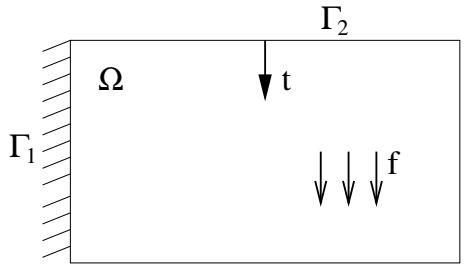


Figure 5.2: Example of a boundary value problem in elasticity.

As mentioned we are solving the elasticity equation for the final equilibrium position $\bar{u}$ of the elastic body after being subjected to deformations due to application of external forces.

5.2 Physical interpretation

The mathematical model of elastic deformation is based on Newton's second law of motion and on a constitutive relation between stress and displacement. Starting with Newton's second law:

$$\text{mass} \cdot \text{acceleration} = \text{net force}$$

We represent the mass through the body density, given by ϱ and defining the acceleration in terms of the velocity $\bar{v}$ as

$$\text{acceleration} = \frac{\partial \bar{v}}{\partial t} + \bar{v} \cdot \nabla \bar{v}$$

We let the sum of forces consist of the net effect of internal stresses, $\nabla \cdot \bar{\bar{\sigma}}$, where $\bar{\bar{\sigma}}$ still is the stress tensor and a general body force, $\varrho \bar{b}$, where $\bar{b}$ is body force. Thus Newton's second law gives us the following partial differential equation:

$$\varrho \left(\frac{\partial \bar{v}}{\partial t} + \bar{v} \cdot \nabla \bar{v} \right) = \nabla \cdot \bar{\bar{\sigma}} + \varrho \bar{b}. \tag{5.4}$$

Constitutive laws, relating forces and deformations, close the system. In our mathematical model we have used Hooke's law as such a constitutive law. Hooke's law states that the stress is directly proportional to the strain:

$$\bar{\bar{\sigma}} = \lambda \text{tr } \bar{\bar{\epsilon}} \, \bar{\bar{\delta}} + 2\mu \bar{\bar{\epsilon}}$$

where $\bar{\bar{\epsilon}}$ is the strain tensor defined above in equation (5.3) and λ and μ are the Lamé constants. Furthermore, for small deformations we have

$$\frac{\partial \bar{u}}{\partial t} \approx \bar{v}.$$

Altogether this yields the classical equation of elasticity:

$$\varrho \frac{\partial^2 \bar{u}}{\partial t^2} = \nabla((\lambda + \mu) \nabla \cdot \bar{u}) + \nabla \cdot (\mu \nabla \bar{u}) + \varrho \bar{b} \tag{5.5}$$

Omitting the left hand term, which is only relevant for describing elastic waves, this equation is the elasticity equation we presented as our mathematical model in Section 5.1.

Other constitutive laws which appear in other models are described in *When the Physics Gets Mixed*, [10]. For instance for fluid models we have the constitutive relation

$$\bar{\bar{\sigma}} = -p \bar{\bar{\delta}} + \mu (\nabla \bar{v} + (\nabla \bar{v})^T)$$

where p is the fluid pressure, $\bar{v}$ is the velocity and $\bar{\bar{\delta}}$ is the identity matrix. For an incompressible fluid the mass conservation principle gives us $\nabla \cdot \bar{v} = 0$. Neglecting the acceleration term (in the equation of motion), results in the well-known Navier-Stokes equations:

$$\mu \nabla^2 \bar{v} - \nabla p = -\rho \bar{b}$$
$$\nabla \cdot \bar{v} = 0$$

In elasto-viscoplastic solids the governing constitutive law is

$$\frac{\partial \bar{\bar{\sigma}}}{\partial t} = \lambda \nabla \cdot \frac{\partial \bar{u}}{\partial t} \bar{\bar{\delta}} + \mu \left(\nabla \frac{\partial \bar{u}}{\partial t} + \left(\nabla \frac{\partial \bar{u}}{\partial t} \right)^2 \right) - \gamma \frac{\partial}{\partial \bar{\bar{\sigma}}} Q(\bar{\bar{\sigma}})$$

where γ depends non-linearly on $\bar{\bar{\sigma}}$ in the plastic state and vanishes for purely elastic deformations. The function Q is a non-linear function of $\bar{\bar{\sigma}}$.

This constituive law cannot be inserted into the law of motion, equation (5.4), as we did in the simpler linear case. $\bar{\bar{\sigma}}$ therefore cannot be eliminated from the system. Also the Navier-Stokes equations cannot be reduced to only one equation.

We are thus left with a coupled set of partial differential equations. Applying the Galerkin finite element method, results naturally in the mixed finite element method for the elasto-viscoplastic model. As mentioned introductorily, this fact is also a strong motivation for applying mixed finite element method initially on the simpler elasticity equation, for later to be able to extend the model to for instance elasto-viscoplastic models.

5.3 The interesting variable in elasticity

Above we stated that the central point in elasticity was to find the equilibrium position of the elastic body when subjected to an applied force. Hence the interesting unknown might be assumed to be the displacement $\bar{u}$.

However, in physical terms often the stress is of interest. Thus in many cases actually the stress $\bar{\bar{\sigma}}$ is the interesting variable. Using mixed method, this unknown is easily accessible as it is directly solved for. Thus numerical derivation is not necessary and does not introduce further numerical errors.

Not only the fact that we want to have the stress as one of the primary unknowns, motivate the use of mixed element method. Earlier we also mentioned that complex elasticity models, as visco-elasticity are often naturally on mixed form and are difficult to reduce to one equation and thus hard to solve with standard element methods. Moreover, as Section 5.6 will describe, the locking phenomenon may occur in standard element methods when materials are nearly incompressible and is avoided by introduction of mixed methods. There are thus several reason for applying mixed element method on elasticity.

5.4 Variational formulation of elasticity

In Chapter 3, we developed the finite element theory for elliptic problems and we stated the variational formulations for the scalar elliptic equation, the Poisson equation.

As mentioned in the analogy between the Poisson equation and the elasticity equation, we know that both are elliptic. Thus in the same manner, we can apply the variational formulation to the elliptic vector equations, the elasticity equation.

Repeating the elasticity equation:

$$\begin{aligned} -\bar{\text{div}}\ \bar{\bar{\sigma}} &= \bar{f} \quad \text{in } \Omega \\ \bar{u} &= \bar{g}_1 \quad \text{on } \Gamma_1 \\ \bar{\bar{\sigma}}(\bar{u})\bar{n} &= \bar{g}_2 \quad \text{on } \Gamma_2 \end{aligned} \tag{5.6}$$

where $\bar{n}$ is the unit outer normal vector and where stress tensor $\bar{\bar{\sigma}}$, $\bar{\bar{\epsilon}}(\bar{u})$ and $\bar{\bar{\delta}}$ are defined as above.

The variational formulation is as follows: Find $\bar{u} \in \bar{H}^1(\Omega)$ such that

$$\bar{u} = \bar{g}_1 \quad \text{on } \Gamma_1$$

and

$$a(\bar{u}, \bar{v}) = \int_\Omega \bar{f} \cdot \bar{v}\ d\Omega + \int_{\Gamma_2} \bar{g}_2 \cdot \bar{v}\ d\Gamma$$

for all $\bar{v} \in \bar{V}$, where the innerproduct is

$$a(\bar{u}, \bar{v}) = \int_\Omega \bar{\bar{\sigma}} : \text{grad } \bar{v} \; d\Omega = \int_\Omega \left(2\mu \bar{\bar{\epsilon}}(\bar{u}) + \lambda \text{tr } (\bar{\bar{\epsilon}}(\bar{u})) \bar{\bar{\delta}} \right) : \text{grad } \bar{v} \; d\Omega$$

and

$$\bar{V} := \left\{ \bar{v} \in \bar{H}^1(\Omega) \; : \; \bar{v}|_{\Gamma_1} = \bar{0} \right\}.$$

From this variational formulation we can, by inserting the approximation sum for the unknown $\bar{u}$, develop the linear system to be implemented and solved.

5.4.1 Existence and uniqueness of solution

In Chapter 3 we stated the Lax-Milgram theorem as the theorem to ensure existence and uniqueness of a variational formulation. The model equation in that case was a general second-order partial differential equation on a bounded domain. For the theorem to apply, we require the bilinear form, appearing when the model is put into variational formulation, is *continuous* and *coercive*.

We will not restate the theorem here. Instead we will restate that the elasticity equation, being an elliptic partial differential equation, is both continuous and coercive and this means that the theorem applies. We thus know there exists a solution of the elasticity equation.

If the boundary value problem (5.6) has only traction boundary contions ($\Gamma_1 = \emptyset$), that is being a pure traction problem, then the solution is unique only up to rigid motion. This is the same as the case of pure Neumann boundary conditions from Section 2.1.3. If $\Gamma_1 \neq \emptyset$ we can uniquely determine the solution.

5.4.2 Mixed variational formulation of elasticity

The mixed formulation of the elasticity equation reads

$$\begin{aligned} -\bar{\text{div}} \; \bar{\bar{\sigma}}(\bar{u}) &= \bar{f} \\ \bar{\bar{A}} \bar{\bar{\sigma}}(\bar{u}) &= \bar{\bar{\epsilon}}(\bar{u}) \end{aligned} \tag{5.7}$$

in the domain Ω and where the operator A is defined as

$$\bar{\bar{A}} \bar{\bar{\sigma}} = \frac{1}{2\mu} \bar{\bar{\sigma}} + \frac{-\lambda}{2\mu(2\mu + 2\lambda)} \text{tr } \bar{\bar{\sigma}} \bar{\bar{\delta}}.$$

This is in fact the inverse of the operator in Hooke's law.

In addition we have the usual displacement and traction boundary conditions as in equation (5.6).

Find $(\bar{\bar{\tau}}, \bar{u}) \in H(\text{div}, \Omega, \mathbb{S}) \times L^2(\Omega, \mathbb{R}^2)$ such that the boundary conditions are fulfilled and such that

$$\begin{aligned} (\bar{\bar{A}} \bar{\bar{\sigma}}, \bar{\bar{\tau}}) + (\bar{u}, \bar{\text{div}} \; \bar{\bar{\tau}}) &= \int_{\partial\Omega_1} \bar{g} \cdot \bar{\bar{\sigma}} \bar{n} \; d\Gamma + \int_{\partial\Omega_2} \bar{u} \cdot \bar{t} \; d\Gamma \\ -(\bar{\text{div}} \; \bar{\bar{\sigma}}, \bar{v}) &= (\bar{f}, \bar{v}) \end{aligned} \tag{5.8}$$

for all $\bar{v} \in \bar{V}$ as above and all $\bar{\bar{\tau}} \in \bar{\bar{\Sigma}} \subset H(\text{div}, \Omega, \mathbb{S})$.

The inner products are as follows:

$$
\begin{aligned}
(\bar{\bar{A}}\bar{\bar{\sigma}}, \bar{\bar{\tau}}) &= \int_\Omega \bar{\bar{A}}\bar{\bar{\sigma}} : \bar{\bar{\tau}}\ d\Omega = \int_\Omega \left(\frac{1}{2\mu}\bar{\bar{\sigma}} + \frac{-\lambda}{2\mu(2\mu+2\lambda)} \mathrm{tr}\ \bar{\bar{\sigma}}\bar{\bar{\delta}} \right) : \bar{\bar{\tau}}\ d\Omega \\
(\bar{u}, \bar{\mathrm{div}}\ \bar{\bar{\tau}}) &= \int_\Omega \bar{u} \cdot \bar{\mathrm{div}}\ \bar{\bar{\tau}}\ d\Omega \\
(\bar{\mathrm{div}}\ \bar{\bar{\sigma}}, \bar{v}) &= \int_\Omega \bar{\mathrm{div}}\ \bar{\bar{\sigma}} \cdot \bar{v}\ d\Omega \\
(\bar{f}, \bar{v}) &= \int_\Omega \bar{f} \cdot \bar{v}\ d\Omega.
\end{aligned}
$$

As for the standard variational formulation for elasticity, we have that the theory of mixed finite elements from Chapter 3 applies to the mixed elasticity. Thus we know there is a solution to the elasticity equation.

5.5 Linear system for implementation

Our purpose of this thesis is to implement the mixed formulation of elasticity. Diffpack is the software package we are using, which provide base classes suitable for solving problems with finite element methods. As will be described in the chapter on implementation, Chapter 6, we need to have expressions for the linear systems we get from the element method formulation. More specifically we need the expressions of the integrals making the element matrix.

We therefore consider the mixed variational formulation from the section above. By writing out the integrals and substituting the unknown variables with the approximating sums, we will arrive at the desired expressions.

Let us assume that we have a model problem with only boundary conditions on $\bar{u}$: $\bar{u}|_{\partial\Omega} = \bar{g}$. Thus in the mixed variational formulation (5.8), the boundary integral will appear differently, that is $\bar{\bar{\sigma}}(\bar{u})\bar{n}$ will not be replaced by $\bar{t}$, however the term $\bar{\bar{\sigma}}(\bar{u})$ will be replaced by the approximating sum.

Firstly we write out the mixed variational formulation with integrals with our boundary conditions:

$$
\begin{aligned}
\int_\Omega \bar{\bar{A}}\bar{\bar{\sigma}} : \bar{\bar{\tau}}_j\ \mathrm{d}\Omega + \int_\Omega \bar{u} \cdot \mathrm{div}\ \bar{\bar{\tau}}_j\ \mathrm{d}\Omega &= \int_{\partial\Omega} \bar{g} \cdot \bar{\bar{\tau}}_j \bar{n}\ \mathrm{d}\Gamma \quad j = 1, \ldots, n_\tau \\
-\int_\Omega \bar{\mathrm{div}}\ \bar{\bar{\sigma}} \cdot \bar{v}_l\ \mathrm{d}\Omega &= \int_\Omega \bar{f} \cdot \bar{v}_l\ \mathrm{d}\Omega \quad l = 1, \ldots, n_v
\end{aligned}
\tag{5.9}
$$

where $\bar{n}$ is the normal vector on the boundary $\partial\Omega$.

The approximations to the two unknowns are

$$
\bar{\bar{\sigma}} \approx \hat{\bar{\bar{\sigma}}} = \sum_{i=1}^{n_\sigma} \sigma_i \bar{\bar{\tau}}_i, \qquad \bar{u} \approx \hat{\bar{u}} = \sum_{k=1}^{n_u} u_k \bar{v}_k
$$

where σ_i and u_k are scalars and $\bar{\bar{\tau}}_i \in \bar{\bar{\Sigma}} \subset H(\mathrm{div}, \Omega, \mathbb{S})$ and $\bar{v}_k \in \bar{V} \subset L^2(\Omega, \mathbb{R}^2)$.

Inserting these approximations into system (5.9):

$$
\begin{aligned}
\int_\Omega (\bar{\bar{A}} \sum_{i=1}^{n_\sigma} \sigma_i \bar{\bar{\tau}}_i) : \bar{\bar{\tau}}_j\ \mathrm{d}\Omega + \int_\Omega \sum_{k=1}^{n_u} u_k \bar{v}_k \cdot \mathrm{div}\ \bar{\bar{\tau}}_j\ \mathrm{d}\Omega &= \int_{\partial\Omega} \bar{g} \cdot \bar{\bar{\tau}}_j \bar{n}\ \mathrm{d}\Gamma \quad j = 1, \ldots, n_\tau \\
-\int_\Omega (\bar{\mathrm{div}} \sum_{i=1}^{n_\sigma} \sigma_i \bar{\bar{\tau}}_i) \cdot \bar{v}_l\ \mathrm{d}\Omega &= \int_\Omega \bar{f} \cdot \bar{v}_l\ \mathrm{d}\Omega \quad l = 1, \ldots, n_v
\end{aligned}
$$

Expanding the operator $\bar{\bar{A}}$ into the above system:

$$\begin{aligned}\int_\Omega \frac{1}{2\mu}(\sum_{i=1}^{n_\sigma}\sigma_i\bar{\bar{\tau}}_i) : \bar{\bar{\tau}}_j + \frac{-\lambda}{4\mu(\mu+\lambda)}\mathrm{tr}(\sum_{i=1}^{n_\sigma}\sigma_i\bar{\bar{\tau}}_i)\bar{\bar{\delta}} \ : \bar{\bar{\tau}}_j \ \mathrm{d}\Omega \\ + \int_\Omega \sum_{k=1}^{n_u} u_k\bar{v}_k \cdot \bar{\mathrm{div}} \ \bar{\bar{\tau}}_j \ \mathrm{d}\Omega = \int_{\partial\Omega} \bar{g}\cdot\bar{\bar{\tau}}_j\bar{n} \ \mathrm{d}\Gamma \quad j = 1,\ldots,n_\tau \\ - \int_\Omega (\bar{\mathrm{div}} \ \sum_{i=1}^{n_\sigma}\sigma_i\bar{\bar{\tau}}_i)\cdot\bar{v}_l \ \mathrm{d}\Omega = \int_\Omega \bar{f}\cdot\bar{v}_l \ \mathrm{d}\Omega \quad l = 1,\ldots,n_v\end{aligned}$$

Rearranging by interchanging the summation and integration and isolating the unknowns give us:

$$\begin{aligned}\sum_{i=1}^{n_\sigma}\int_\Omega \frac{1}{2\mu}\bar{\bar{\tau}}_i : \bar{\bar{\tau}}_j \ \mathrm{d}\Omega \ \sigma_i + \sum_{i=1}^{n_\sigma}\int_\Omega \frac{-\lambda}{4\mu(\mu+\lambda)}\mathrm{tr}(\bar{\bar{\tau}}_i)\bar{\bar{\delta}} \ : \ \bar{\bar{\tau}}_j \ \mathrm{d}\Omega \ \sigma_i \\ + \sum_{k=1}^{n_u}\int_\Omega \bar{v}_k\cdot\bar{\mathrm{div}} \ \bar{\bar{\tau}}_j \ \mathrm{d}\Omega \ u_k = \int_\Gamma \bar{g}\cdot\bar{\bar{\tau}}_j\bar{n} \ \mathrm{d}\Gamma \quad j = 1,\ldots,n_\tau \\ \sum_{i=1}^{n_\sigma}\int_\Omega \bar{\mathrm{div}} \ \bar{\bar{\tau}}_i\cdot\bar{v}_l \ \mathrm{d}\Omega \ \sigma_i = -\int_\Omega \bar{f}\cdot\bar{v}_l \ \mathrm{d}\Omega \quad l = 1,\ldots,n_v\end{aligned}$$

Finally, we have arrived at the linear system

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix}\begin{bmatrix} \boldsymbol{\sigma} \\ \boldsymbol{u} \end{bmatrix} = \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{b} \end{bmatrix} \tag{5.10}$$

where the entries into the matrices and vectors are

$$\begin{aligned}\boldsymbol{A}_{i,j} &= \int_\Omega \frac{1}{2\mu}\bar{\bar{\tau}}_i : \bar{\bar{\tau}}_j \ \mathrm{d}\Omega + \int_\Omega \frac{-\lambda}{4\mu(\mu+\lambda)}\mathrm{tr}(\bar{\bar{\tau}}_i)\bar{\bar{\delta}} : \bar{\bar{\tau}}_j \ \mathrm{d}\Omega \\ \boldsymbol{B}^T_{k,j} &= \int_\Omega \bar{v}_k\cdot\bar{\mathrm{div}} \ \bar{\bar{\tau}}_j \ \mathrm{d}\Omega \\ \boldsymbol{B}_{i,l} &= \int_\Omega \bar{\mathrm{div}} \ \bar{\bar{\tau}}_i\cdot\bar{v}_l \ \mathrm{d}\Omega \\ \boldsymbol{g}_j &= \int_\Gamma \bar{g}\cdot\bar{\bar{\tau}}_j\bar{n} \ \mathrm{d}\Gamma \\ \boldsymbol{b}_l &= -\int_\Omega \bar{f}\cdot\bar{v}_l \ \mathrm{d}\Omega \\ \boldsymbol{\sigma} &= [\sigma_1,\sigma_2,\ldots,\sigma_{n_\tau}]^T \\ \boldsymbol{u} &= [u_1,u_2,\ldots,u_{n_v}]^T\end{aligned} \tag{5.11}$$

The definition of the matrix inner product between $\bar{\bar{\sigma}}$ and $\bar{\bar{\tau}}$ is defined as

$$\bar{\bar{\sigma}} : \bar{\bar{\tau}} = \sum_{i=1}^{2}\sum_{j=1}^{2}\sigma_{ij}\tau_{ij}.$$

The trace of a matrix $\bar{\bar{\tau}}$ is defined as

$$\mathrm{tr}(\bar{\bar{\tau}}) = \bar{\bar{\tau}} : \bar{\bar{\delta}} = \tau_{11} + \tau_{22}.$$

With the above expressions we are given all expressions in connection with elasticity which we will implement in Diffpack later.

The resulting linear system

We now have a linear system on the form as in equation (5.10). This is a symmetric indefinite linear system. There are known methods for solving such systems and in our implemenation later, we will solve the linear system by Gaussian elimination.

5.6 Locking phenomenon

As mentioned, finite element computation deteriorates as the Lamé constant λ approaches ∞. In materials which are nearly incompressible, for instance rubber, it takes a lot of energy to cause small changes in density. In such cases we have a large difference in the Lamé constants, $\lambda \gg \mu$.

The phenomenon is identified by inaccurate results and slow convergence which is not uniform as the grid size is refined. Moreover, the phenomenon is characterized by an underestimation of the displacement. Said in mechanical terms, the structural response is too stiff. From this comes the name locking, to reflect the graphical perception that the structure "locks" itself against deformation.

However, this locking phenomenon is a purely numerical phenomenon. Mathematically said, the locking phenomenon corresponds to poorly conditioned problems and the fact that the coercivity and continuity constants from the theory of finite element method are dependent on the Lamé constants. One way to overcome the locking phenomenon is to use a formulation involving a penalty term. Another way is to use mixed finite element computation. Here the coercivity and continuity constants for both the two involved bilinear forms are independent of the Lamé constants.

Chapter 6

Implementation of Mixed Elasticity in Diffpack

The implementation of mixed elasticity is done using Diffpack. Diffpack is an object-oriented software package, written in C++, for developing numerical software for solving partial differential equations.

Implementation in Diffpack consists mainly of creating subclasses of already existing Diffpack classes. The description of the mixed elasticity implementation uses the existing Diffpack structure as a starting point. This chapter therefore requires knowledge of this software package. For an introduction to Diffpack, the reader is referred to *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*, [11] and the World Wide Web documentation, [6].

The principles of implementation of FEM and MxFEM, described in Chapter 3, are already incorporated in Diffpack and they will therefore not be described in any more detail here. Instead the implementation done on top of the existing Diffpack structure will be described, since this is the relevant part of implementing mixed elasticity.

The implementation of mixed elasticity in Diffpack consists of two main parts: Implementing the new Arnold-Winther elements and implementing the simulator of the model using these elements. The following sections describe these two parts.

6.1 Implementation of a new mixed element in Diffpack

As described and computed in Chapter 4, we transform our abstract function space for the mixed elasticity elements into a set of basis functions. These basis functions are the essential information we want to implement in Diffpack.

The class `ElmDefs` contains the definitions of all elements in Diffpack. Adding a new element in Diffpack is achieved by extending Diffpack with a new subclass of `ElmDefs`.

The definition of elements is based on both parameters and functions in the new subclass of `ElmDefs`. Firstly, the parameters giving the number of basis function nodes (`nne_basis`), number of basis functions (`nbf`) and number of nodes on a side (`nbno_on_a_side`) are some of the necessary variables to set. Moreover, the coordinates of the basis functions in a reference element need to be given in the array `basis_coor`.

Secondly, the functions `basisFunc` and `dLocBasisFunc` giving respectively the basis functions and their derivatives need to be implemented. Geometry functions are defined in `geomtFunc`.

Such a definition of a subclass of `ElmDefs` returns only one scalar basis function for each basis function. Our mixed elasticity element pair consists of *vector* and *tensor* elements. For our `ElmDefs`-subclass we could therefore want the `basisFunc` to return the basis function as a vector or a tensor, both consisting of different function values in the different components.

To avoid the introduction of new functions returning the basis functions as vector or tensor, we consider each of the two new elements as a *set* of respectively two or three elements. Therefore the displacement element is implemented as two subclasses of `ElmDefs`, each subclass holding the set of one of the components of the basis functions. The stress element is similarly implemented as three subclasses. We only need three subclasses since the tensor is symmetric, thus the four components consist of only three different functions. Each subclass thus corresponds to one component of the original element basis function.

Usage of the elements in Diffpack

The usage of the basis functions implemented in subclasses of `ElmDefs` occurs through the class `FiniteElement` or in our case `MxFiniteElement`. These classes offer easy access to the elements' basis functions, derivatives and the Jacobi determinant. As we remember from Chapter 4, the Jacobi determinant is used in the computation of the element matrix for a reference system, when changing between global and local coordinates.

The class `MxFiniteElement` gets information not only from subclasses of `ElmDefs` but also from classes as `GridFE`, `BasisFuncGrid` and `FieldsFE` to provide necessary data structures such as the underlying grids and fields to hold the solutions.

The simulator class for a mixed system has the functions `integrandsMx` and `integrands4sideMx` to build element matrices which in turn are assembled to the global system. In this process, the `MxFiniteElement` object is vital to keep all the necessary information.

Naming the elements in Diffpack

Before we move on to the actual implementation of the elements, we give a reason for the naming of the Arnold-Winther elements in Diffpack.

The convention of naming elements in Diffpack and in `ElmDefs`, is that the elements start with "Elm". Then the shape of the element is indicated as a "B" for a box or "T" for a triangle. The following number denotes the number of geometry nodes, with the following "gn" for geometry nodes. The next number gives the number of basis functions, thus the subsequent "bn" (basis nodes). Finally, the space dimension is given as "1D", "2D" or "3D".

Not all elements follow this convention. The linear element we have described earlier, has the name `ElmT3n2D` and has thus a simpler name, than what this convention for other elements dictates.

As far as my experience goes, all the implemented elements in Diffpack have up to now had one basis function associated to each basis function node. Thus the number denoting the number of basis function nodes, has indirectly indicated the number of basis functions.

For both of the elements in the Arnold-Winther element pair, each basis function node has more than one basis function associated to it.

Following the above described naming convention, the name of the Arnold-Winther displacement element should be "`ElmT3gn3bn2D`" and for the Arnold-Winther stress element "`ElmT3gn7bn`". These names do not attempt to imply which component is represented; This information might easily be achieved by using a suffix to the name.

The suggested displacement element name, `ElmT3gn3bn2D` has already been taken by the Crouzeix-Raviart velocity element. We observe that only the information of number of geometry nodes and basis function nodes does not give a unique name. The missing number of basis functions is therefore necessary information to make the names unique.

We could have modified the method of naming the new mixed elements to consist of all information: number of geometry nodes, basis function nodes as well as basis functions, "`bf`". According to this principle, the names of the Arnold-Winther elements will be `ElmT3gn6bn6bf2D` and `ElmT3gn7bn24bf2D`.

However, these are long and too technical and are tedious to use. The paper *Overlapping Schwarz preconditioner for the mixed formulation of plane elasticity*, [14] concerning these elements and throughout this thesis, the name Arnold-Winther element has been used. A more informative name would thus be `ArnoldWintherDisp` and `ArnoldWintherStress` – which are the names used in this implementation.

To denote which component the subclasses of `ElmDefs` represents, suffixes will be used. For the first displacement component we add "1" to the name and for the second displacement component we add "2". For the stress elements we add the numbers "1", "2" and "4", skipping number "3" because of the symmetry of the stress tensor.

6.1.1 Arnold-Winther displacement element

We remember the displacement element in the Arnold-Winther pair to consist of six linear basis functions defined on three interior basis function nodes. The basis functions are not the same as the geometry function, thus the element is not isoparametric. We provide the above information in the element definition as

```
this->isoparametric_elm=false;
this->nne_basis         = 3;
this->nbf               = 6;
this->nbno_on_a_side    = 0;
this->order.fill(1);
this->value_node_rep = false;
this->corner_nodes= false;
elmtype = ARNOLDWINTHERDISP1;
```

All code extracts for the displacement element is taken from the element corresponding to the first component. All the information is the same in both subclasses, naturally only the basis functions are different.

`nbno_on_a_side` gives the number of basis nodes on a side, `order` refers to the order of the functions. The boolean variables `value_node_rep` and `corner_nodes` refer to respectively whether the nodal value can be used directly in visualization and whether the node is in the corner of the element.

Not all this information is relevant for the simplest use of the elements, that is to compute element matrices. For instance the variable `value_node_rep` is only used for plotting purposes.

The basis function nodes are given as

```
basis_coor.redim(nne_basis, nsd);
basis_coor(1,1) = 0.25; basis_coor(1,2) =  0.25;
basis_coor(2,1) = 0.50; basis_coor(2,2) =  0.25;
basis_coor(3,1) = 0.25; basis_coor(3,2) =  0.50;
```

As we have two degrees of freedom in each basis function node, this information is given in the following function:

```
int ArnoldWintherDisp1::getNoDof4scalarFunction (int node) const { return 2; }
```

Finally, the basis functions with the derivatives for the first component of the displacement element is implemented as

```
void ArnoldWintherDisp1:: basisFunc (Vec(real)& N, const Ptv(real)& loc_pt) const
{
  const real x=loc_pt(1);
  const real y=loc_pt(2);

  N(1) = 3 - 4*x - 4*y;
  N(2) = 0;
  N(3) = -1 + 4*x;
  N(4) = 0;
  N(5) = -1 + 4*y;
  N(6) = 0;
}

void ArnoldWintherDisp1:: dLocBasisFunc(Mat(real)& dNloc,const Ptv(real)& loc_pt)const
{
  const real x=loc_pt(1);
  const real y=loc_pt(2);

  dNloc(1,1) = -4;  dNloc(1,2) = -4;
  dNloc(2,1) = 0;   dNloc(2,2) = 0;
  dNloc(3,1) = 4;   dNloc(3,2) = 0;
  dNloc(4,1) = 0;   dNloc(4,2) = 0;
  dNloc(5,1) = 0;   dNloc(5,2) = 4;
  dNloc(6,1) = 0;   dNloc(6,2) = 0;
}
```

The numbering of the basis functions is related to the degrees of freedom as the earlier Figure 4.4 on page 32 shows. `dNloc(i,j)` in the function `dLocBasisFunc` is the derivative of the ith basis function in the jth variable.

As mentioned, we need to have two subclasses of `ElmDefs` for the displacement element. The other subclass `ArnoldWintherDisp2` has exactly the same information in the constructor with only the basis functions and the derivatives differing.

6.1.2 Arnold-Winther stress element

As the displacement element above, we provide the constructor in the subclass `ArnoldWintherStress` with the information defining the Arnold-Winther stress element. We remember this element has 24 basis functions of order up to three, defined at seven basis function nodes. Of these nodes, the vertices of the triangle and the interior node have three degrees of freedom, while each edge of the triangle has four.

For easier implementation, the degrees of freedom on each edge are associated with the midpoint of that edge. The numbering of the associated basis function to such a midnode follows the rule that the lowest number goes to the moment of degree 0, with the right hand side equal to $[1, 0]$, the next, goes to moment of degree 0, with the right hand side equal to $[0, 1]$. The next two are the moments of degree 1, with right hand side first $[1, 0]$ and for the last basis function $[0, 1]$.

The numbering for the nodal basis functions, follows the rule that the lowest number goes to the degree of freedom with a right hand side equal to $[1, 0; 0, 0]$, the next number in the sequence goes to the one with the associated right hand side equal to $[0, 1; 1, 0]$, while the highest number in a specific node goes to the basis function associated with the right hand side equal to $[0, 0; 0, 1]$.

Again, the element has not the same basis functions as geometry functions and is therefore not isoparametric.

The information provided in the constructors of all three subclasses which together make up the stress element, is:

```
this->isoparametric_elm=false;
this->nne_basis         = 7;
this->nbf               = 24;
this->nbno_on_a_side    = 3;
this->order.fill(3);
this->value_node_rep = false;
```

```
this->corner_nodes= true;
this->special_mapping=true;
elmtype = ARNOLDWINTHERSTRESS1;
```

Observe that the order of the element is set to three. In the later described `integrandsMx` function, two of such elements are multiplied before integrated. This makes it necessary to have a 6^{th} order numerical integration rule. The current version of Diffpack has only numerical integration rules up to order 5 implemented. A new rule was thus necessary and is described later in Section 6.3.1.

The variable `special_mapping` is set to signalize that we have a tensor basis function with scalar coefficient as the unknown.

The basis nodes are

```
basis_coor.redim(nne_basis, nsd);
basis_nodes_sides.redim(nsides, nbno_on_a_side);

basis_coor(1,1) = 0; basis_coor(1,2) = 0;
basis_coor(2,1) = 0.5; basis_coor(2,2) = 0;
basis_coor(3,1) = 1; basis_coor(3,2)= 0;
basis_coor(4,1) = 0.5; basis_coor(4,2) = 0.5;
basis_coor(5,1) = 0; basis_coor(5,2) =  1;
basis_coor(6,1) = 0; basis_coor(6,2) = 0.5;
basis_coor(7,1)=0.3333333333333333333;basis_coor(7,2)=0.3333333333333333333;
```

As we have basis function nodes on the edges we need to provide information about which side the nodes are on.

```
basis_nodes_sides(1,1) = 1;
basis_nodes_sides(1,2) = 2;
basis_nodes_sides(1,3) = 3;
basis_nodes_sides(2,1) = 3;
basis_nodes_sides(2,2) = 4;
basis_nodes_sides(2,3) = 5;
basis_nodes_sides(3,1) = 5;
basis_nodes_sides(3,2) = 6;
basis_nodes_sides(3,3) = 1;
```

`basis_node_sides(i,j)` denotes which basis node is the `j`$^{\text{th}}$ node on edge `i`. The corresponding figure is shown in Figure 6.1.

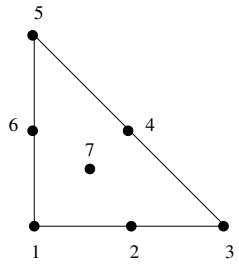


Figure 6.1: The numbering of the basis function nodes in the stress element.

The function defining the number of degrees of freedom in each node is for the stress element given as

```
int ArnoldWintherStress1::getNoDof4scalarFunction (int node) const {
  // three dofs in corner nodes and mid-node
  if(node==1 || node==3 || node ==5 || node==7)
    return 3;
  // four dofs along the egdes
  else if(node==2 || node==4 || node==6)
    return 4;
}
```

The basis functions and their derivatives are given on the form `N(i)` and `dNloc(i,j)` in the functions `basisFunc` and `dLocBasisFunc` in the usual manner. Due to the large

number of them, the code extract will not be listed.

The two other components of the stress element are given the names `ArnoldWinther-Stress2` and `ArnoldWintherStress4`.

6.2 Implementation of a mixed elasticity simulator in Diffpack

In *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*, [11], a typical finite element solver in Diffpack is described. In *Mixed Finite Elements* [12] Kent-André Mardal extends these ideas to the mixed case. The same ideas have to a large extent been used in the following implementation of the mixed elasticity simulator.

The typical finite element solver in Diffpack has a simulator class which inherits the standard finite element method algorithms from class `FEM`. This class contains the principles of FEM implementation mentioned in Chapter 3. In the mixed case, `MxFEM` is the appropriate base class. `FEM` and `MxFEM` are in turn derived from `SimCase`, which allows multiple loops and handling of numerical experiments from the menu or input-file system.

The data structures for FEM and MxFEM solvers are typically a grid (`GridFE`), finite element field(s) over the grid (`FieldFE`/`FieldsFE`), an interface to the linear system and linear solvers (`LinEqAdmFE`), a mapping between the field representation and the linear system representation of the unknowns (`DegFreeFE`). Finally problem-dependent functions and parameters complete the data structures of the simulator class.

Furthermore, the standard member functions in a simulator class are on page 334 in [11] listed to be

- `adm` for administering the menu system
- `define` for defining the items in the menu
- `scan` for reading input data and initializing the internal objects such as grid, fields and so on.
- `integrands(ElmMatVec&,FiniteElement&)` replaced by `integrandsMx(ElmMat-Vec&, MxFiniteElement&)` in the mixed case, for defining the integrals of the element-level equations
- optionally also `integrands4side/integrands4sideMx` for defining the integrands corresponding to boundary integrals in the weak formulation
- `fillEssBC` for marking essential boundary conditions
- `solveProblem` for the main program flow of the solver
- `resultReport` for writing results
- `saveResults` for storing results on file

The functions `adm`, `define` and `scan` will not be described in any detail. These functions take care of reading information about grid, elements and problem-specific parameters, and call appropriate functions to initialize the data structure and start out the flow of the solver. Below only specific parts associated with the mixed elasticity case will be described.

6.2.1 Data structures

As mentioned above, a MxFEM simulator needs underlying data structures to keep track of all information.

Firstly, we need an underlying grid, `Handle(GridFE) grid`, which partitions the domain into subdomains or cells. These subdomains are often referred to as elements in the code. This is an improper naming convention. A more proper name would of course be "subdomains" according to the formal definition of an element given in Chapter 4.

```
String gridfile = menu.get ("gridfile");
grid.rebind (new GridFE ());
readOrMakeGrid (*grid, gridfile);
```

The function `readOrMakeGrid` is an incorporated Diffpack function which as the name indicates, reads information about the grid from the input and makes the grid. `menu` is the object where all information from the input file or menu is stored before extracted.

`Handle(X)` is a smart pointer, pointing to an object of the class `X`. It has the same features as a standard C++-pointer and in addition offers garbage collection as memory is freed when a handle goes out of scope and no other handles point to the object.

Before describing the basis function grid and the fields, we consider the pair of elements again and the fields of solutions.

For the displacement field, we have two components. The solution is computed as a sum of the six basis functions. In front of the vector basis functions, there are scalars, the unknown degrees of freedom we are solving for. The scalars are common for both components of the basis function vector.

Similarly for the stress field, we have then four components. The solution is computed as a sum of basis function tensors, with a scalar in front. These scalars are again the unknown degrees of freedom in the linear system, and also common for all four basis functions making up the tensor.

We thus need to initialize six basis function grids, one for each basis function component. The basis function grids are created from the information from the elements and they are all initialized as shown by the example of `Handle(BasisFuncGrid) u1_grid`:

```
String u1_elm_tp = menu.get("u1 element");

u1_grid.rebind (new BasisFuncGrid (*grid));
u1_grid->setElmType (u1_grid->getElmType(), u1_elm_tp);
```

Similarly, we will need six `FieldFE` objects to store the values of the different components separately. In additions, we create two `FieldsFE` objects to collect the displacement and stress variable.

As an example, we initialize the stress fields from the separate component fields as

```
sigma_1.rebind (new FieldFE (*sigma1_grid, "sigma1_grid"));
sigma_2.rebind (new FieldFE (*sigma2_grid,sigma_1->values(),"sigma2_grid"));
sigma_3.rebind (new FieldFE (*sigma3_grid,sigma_1->values(),"sigma3_grid"));
sigma_4.rebind (new FieldFE (*sigma4_grid,sigma_1->values(),"sigma4_grid"));
```

The displacement field is initialized similarly. The composite stress field is initialized as

```
sigma.rebind(new FieldsFE(4, "sigma_fields"));
sigma->attach(*sigma_1, 1);
sigma->attach(*sigma_2, 2);
sigma->attach(*sigma_3, 3);
sigma->attach(*sigma_4, 4);
```

When these fields are to be connected to the unknowns represented by the `Handle(DegFreeFE) dof`-object, we first collect the fields. We also need to remember that we have *scalars* in front of the vector and tensor basis functions. We therefore cannot attach all fields to the `DegFreeFE`-object. Instead we deal with two "collectors":

```
fields_all.rebind (new FieldsFE (6,"collector"));// need 6 fields!
fields_all->attach(*sigma_1, 1);
fields_all->attach(*sigma_2, 2);
fields_all->attach(*sigma_3, 3);
fields_all->attach(*sigma_4, 4);
fields_all->attach(*u_1,5);
fields_all->attach(*u_2,6);

if(special_mapping){
  fields_numbering.rebind(new FieldsFE(2,"u-collector"));
  fields_numbering->attach(*sigma_1, 1);
  fields_numbering->attach(*u_1,2);
}
```

Note the boolean variable `special_mapping`. This variable signify in Diffpack that we have scalars in front of basis functions. Such a boolean variable is also set in the `MxMapping`-object. The `MxMapping` will be treated in a later section.

The collector of `fields_all` holds all components of the fields and is used to hold the solutions after computation. The collector `fields_numbering` is used in the linear system that is solved and is the collector which the `DegFreeFE` object is based on, which in turn holds the unknowns. To remind ourselves, the solution of the linear system gives us the coefficients in front of the functions in the approximating sum. The coefficients are also called the degrees of freedom, while the solution fields are the coefficients multiplied appropriately with the basis functions.

In case we would not have the "special mapping", the collection of all fields would look like:

```
else{    // not special mapping
    fields_numbering.rebind(*fields_all);
  }
```

However, this is never the case in our simulator because we are only using the Arnold-Winther elements which dictates "special mapping".

The `DegFreeFE`-object is initialized as

```
Handle(DegFreeFE)     dof;
dof.rebind(new DegFreeFE (*fields_numbering));
```

Note that the field `fields_numbering` is attached to the `DegFreeFE`-object.

Once having the `dof`-object initialized, we have the necessary data structure for the unknown scalars. We thus need to build the system of linear equations and to solve this system.

The functions in Subsection 6.2.2 describe the initialization of this system.

As for all finite element solvers, we have additional data structures concerning the linear system to be solved:

```
Vec(real)            linsol;  // solution of the linear system for (sigma,u)
Handle(LinEqAdmFE)   lineq;   // linear system, storage and solution
```

`linsol` contains the solution to the linear system and `lineq` administers the linear system. The use of these variables are not special for `MxElasticity`.

To be able to produce simulation files, we use `Handle(SaveSimRes) database` which offers a convenient interface to visualization of the simulation results. In our simulations the `database` object is used to dump fields and solutions to files for later visualization.

6.2.2 Problem-dependent functions

One of the main features of FEM implementation in Diffpack, is using the element matrix and element vector. The Diffpack system assembles these element matrices and vectors to

the global system. The user does not have to know the details of this process. However, the user has to implement the functions defining the element matrices and vectors.

The `integrands` or `integrandsMx` functions defines the element matrices and vectors. In the mixed case, the way the integrands are computed reflects the block structure of the linear system at the element level, as in the equation (3.18) and (5.10). There are three for-loops, one for each of the three blocks and one for-loop for the element vector, the right hand side of the equation. The integrands are found from the expressions in (5.11) and implemented in `integrandsMx` as follows. Assuming we have

```
real C = -lambda/(4*mu*(mu+lambda));
Handle(MxMapping) mxmapping = mfe.map;
```

then the upper left block matrix, $\boldsymbol{A}$ has entries implemented by

```
//upper left block matrix, the term mfe(Sigma).N(i)*mfe(Sigma).N(j)+tracepart
for (i = 1; i <= nSigmabf; i++) {
  for (j = 1; j <= nSigmabf; j++) {
    innerproduct=0;
    tracepart=0;
    for(int f = 1; f<=sigmafields; f++){
       innerproduct += mxmapping->N(f,i)*mxmapping->N(f,j);
    }
    innerproduct = (0.5/mu)*innerproduct;
    tracepart=(mxmapping->N(1,i)+mxmapping->N(4,i))
      *(mxmapping->N(1,j)+mxmapping->N(4,j));
    tracepart= C*tracepart;

    elmat.A(i,j) += (innerproduct+tracepart)*detJxW;
  }
}
```

Next the $\boldsymbol{B}$ and $\boldsymbol{B^T}$ have entries implemented by

```
// upper right block matrix, the term mfe(U).N(i)*mfe(Sigma).dN(j)
for (i = 1; i <= nSigmabf; i++)
  for (j = 1; j <= nUbf; j++){
    elmat.A(i,nSigmabf+j) +=
       (mxmapping->N(5,j)*mxmapping->divN(1,i)
       + mxmapping->N(6,j)*mxmapping->divN(2,i))*detJxW;
  }
// lower left block matrix
for (i = 1; i <=  nUbf; i++)
  for (j = 1; j <= nSigmabf; j++){
    elmat.A(nSigmabf+i,j) +=
       (mxmapping->divN(1,j)*mxmapping->N(5,i)
       + mxmapping->divN(2,j)*mxmapping->N(6,i))*detJxW;
  }
```

Finally the right hand side is set partially in `integradsMx` with the loop

```
// right hand side
for (i=1; i<= nUbf; i++) {
  elmat.b(nSigmabf+i) += ((mxmapping->N(5,i)*u_source1)
     +(mxmapping->N(6,i)*u_source2))*detJxW;
}
```

and in `integrands4sideMx` with the loop

```
 for (int i = 1; i <= nSigmabf; i++){
   elmat.b(i) +=
    (uting(1)*(mxmapping->N(1,i)*nv2(1)+mxmapping->N(2,i)*nv2(2)) +
     uting(2)*(mxmapping->N(3,i)*nv2(1)+mxmapping->N(4,i)*nv2(2)))
     *detSideJxW;
    }
```

Note that we in the last for-loop integrate over the boundary, using the variable `detSideJxW`. Here the Dirichlet boundary condition on u appears naturally, as expected in mixed formulations, in the boundary integral.

Secondly, note that the displacement basis functions are found as `mxmapping->N(5,i)` and `mxmapping->N(6,i)` and the stress basis functions are `mxmapping->N(1,i)` up to

`mxmapping->N(4,i)`. `mxmapping` of the class `MxMapping` takes care of calling the functions `N` and `dN` with the basis functions implemented in the elements. Instead of going directly through the functions through the `MxFiniteElement` by the call `mfe(1).N(i)` and thus using the basis functions directly, the `mxmapping` object applies the matrix Piola transform and necessary negative signs before the actual use of the functions. These transformations are highly necessary for the use of the functions. The `MxMapping` class will be described in a later subsection.

As no transformation is needed on the displacement basis functions, the call on the basis functions through the `MxMapping` object is in fact the same as going directly through the `MxFiniteElement` object.

Since we do not have boundary conditions involving derivatives, we do not have essential boundary conditions in the mixed formulation. Thus implementation of `fillEssBC` neither for the displacement nor the stress is necessary.

However, the implementation was still done for testing purposes. Since the way to go through all degrees of freedom is slightly different from the standard case where each basis function node only has one degree of freedom associated to it, the relevant parts from `fillEssBC4Sigma` will be listed as an example.

```
// for all elements
for (int e=1; e<= no_elms; e++) {
   nodes_in_elm = sigma_grid.getNoNodesInElm(e);

   //for all nodes in elements
   for (int node=1; node<=nodes_in_elm; node++) {
      glob_node = sigma_grid.nodel(e,node);

       // check if it is a boundary node
       if(sigma_grid.essBoNode(glob_node)){
          sigma_grid.getCoor(x,glob_node);
          sigmaanal->valuePt(tensor,x);  //analytic value in x

          dofsInNode = sigma_grid.getNoBasisFuncAtBasisNodeInElm(node,e);
          local_startdof = sigma_grid.node2dof(node);

          // for all dof's in one node
          for(int i=1; i<=dofsInNode; i++){
            loc_dof = local_startdof + (i-1);
            glob_dof = dof.loc2glob(e,loc_dof);

            //if a corner node, where we have three dofs
            if(node==1 || node== 3 || node==3){
              if(i==1)      value = tensor(1);
              else if(i==2) value = tensor(2);
              else if(i==3) value = tensor(4);
              dof.fillEssBC (glob_dof, value);
            }
         }
      }
   }
}
```

This code extract shows how we go through each element, and in each element we check all nodes if they are a boundary node. In case they are, we go through each degree of freedom in this node. We get the lowest local degree of freedom number in this node by the call `node2dof(node)`. This number is then just incremented by one for each consecutive degree of freedom in this node. This local degree of freedom is then mapped to the global degree of freedom by the call `dof.loc2glob(e,loc_dof)`.

This code excerpt only fills in the boundary node in the corners of an element, that is the degrees of freedom which are associated with nodal values. To fill in boundary nodes on the edges we would have to calculate the values of moments of degree 0 and 1 along the edges. This was not done, since it was not necessary for the simulator.

Finally, since we have discontinuous Lamé constants over our domain, the last problem dependent function is the function returning the appropriate Lamé constants according to where we are in the domain. The different values for the Lamé constants are saved in arrays, where the index refers to a subregion. When calling for instance from the function `integrandsMx`, the `getConstants` function returns the proper μ and λ values according to which integration point is evaluated. The function is the same as the one used later in the simulator for Galerkin FEM solution of elasticity.

6.2.3 Functors

A functor is a function represented in terms of a class. In Diffpack the class `FieldFunc` is such a functor which represents a function of space and time.

For analytical purposes, we simulate solutions which we know have an analytical solution. To be able to compare our numerical solutions obtained from the simulator with the analytical solutions, we need to have the analytical solutions to both fields as functors implemented as subclasses of `FieldFunc`.

An example of such a subclass will be:

```
#define ClassType sigmaAnal
#include <Handle.h>
#undef ClassType

class sigmaAnal : public FieldsFunc
{
public:
  MxElasticity* data;
  sigmaAnal (MxElasticity* data_);
  virtual int getNoFields() const { return 4; }
  virtual void valuePt (Ptv(NUMT)& tensor, const Ptv(real)& x, real t = DUMMY);
  virtual void vlhdivergenceFEM(Ptv(NUMT)& div, const FiniteElement& fe,real t=DUMMY);
};

sigmaAnal::sigmaAnal(MxElasticity* data_){ data=data_; }

void sigmaAnal:: valuePt (Ptv (NUMT) & tensor, const Ptv (real) & p, real t) {
  tensor.redim (4);
  real x = p(1); real y = p(2);

  real lambda = data->lambda;
  real mu = data->mu;

  tensor(1) = 2*mu*(1-2*x)+lambda*(2-2*x-2*y);
  tensor(2) = 2*mu*(1-x-y);
  tensor(3) = tensor(2);
  tensor(4) = 2*mu*(1-2*y)+lambda*(2-2*x-2*y);
}

void sigmaAnal:: vlhdivergenceFEM(Ptv(real) &div, const FiniteElement& fe, real t){
  real lambda = data->lambda;
  real mu = data->mu;

  div.redim(2);
  div(1)= -6*mu-2*lambda;
  div(2)= -6*mu-2*lambda;
}
```

For simulations of models to which we do not know the analytical solutions, these functors and all references to them should be removed. Also all error computations have to be removed.

When having originally natural boundary conditions which change to essential boundary conditions, the known function on the boundary might be implemented as such a functor.

6.2.4 MxMapping

The `MxMapping` class takes care of geometry mapping of certain mixed finite elements. The mapping is more specifically the transformation from coordinates in the reference element to the global coordinates in the given element.

Some mixed finite elements as the Arnold-Winther elements are defined to have continuous normal components at the midpoint of each edge. Standard mapping using the geometry basis functions from an element in the `ElmDef` hierarchy, does not preserve the normal components. Hence to preserve the normal vectors, `MxMapping` implements a geometric mapping for this purpose. This class contains therefore crucial information for using the Arnold-Winther elements. The implemented geometric mapping is the matrix Piola transform, described in Chapter 4. In addition the `MxMapping` class takes care of adding negative signs in front of the basis functions when necessary.

The following code extract shows the function `N` providing the basis functions for for instance the function `integrandsMx`

```
real MxMapping:: N (int field_no, int basis_no)
{
  if (use_special_mapping)
    {
      if (elm_no != mfe->getElmNo())
        refill();
      real scale;
      int sign;

      real buf = 0.0;
      int  pt =(*mfe)(1).getCurrentItgPt();
      const Mat(real)& jacobi = (*mfe)(1).getDataAtNumItgPt(pt)->
         JacobiMatrix();

        // displacement basis functions
        if ( field_no > 4) {    //4 istedet for nsd
          return (*mfe)(field_no).N(basis_no);
        }

        //stress basis functions
        else {

          real b1 = jacobi(1,1);
          real b2 = jacobi(1,2);
          real b3 = jacobi(2,1);
          real b4 = jacobi(2,2);

          // Piolatransform
          if(field_no==1){
            buf = b1*(*mfe)(1).N(basis_no)*b1 + b1*(*mfe)(2).N(basis_no)*b2 +
              b2*(*mfe)(3).N(basis_no)*b1 + b2*(*mfe)(4).N(basis_no)*b2;
          }
          if(field_no==2){
            buf = b1*(*mfe)(1).N(basis_no)*b3 + b1*(*mfe)(2).N(basis_no)*b4 +
              b2*(*mfe)(3).N(basis_no)*b3 + b2*(*mfe)(4).N(basis_no)*b4;
          }
          if(field_no==3){
            buf = b3*(*mfe)(1).N(basis_no)*b1 + b3*(*mfe)(2).N(basis_no)*b2 +
              b4*(*mfe)(3).N(basis_no)*b1 + b4*(*mfe)(4).N(basis_no)*b2;
          }
          if(field_no==4){
            buf = b3*(*mfe)(1).N(basis_no)*b3 + b3*(*mfe)(2).N(basis_no)*b4 +
              b4*(*mfe)(3).N(basis_no)*b3 + b4*(*mfe)(4).N(basis_no)*b4;
          }

          // "normal" triangle
          if(dir(1)==1){
            sign = 1;
          }
          // triangle rotated 180 degrees
          else if (dir(1)== -1){
            // moments of deg 1
            if(basis_no==6 || basis_no==7 || basis_no ==13 ||
               basis_no==14 || basis_no==20 || basis_no ==21)
              sign = -1;
            else
```

```
            sign = 1;
          }
        return 1/mfe->detJ()*sign*buf;
        }
  else {
    return (*mfe)(field_no).N(basis_no);
  }
}
```

In this code exerpt we see the Piola transform for the tensor basis functions as well as the negative sign in case of rotated triangles and moments of degree 1 basis functions.

In addition, a new function was added to `MxMapping`: A new function for getting the divergence of the tensor basis function, `real MxMapping::divN(int component, int basis_no)`. In the `integrandsMx` functions it can easily be called by `mxmapping->divN(1,i)` for the first component in the divergence of the i^{th} basis functions. This divergence function called in turn the function `MxMapping::dN`.

```
real MxMapping:: divN(int component, int basis_no){
  if ( elm_no != mfe->getElmNo() )
    refill();

  real buf = 0.0;
  int sign=1;
  real scale = 1;
  int  pt = (*mfe)(1).getCurrentItgPt();
  const Mat(real)& jacobi = (*mfe)(1).getDataAtNumItgPt(pt)->
    JacobiMatrix();

  real b1 = jacobi(1,1);
  real b2 = jacobi(1,2);
  real b3 = jacobi(2,1);
  real b4 = jacobi(2,2);

  // PIOLA TRANSFORM FOR DIV(SIGMA)
  if(component==1){
    buf= b1*((*mfe)(1).dN(basis_no,1) + (*mfe)(2).dN(basis_no,2))+
      b2*((*mfe)(3).dN(basis_no,1) + (*mfe)(4).dN(basis_no,2));
  }
  else if(component==2){
    buf= b3*((*mfe)(1).dN(basis_no,1) + (*mfe)(2).dN(basis_no,2))+
      b4*((*mfe)(3).dN(basis_no,1) + (*mfe)(4).dN(basis_no,2));
  }

  // "normally" oriented triangle
  if(dir(1)==1){
    sign = 1;
  }
  // triangle rotated 180 degrees
  else if(dir(1)==-1){
    // moments of degree 1
    if(basis_no==6 || basis_no==7 || basis_no ==13 ||
       basis_no==14 || basis_no==20 || basis_no ==21){
      sign = 1;
    }
    else{
       sign = -1;
      }
   }
  return 1/sqrt(mfe->detJ())*sign*buf;
}
```

In this code excerpt we see the Piola transform applied on the divergence of the stress. Next we see that on rotated triangles all basis functions apart from the ones associated with moment of degree 1 get a negative sign in front of the function.

6.3 Minor additional changes to existing Diffpack

Some of the additions and changes below done to Diffpack have been necessary for the solution of the elasticity model when using the new Arnold-Winther elements. For in-

stance without the numerical integration rule for 6^{th} order polynomials, the calculations of integrals in the element matrix would not be correct. Other changes as adding a function to be able to plot the solutions, were not strictly necessary. However, being able to plot solutions is quite convenient for testing and comparisons.

6.3.1 Numerical integration rule for 6^{th} order polynomials

In Diffpack, integration rules are collected in the class `ElmItgRules`. Adding a new integration rule for integrating over a triangle, means adding a new `if`-part in the function `rulesTriangle` in class `ElmItgRules`.

A numerical integration rule consists of a certain number of coordinates in the desired domain and a weight to each of these points. Then the numerical rule means evaluating the integrating function in the specified points and weighting with the given weight, before summing up. This is described in any textbook on numerical computation.

For our 6^{th} order integration rule, we used Gauss quadrature points and weights on a reference triangle.

6.3.2 Divergence function for use in `ErrorNorms`

To be able to compute the $H(div)$ error of the stress field, the functors were equipped with a new function returning the divergence of the stress tensor. This new function was then used in the error computations of the class `ErrorNorms`. In contrast to the previous divergence function, this new function returns a vector instead of a scalar.

6.3.3 Derivatives of the basis functions used in `ErrorNorms`

For the computation of the element matrix in the functions `integrandsMx` and `integrands4sideMx` we only used the standard form of the basis functions and the divergence of them. For error calculations we also need the derivaties of the basis functions. These derivatives are easily accessible from the element definitions as they are explicitly implemented. However, again we have to be careful with respect to the matrix Piola transform and possible negative signs when dealing with a rotated triangle.

In cases of 180-degrees rotated triangles, the derivatives of all basis functions apart from the ones associated with the moment of degree 0, have to have a negative sign when used in computations. This is implemented in the class `MxMapping` in addition to the matrix Piola transform.

6.3.4 Plotting

Usually the resulting files which Diffpack produces might easily be converted into plotable files through appropriate Diffpack scripts. *Plotmtv* is a plotting program often used with Diffpack. However, *Plotmtv* requires the values which are to be plotted, to be given in the nodal points of an underlying grid.

Our solutions consist of more values than just in the nodes of the underlying grids. For the displacement, the values are given in interior nodes in a triangulation. For the stress field, the values are in addition to be in the nodes, also given on the edges and in the interior of an element.

It is therefore necessary to adjust the representation of the solutions before plotting them. This means it is necessary to convert our solutions to the format *Plotmtv* requires.

The plots achieved are only used for visualizing the fields. All error computations should be done using original fields. The interpolation and other adjustments done

before plotting introduce some error, and it is therefore not recommended to use these fields in any further calculations.

Plot of the displacement solutions

The original displacement field has values in *interior* nodes of the triangles in the triangulation. Clement interpolation fixes this representation easily to fit the nodal value representation: For each node in the grid, the value of the midpoint of all neighbouring elements was computed and the average of these values was stored in the node, ready for plotting.

The Clement interpolation introduces some numerical errors. This might become visible in the visualization. However, an awareness of this, makes it still possible to use the plot for qualitative discusssion.

Plot of the stress solutions

For the stress fields, the original values in the corner nodes of the triangles in the mesh were stored in new fields. The other nodes were disregarded. The new fields were then ready to be plotted.

6.4 Standard FEM elasticity solver

For the purpose of comparing the MxFEM solution with a standard FEM solution, a simulator using FEM had to be implemented.

Already existing `Elasticity1`, described in *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*, [11], Chapter 5, models linear elasticity with thermal effects. The FEM simulator for elasticity is based on this code. The code of `Elasticity1` will not be described; The interested reader is referred to [11]. Only the modifications and adaptations will be briefly described in the following subsections.

Removal of the terms associated with the thermal effects

As for all simulators, one of the most problem-specific functions is the `integrands`-function. The expressions for the integrands defining the entries in the element matrices and element vectors, are derived from the variational formulation of elasticity.

In `Elasticity1` the model is a thermo-elastic problem. This model is similar to the model developed in the chapter on elasticity, Chapter 5. However, Hooke's law is combined with an empirical model for thermal strains. This means that the right hand side of the equation includes a thermal term. Simplified we have

$$\bar{\text{div}}\ \bar{\bar{\sigma}} = \bar{f} + \text{thermal term}.$$

This thermal term is removed from the definition of the entry into the element vector. The interesting part we end up with in `integrands` is

```
// add block matrix i to elmat.b:
   for (r = 1; r <= d; r++) {
     us = f(r)*fe.N(i);   // minus-tegn gir gøy plott
     ig = d*(i-1)+r;
     elmat.b(ig) += us*detJxW;
   }
```

Since we defined our test problems to have Dirichlet boundary condition, there was no necessity of implementing `integrands4side`.

Adding discontinuous Lamé constants

A similar strategy for finding the right Lamé constants depending on the location within the domain was added in the function `getConstants(Ptv(real)&ml,Ptv(real) point)`, returning both constants in a vector. Thus all references to these constants were done through this function:

```
getConstants(ml, point);
mu = ml(1);
lambda = ml(2);
```

Finding the derived stress quantity

The mixed method solves the system for both the displacement and the stress unknowns and both are of interest. Thus also in the case of FEM, the stress is of interest, not only the displacement that we actually solve for.

We extract the stress quantity by applying the operator A, a linear combination of the derivatives of the displacement field.

The original simulator `Elasticity1` works out a smooth stress measure. We want a stress field of four components. The `FieldsFEAtItgPt`-object has functionality for running through all elements and for each sampling point calling the virtual function `derivedQuantitiesAtItgPt` in the simulator class. The latter function was modified such that according to which field was to be computed, the function returns the correct component of the calculated stress. This is in contrast to the single field returned in `Elasticity1` containing the smoothed stress measure.

Adding functors

Finally functors or subclasses of `FieldsFunc` were implemented. These are similar to the functors we used in `MxElasticity`. This made it possible to compute error norms for the standard FEM solutions.

Chapter 7

Numerical Experiments

This chapter will first verify the mixed elasticity simulator by considering the errors of the numerical solutions produced by the simulator. Then differences in errors from FEM solutions and MxFEM solutions will be briefly commented, before we look at plots of solutions of elasticity problems with discontinuous Lamé constants. This last part will be given greatest focus, as it will answer the research question of this thesis.

All of the elasticity models and examples in this chapter are chosen from an academic point of view. No physical interpretation of the elasticity constants are considered. Thus there has been no attempt to simulate physical phenomena. However, in further work on mixed elasticity, such simulations are highly relevant.

7.1 Verification of the mixed elasticity simulator

To verify the implemented simulator we construct a problem with a known non-polynomial smooth analytical solution. We let the analytical solution be

$$\bar{u} = \begin{bmatrix} \cos y \\ \sin x \end{bmatrix}, \quad \bar{\bar{\sigma}} = \mu \begin{bmatrix} 0 & \cos x - \sin y \\ \cos x - \sin y & 0 \end{bmatrix}.$$

Hence the equation to solve on the domain Ω is

$$-\bar{\text{div}}\ \bar{\bar{\sigma}}(\bar{u}) = -\mu \begin{bmatrix} \cos y \\ \sin x \end{bmatrix} \tag{7.1}$$

with Dirichlet boundary conditions

$$\bar{u} = \begin{bmatrix} \cos y \\ \sin x \end{bmatrix} \quad \text{on } \partial\Omega. \tag{7.2}$$

We set the Lamé constants to be

$$\mu = 1, \lambda = 1.$$

We solve the equation on the domain $[0,1] \times [0,1]$. For the different simulations we refine the grid: Starting with mesh size equal to $h = \frac{1}{2}$ and refining each time with a factor $\frac{1}{2}$ to the final mesh size $h = \frac{1}{2^4}$. For these simulations we compute errors of the numerical solution in different norms.

Error \ h	2^{-1}	2^{-2}	2^{-3}	2^{-4}
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{H(\mathrm{div};\Omega)}$	7.21910e-03	1.80456e-03	4.51082e-04	1.12766e-04
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	5.84120e-04	7.78217e-05	9.99169e-06	1.26383e-06
$\|\|\mathrm{div}\ \bar{\bar{\sigma}} - \mathrm{div}\ \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	7.19543e-03	1.80288e-03	4.50971e-04	1.12758e-04
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{\infty}$	1.10843e-03	1.72269e-04	2.28717e-05	2.98307e-06
$\|\|\bar{u} - \bar{u}_h\|\|_{L^2(\Omega)}$	1.01382e-03	2.47301e-04	6.14860e-05	1.53516e-05
$\|\|\bar{u} - \bar{u}_h\|\|_{\infty}$	1.42973e-03	3.69931e-04	9.39036e-05	2.35465e-05

Table 7.1: Errors obtained for different grids with Arnold-Winther element.

Results

The errors are listed in Table 7.1. The leftmost column lists the type of norm of the error. The topmost row gives the mesh sizes h.

From this table we see, as expected, that the errors get smaller as the grid size gets smaller. Thus the numerical solutions converge to the analytical as h goes to 0.

Error estimates

By Theorem 5.1 in *Mixed finite elements for elasticity* by D. Arnold and R. Winther [1], we have the following error estimates for the Arnold-Winther elements:

$$\begin{aligned} ||\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h||_{L^2} &\leq ch^m||\bar{\bar{\sigma}}||_m, \qquad 1 \leq m \leq 3, \\ ||\bar{\mathrm{div}}\ \bar{\bar{\sigma}} - \bar{\mathrm{div}}\ \bar{\bar{\sigma}}||_{L^2} &\leq ch^m||\bar{\mathrm{div}}\ \bar{\bar{\sigma}}||_m, \quad 0 \leq m \leq 2, \\ ||\bar{u} - \bar{u}_h||_{L^2} &\leq ch^m||\bar{u}||_{m+1}, \quad 1 \leq m \leq 2. \end{aligned}$$

The errors in Table 7.1 agree with these error estimates for the Arnold-Winther elements. This fact verifies our simulator.

Rate of convergence

As mentioned above, from Table 7.1 we see that the numerical solutions converge to the analytical as h goes to 0. Considering the rate of convergence for the errors, enables us to say something about how fast the numerical solutions converge to the analytical solutions.

Table 7.2 lists the relationship between subsequent errors calculated from the formula

$$R(n) = \frac{\log(E_{2^{-n}}/E_{2^{-(n+1)}})}{\log(2)}.$$

Here we let $R(n)$ denote the rate of convergence calculated on basis of the error in mesh size 2^{-n} and $2^{-(n+1)}$.

From Table 7.2 we see that the Arnold-Winther element gives second order convergence for the displacement $\bar{u}$ in both L^2 and max norm. The convergence of the stress $\bar{\bar{\sigma}}$ is of third order or very close to third order measured with respectively L^2 and max norm. Measured with $H(\mathrm{div})$ norm the convergence is of second order. The L^2 of the divergence of the stress converges in second order as well.

Error \ h	$R(1)$	$R(2)$	$R(3)$
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{H(\mathrm{div};\Omega)}$	2.0002	2.0002	2.0001
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	2.9080	2.9614	2.9829
$\|\|\mathrm{div}\ \bar{\bar{\sigma}} - \mathrm{div}\ \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	1.9968	1.9992	1.9998
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{\infty}$	2.6858	2.9130	2.9387
$\|\|\bar{u} - \bar{u}_h\|\|_{L^2(\Omega)}$	2.0355	2.0079	2.0019
$\|\|\bar{u} - \bar{u}_h\|\|_{\infty}$	1.9504	1.9780	1.9957

Table 7.2: Rate of convergence of the different error norms for Arnold-Winther element.

7.2 Comparison with Galerkin finite element method

Next we solve the same equation (7.1) and (7.2) as above, with the same constants and mesh sizes. However, now we use Galerkin finite element method with linear elements. Hence only the displacement $\bar{u}$ is solved for, while the stress $\bar{\bar{\sigma}}$ is computed from the displacement.

Table 7.3 and Table 7.4 show respectively the same errors and convergence rate as computed for the same problem using mixed finite element method and Arnold-Winther elements.

Error \ h	2^{-1}	2^{-2}	2^{-3}	2^{-4}
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{H(\mathrm{div};\Omega)}$	5.37778e-01	3.24572e-01	2.13174e-01	1.45744e-01
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	1.72063e-01	6.02387e-02	2.09422e-02	7.31669e-03
$\|\|\mathrm{div}\ \bar{\bar{\sigma}} - \mathrm{div}\ \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	5.09509e-01	3.18933e-01	2.12143e-01	1.45560e-01
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{\infty}$	3.84037e-01	1.93267e-01	9.60968e-02	4.78075e-02
$\|\|\bar{u} - \bar{u}_h\|\|_{L^2(\Omega)}$	2.11255e-02	5.30393e-03	1.32740e-03	3.31938e-04
$\|\|\bar{u} - \bar{u}_h\|\|_{\infty}$	3.28900e-02	8.72714e-03	2.23038e-03	5.62735e-04

Table 7.3: Errors obtained for different grids with Galerkin FEM and linear elements.

From Table 7.3 we see that the errors are generally higher when using FEM than with MxFEM. From Table 7.4 we observe that the order of convergence for the displacement $\bar{u}$ is approximately the same using both methods. For the stress the order of convergence of the calculated stress is clearly less than when using the mixed finite element method. Especially the errors of the divergence of the stress converge very slowly.

The stress has thus, as expected, slower convergence with Galerkin finite element method than with mixed finite element method. The reason might be found in the fact that mixed finite element method solves directly for the stress, while standard Galerkin method only solves for the displacement and from this we must calculate the stress. However, for the solution of the displacement, both methods are satisfying in this example.

In Section 5.6 the locking phenomenon was mentioned. This phenomenon could cause standard element methods to fail, while mixed element methods would solve the problem. One way of observing locking is to consider the convergence of the finite element solution when λ approaches ∞. In case of locking the convergence is not uniform and the solution may be clearly wrong.

The above test example in this section is too simple to illustrate locking, since

Error \ h	$R(1)$	$R(2)$	$R(3)$
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{H(\mathrm{div};\Omega)}$	0.7285	0.6065	0.5486
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	1.5142	1.5243	1.5171
$\|\|\mathrm{div}\ \bar{\bar{\sigma}} - \mathrm{div}\ \bar{\bar{\sigma}}_h\|\|_{L^2(\Omega)}$	0.6759	0.5882	0.5434
$\|\|\bar{\bar{\sigma}} - \bar{\bar{\sigma}}_h\|\|_{\infty}$	0.9906	1.0080	1.0073
$\|\|\bar{u} - \bar{u}_h\|\|_{L^2(\Omega)}$	1.9939	1.9985	1.9996
$\|\|\bar{u} - \bar{u}_h\|\|_{\infty}$	1.9141	1.9682	1.9868

Table 7.4: Rate of convergence of the different error norms for FEM.

div u = 0. Hence no numerical experiments to show locking was done.

This comparison of FEM with MxFEM is mainly to show that MxFEM solves the stress variable more precisely than what FEM does. A more interesting comparison of these two methods would be for instance examples where locking occurs or where other differences occur. The next section with discontinuous Lamé constants is a case where interesting differences may occur.

7.3 Discontinuous Lamé constants

As outlined in the introduction, the research question sought to be investigated in this thesis is whether the mixed finite element method reveals more details in the simulation of elasticity with discontinuous elasticity constants than the standard Galerkin finite element method.

The physical interpretation of discontinuous elasticity constants is that the body subject to stress is composed of different materials with different elastic properties.

7.3.1 The elasticity model

Three slightly different elasticity models with discontinuities are simulated. For all of them we define the elasticity equation on the domain $\Omega = [0,1] \times [0,1]$ as

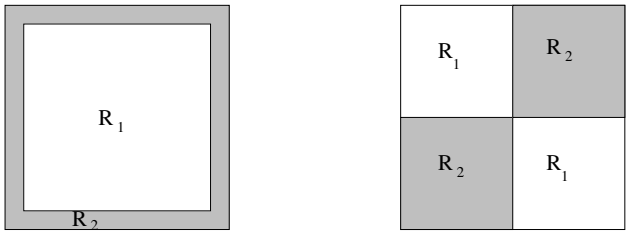
$$-\bar{\mathrm{div}}\ \bar{\bar{\sigma}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{on } \Omega$$

with boundary conditions

$$\bar{u} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{on } \partial\Omega.$$

Then we vary the area of discontinuity of the domain and the Lamé constants. We have two discontinuity fields as shown in Figure 7.1. The discontinuity field in Figure 7.1(a) shows a region with a border having different elastic properties than the middle region. The discontinuity field in Figure 7.1(b) is a region composed of two different materials, forming an intersection between four parts in the center.

These two discontinuity fields were used. In addition the Lamé constants were varied such that we got three examples to simulate. Table 7.5 summarizes the three example sets.



(a) Border. (b) Intersection.

Figure 7.1: Different discontinuity fields.

In Table 7.5 μ_1 and λ_1 are the elasticity parameters in the region R_1, while μ_2 and λ_2 are the elasticity parameters in the region R_2. All simulations are run with the mesh size $h = 16$, that is grid partition of [16,16]. In standard Galerkin finite element method linear elements are used to find the displacement. In mixed finite element method the Arnold-Winther elements are used.

These three examples are chosen among several test examples on the basis that they seem to show interesting differences and similarities. All test examples point in the same direction, but naturally not all of them could be included.

Observations

No numerical analysis of the two methods with respect to the discontinuity will be given, only qualitative observations of possible visual differences in the plots will be commented. These qualitative observations will focus especially around possible effects of discontinuities.

Since we do not know the analytical solutions to the problems, measuring or deciding which numerical method is better in these examples cannot be done. Thus no conclusion on which is actually better can be given. However, a tentative conclusion on what the differences in the plots may imply, will be given towards the end.

7.3.2 Plots of solutions

For each of the three simulations, the solutions of the stress $\bar{\bar{\sigma}}$ and the displacement $\bar{u}$ are plotted. For each example there are thus five figures with two plots in each. Three figures are for each of the three different components in the stress $\bar{\bar{\sigma}}$ and two for each of the components of the displacement $\bar{u}$. In each of these figures there are two plots, the leftmost with the solution using Galerkin finite element method and the rightmost with

	Type of field	μ_1	λ_1	μ_2	λ_2
Example 1:	Border	1	1	5	5
Example 2:	Intersection	1	1	5	5
Example 3:	Intersection	0.5	1	1	2

Table 7.5: The example sets.

the solution using mixed finite element method. This organization is meant for easier comparison of the plots of solutions using the different methods.

Figure 7.7 to Figure 7.9 are the $\bar{\bar{\sigma}}$ solutions to Example 1 from Table 7.5. Figure 7.10 and 7.11 are the $\bar{u}$ solutions to Example 1. Likewise Figure 7.12 to 7.16 are the five plots corresponding to Example 2, and Figure 7.17 to 7.21 corresponding to Example 3.

As explained in Chapter 6 on the implementation in Diffpack, to plot the displacement from the mixed finite element simulator, we have to use Clement interpolation. This interpolation as said, might introduce some errors in the plots. However, as we will see, the only differences between FEM and MxFEM appear in the stress variable. So the possible inaccuracy in the displacement plot is not relevant in these examples.

Before any comments on these plots are given, we will briefly visit the same equation with uniform Lamé constants, $\mu = 1$ and $\lambda = 1$ on the whole domain. This example is included to give a reference to where the differences do not occur and to motivate the introduction of discontinuous Lamé constants.

No discontinuities

Figure 7.2 to 7.6 on page 78 show the solutions of the equation (7.1) and (7.2) when the Lamé constants are uniform on the whole domain and take the values of $\mu = 1$ and $\lambda = 1$.

In these plots we do not see any visual differences between the solution using FEM or MxFEM. Apart from differences in the errors and rate of convergence from Section 7.2, there are no visual differences to investigate. Thus qualitative comparison of such differences is not relevant for this case. Instead moving over to the case of discontinuity, as the rest of this section is about, makes the case of comparing more interesting.

Comments to the plots

In general, the qualitative shape of the plots based on FEM calculations and on MxFEM calculations seems to have the same overall features, with minor differences. These visual differences will be pointed out below for each example.

The three components of the stress solution which we will consider, will be referred to as σ_1, σ_2 and σ_4 representing the tensor on the form

$$\bar{\bar{\sigma}} = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_2 & \sigma_4 \end{bmatrix}.$$

Example 1

For the stress component σ_1 in Example 1 (Figure 7.7 on page 79) there are thin lines of higher values along the boundary close to $y = 0$ and $y = 1$. The solution in MxFEM seems to reach a slightly higher value than in standard FEM. In addition, this highest value is very close to the boundary in the case of FEM, while in the case of MxFEM the highest value is slightly away from the boundary, closer to the edge between the two regions R_1 and R_2.

Moreover, in the case of MxFEM the line of highest values along the edges does not go all the way across the region. However, it seems to follow the border of region R_2 in Figure 7.1(a). In the case of FEM, we do not see this detail. Instead the line of higher values stretches from one edge of the outermost boundary to the other.

Finally, the MxFEM plot seems to drop off more sharply at the line where the Lamé constants change.

The common feature in both the FEM and MxFEM solution of σ_1, is that the solution is uniform in the interior region R_1.

Exactly the same comments as for σ_1 go for the stress component σ_4 (Figure 7.9 on page 79), only that the lines of high values now are at the borders close to $x = 0$ and $x = 1$.

For the stress component σ_2 (Figure 7.8 on page 79) the interesting behaviour is in all of the four corners in the plots. The values in the corners are significantly higher or lower than in the interior region. For the MxFEM solution, it seems that the regions of either higher or lower values are more sharply defined. This is especially visible in the corners $(0, 0)$ and $(1, 1)$ where we see that the point taking the lowest values coincides with the corner along the border between regions R_1 and R_2. In the plot of the FEM solution, the lowest value seems to be spread out a little bit between three points.

In addition, the interior region of the MxFEM plot seems to be formed like a square. This corresponds to the fact that the underlying discontinuity field has a square region R_1. While the FEM plot does not show this square region distinctly.

There are no visible differences in the plots of the displacement $\bar{u}$ (Figure 7.10 and 7.11 on page 80) solved by using Galerkin FEM or MxFEM.

Concluding anything about the differences in the stress plots in Example 1 is difficult when we do not know the true solution. Both of them do have the same overall shape. However, the MxFEM plots, especially in component σ_2, seem to reflect the underlying discontinuity field in a more correctly detailed way.

Example 2

At first sight, the plots in Example 2 of stress component σ_1 (Figure 7.12 on page 81) solved with FEM and MxFEM are quite different. This has mainly to do with the fact that in the FEM solution, there are quite high values in the regions R_2 (from Figure 7.1(b)) towards the edges $y = 0$ and $y = 1$. The same region in the MxFEM plot seems to take a more uniform value. In other words, the values in the regions R_2 seem to have more "noise" in the FEM solution than in the MxFEM solution.

Looking closer we see that the interval of values in region R_2 of the FEM and MxFEM plots are approximately the same, only displayed in different colours and thus giving the first impression of greater differences. Another common feature is that the values in the regions R_1 are quite uniform in both plots.

However, in the center of the plot, where the four regions of R_1 and R_2 intersect, we see in the MxFEM plot a distinct peak. Only slight signs of such a peak are visible in the FEM plot. The overall impression of the peak in the MxFEM plot is that it reflects the underlying discontinuity field in a clearer way than the FEM plot does, since it corresponds to the intersection of the regions.

On the other hand, from the general shape of both plots we clearly see the underlying discontinuity field.

The same comments go for the stress component σ_4 (Figure 7.14 on page 81).

Moving over to the plots of stress component σ_2 (Figure 7.13 on page 81) we see that both plots display a peak at the point corresponding to the intersection of the four regions. The peak in the FEM plot is wider in diameter, while the MxFEM plot has a thinner, but significantly higher peak. It seems like the MxFEM peak is more distinct when we take into consideration the discontinuity field of the domain, since the thinner

peak seems to more clearly show where the intersection of the regions is, while the spread out peak makes it less distinct.

Along the boundary of the domain, we see that where it changes from one region to another, there are regions of lower values. Again these areas are more precisely located in the MxFEM plot by being smaller in area, but taking lower values than the ones in the FEM plot. This feature in the FEM plot resembles the smearing effect observed in Chapter 2.

Again, there are no visible differences in the plots of the displacement $\bar{u}$ (Figure 7.15 and 7.16 on page 82) solved by using Galerkin FEM or MxFEM.

In this example, the differences, especially in Figure 7.13 in which tendencies of smearing effect might be seen, lead us to considering the MxFEM plots as more detailed and thus perhaps being more closer to the true solution.

Example 3

This example has the same underlying discontinuity field as Example 2. However, the Lamé constants are different. The general shapes of the plots are still similar to the shapes in Example 2.

For component σ_1 in Example 3 (Figure 7.17 on page 83) the plots are, as mentioned above, similar to the corresponding plots in Example 2 (Figure 7.12). The regions R_1 are in both examples quite uniform. Differences between FEM and MxFEM plots are found in the R_2 regions. These differences are difficult to describe, as they consists of only slightly different shapes in this R_2 region.

Also in this example the MxFEM plot has a more distinct peak in the intersection of the regions than the FEM plot. However, in Example 2 the peak is much more distinct than in Example 3. This is most likely due to change of Lamé constants.

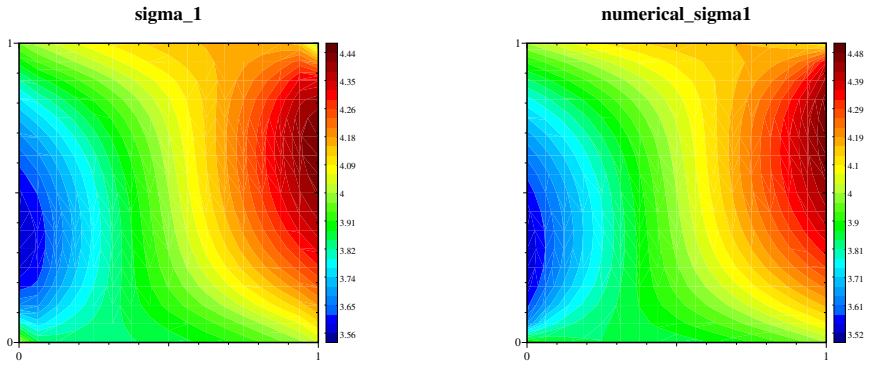
For the stress component σ_4 (Figure 7.19 on page 83) the same comments as for component σ_1 can be given.

Similarly as in Example 2, the component σ_2 (Figure 7.18 on page 83) has a distinct peak in the intersection of the regions in the MxFEM plot. While the peak in the FEM plot is both wider and not as high. Again MxFEM seems to be able to capture a special behaviour of the plot right in the intersection, while the FEM plot has not such a clear indication of the intersection. Thus again the FEM solution seems to be affected by the smearing effect.

Along the boundary where it changes from one region to the other, the "dip" of lower values is much more distinct in the MxFEM plot than in the FEM plot.

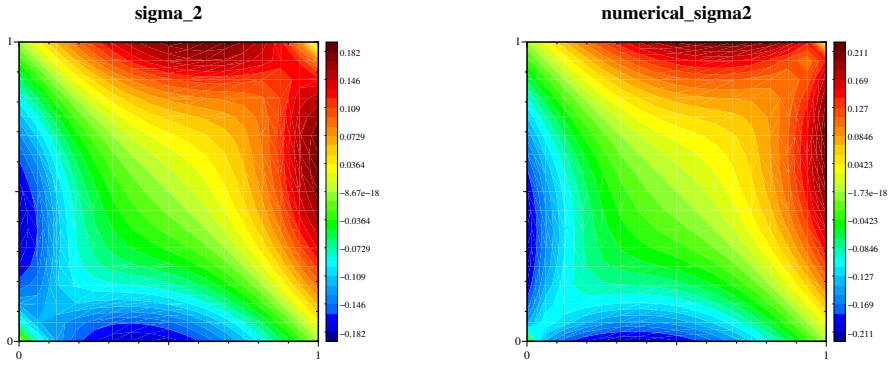
As in the two previous examples, there are no visible differences in the plots of the displacement $\bar{u}$ solved by using Galerkin FEM or MxFEM.

Summing up for this example leads us to the same conclusion as in Example 2. That is, the MxFEM seems to capture the stress solution in a more detailed way than the FEM and that the FEM solutions have slight signs of the smearing effect.



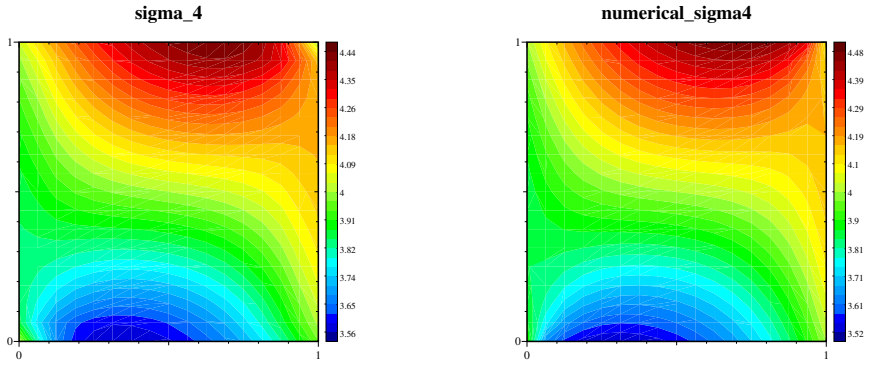
(a) FEM (b) MxFEM

Figure 7.2: No discontinuity: σ_1



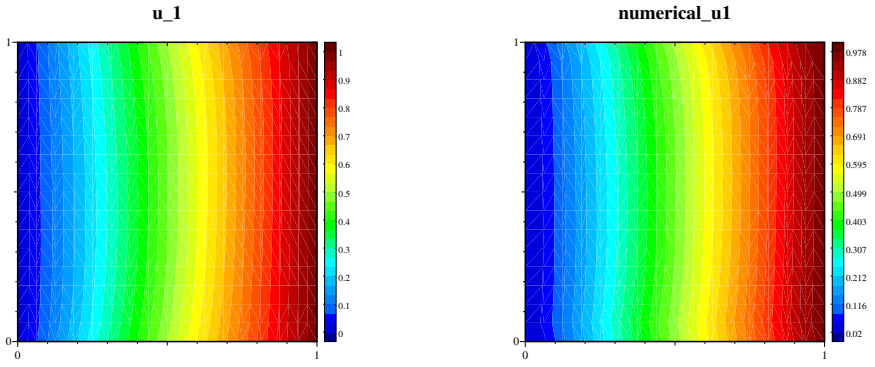
(a) FEM (b) MxFEM

Figure 7.3: No discontinuity: σ_2



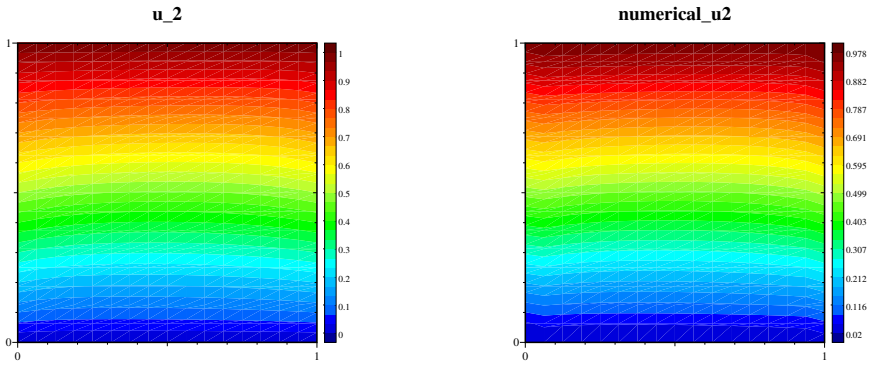
(a) FEM (b) MxFEM

Figure 7.4: No discontinuity: σ_4



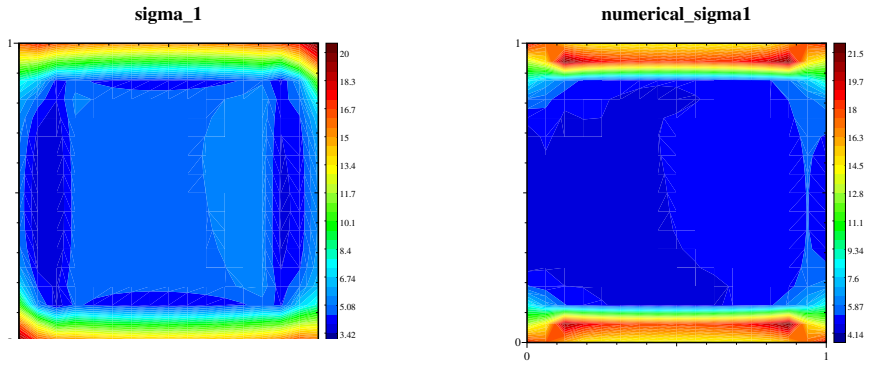
(a) FEM (b) MxFEM

Figure 7.5: No discontinuity: u_1



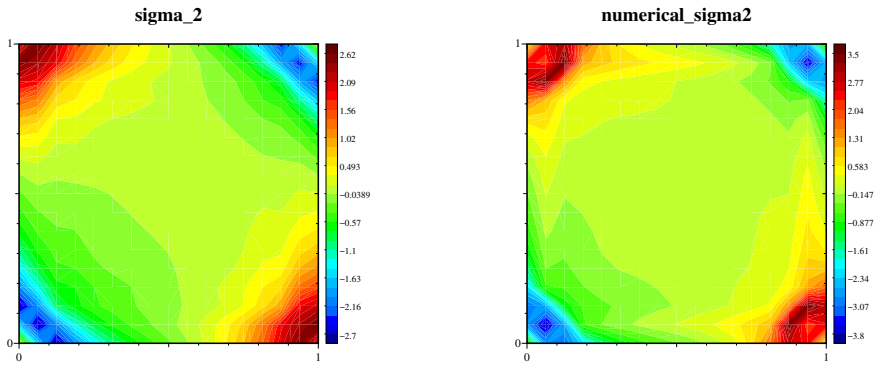
(a) FEM (b) MxFEM

Figure 7.6: No discontinuity: u_2



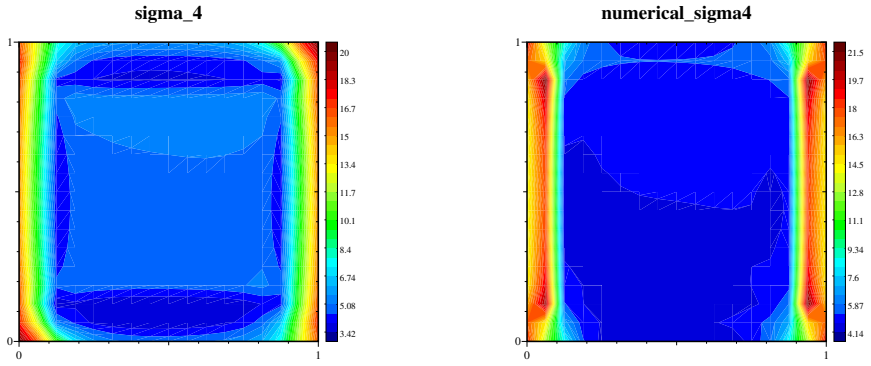
(a) FEM (b) MxFEM

Figure 7.7: Example 1: σ_1



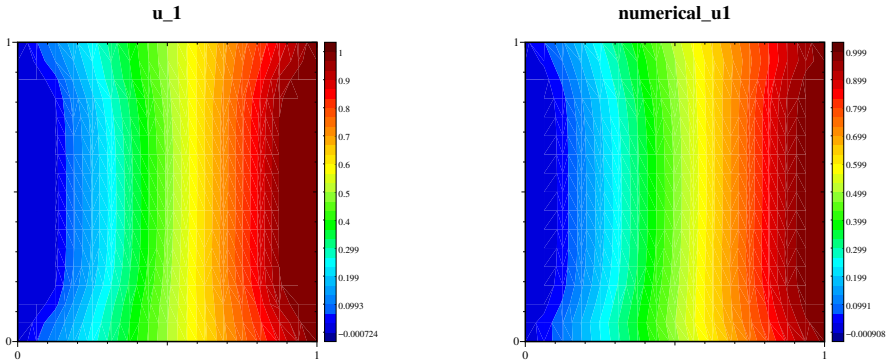
(a) FEM (b) MxFEM

Figure 7.8: Example 1: σ_2



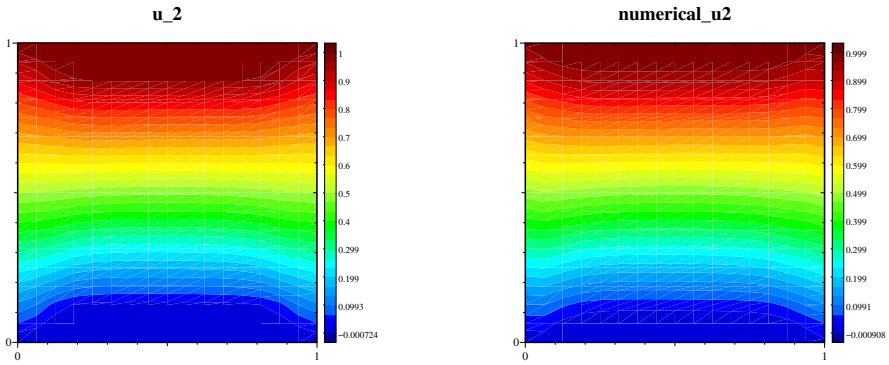
(a) FEM (b) MxFEM

Figure 7.9: Example 1: σ_4



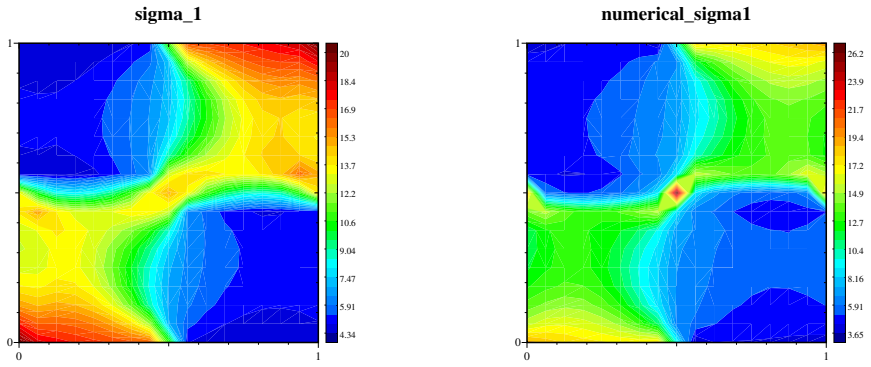
(a) FEM (b) MxFEM

Figure 7.10: Example 1: u_1



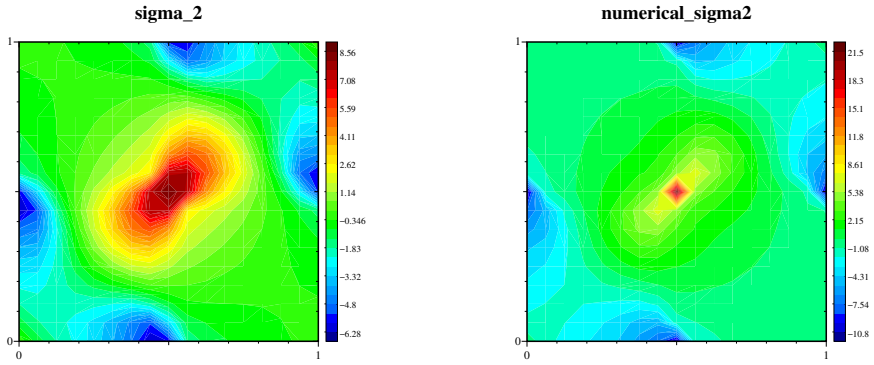
(a) FEM (b) MxFEM

Figure 7.11: Example 1: u_2



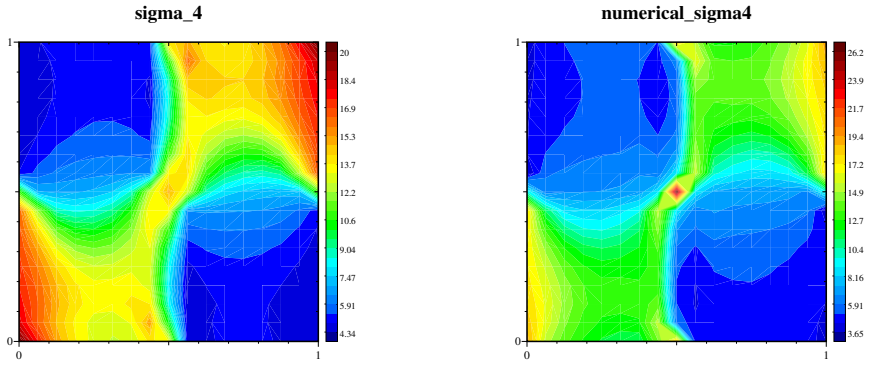
(a) FEM (b) MxFEM

Figure 7.12: Example 2: σ_1



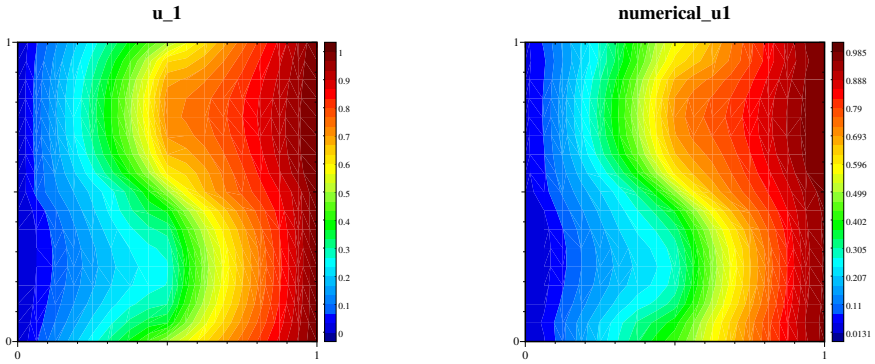
(a) FEM (b) MxFEM

Figure 7.13: Example 2: σ_2



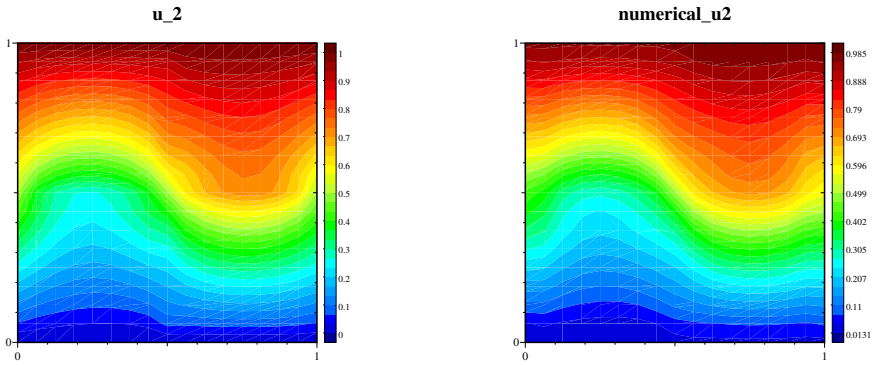
(a) FEM (b) MxFEM

Figure 7.14: Example 2: σ_4



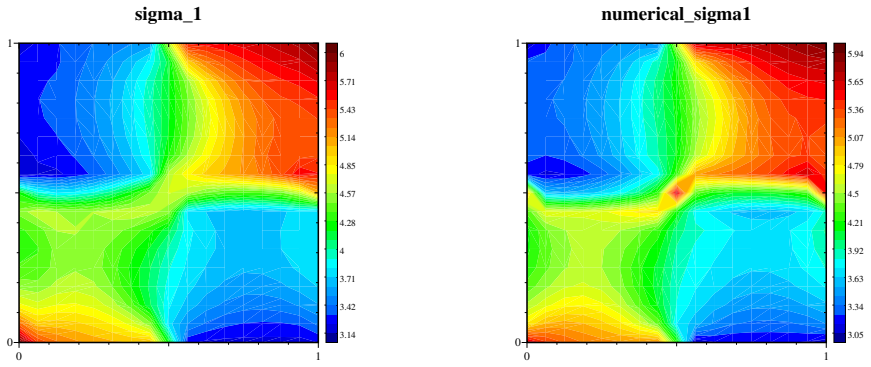
(a) FEM (b) MxFEM

Figure 7.15: Example 2: u_1



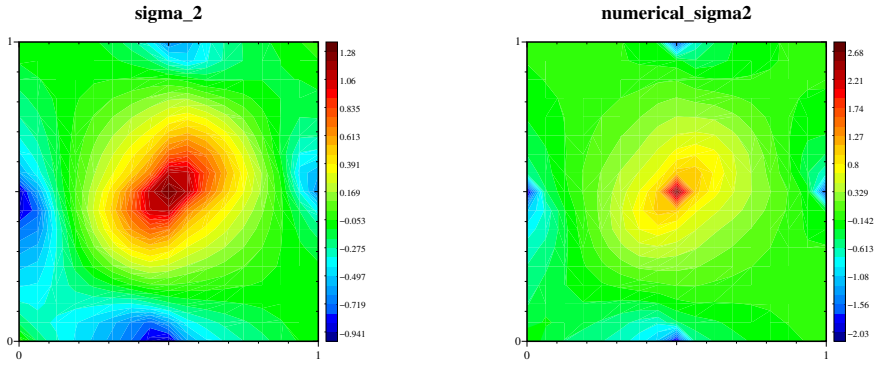
(a) FEM (b) MxFEM

Figure 7.16: Example 2: u_2



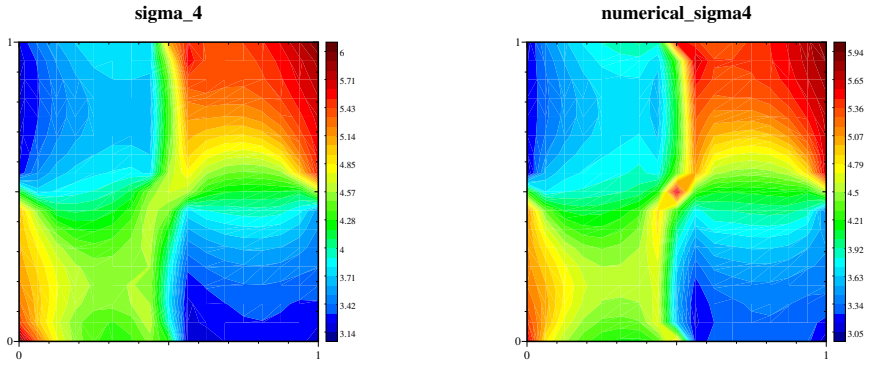
(a) FEM (b) MxFEM

Figure 7.17: Example 3: σ_1



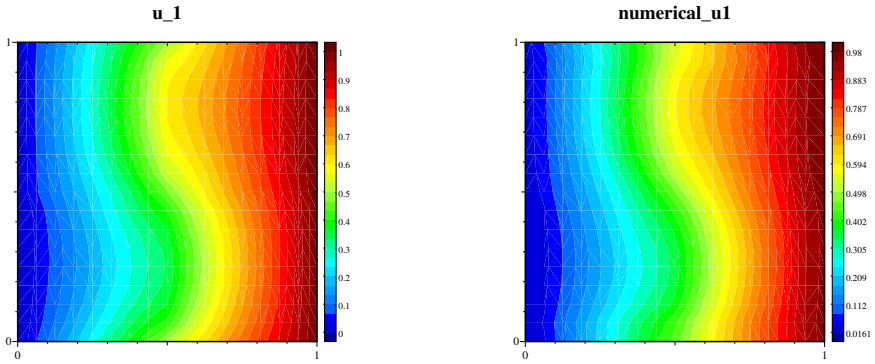
(a) FEM (b) MxFEM

Figure 7.18: Example 3: σ_2



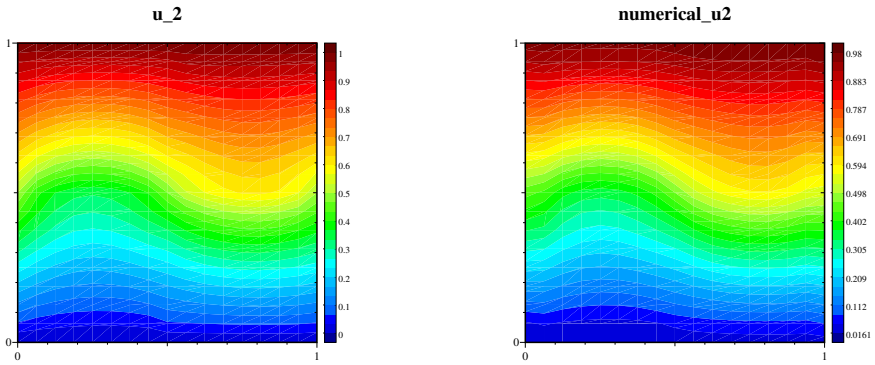
(a) FEM (b) MxFEM

Figure 7.19: Example 3: σ_4



(a) FEM (b) MxFEM

Figure 7.20: Example 3: u_1



(a) FEM (b) MxFEM

Figure 7.21: Example 3: u_2

7.3.3 Tentative conclusion

As pointed out in the section above, for all three examples no visual differences are found in the simulation of the displacement using FEM or MxFEM. On the other side, simulating the stress using MxFEM and calculating the stress from the displacement in FEM, give different details in the stress plots.

There is no obvious answer to which of the two numerical methods show the stress solution in the most correct and detailed way by only looking at the plots. However, using the knowledge of differences between FEM and MxFEM for other elliptic partial differential equations, might help us pointing out a tentative conclusion about the differences in MxFEM and FEM solutions in elasticity. As we saw in the introductory example in Chapter 2, MxFEM displays more details in the solutions of fluid flow in discontinuous permeability fields than Galerkin FEM. Especially the details around the discontinuity are simulated better by MxFEM, while FEM displays smearing effect. This is also generally accepted in this research field, shown in for instance *On Locally Conservative Numerical Methods for Elliptic Problems; Applications to Reservoir Simulations*, [9].

In the three examples in this chapter, it seems that the MxFEM is able to keep the effects of discontinuity in a more detailed way and that FEM has tendencies of the smearing effect. In other words, for instance the peak located at the intersection of the four regions with discontinuous elasticity constants is more distinct using MxFEM. While FEM has a tendency of smearing out such features.

Another point is that MxFEM solves the stress directly, while in FEM it is calculated from the solved displacement. This latter method might introduce further numerical errors to the stress. This also leads us to assuming that the MxFEM solution is the better. Also Section 7.2 and the larger errors using FEM implies that MxFEM solves the stress better than FEM. However, remember that in these simulations there were no discontinuities.

On the other side, the background from fluid flow with discontinuous permeability might as well as guiding us in seeing results, bias our view on the results and make us look for particular details. Moreover, no facts from the physics of elasticity with discontinuous Lamé constants are considered. Thus this section is called only “tentative conclusion”.

One reason for the possibility that the introductory example might bias the results, is the interesting point that if we compare the variables solved for in the fluid flow equation and in elasticity, we can consider the pressure variable in fluid flow to correspond to the displacement in elasticity, and velocity in fluid flow to correspond to the stress. The introductory example displays more details in the MxFEM solution of the pressure, while we in our elasticity example do not see differences in details in the plot of the corresponding displacement. Thus the analogue mentioned in the introductory example may perhaps be nothing more than a motivation for our research questions and using it as a guide in seeing results in elasticity may not be well entitled.

Moreover, the three examples presented in this chapter do by no means cover all possible variation within the topic of discontinuous Lamé constants.

At most the strongest conclusion we can reach is that there do exist differences in the simulations of stress using FEM and MxFEM with discontinuous Lamé constants. With the background of results in fluid flow and general knowledge that stress is solved for directly in MxFEM, leads us to assume that the MxFEM solutions are more detailed. Thus the answer to the research question ends up being that mixed finite element method does reveal more details in the plot of the solutions than the Galerkin finite element method.

Chapter 8

Concluding Remarks and Further Work

8.1 Concluding remarks

In this thesis the mixed Arnold-Winther elements with strongly imposed symmetry have been calculated such that they are expressed explicitly. Next the elements have been implemented in the software package Diffpack. Although calculating the basis functions of the elements is quite straightforward, the use of especially the Arnold-Winther stress element has to be careful with respect to Piola transform and orientation of edges and normal vectors. In Diffpack these special considerations are implemented in the class `MxMapping`, while the explicit basis functions are implemented in subclasses of `ElmDefs`. A simulator using these Arnold-Winther elements for the elasticity equation (with Dirichlet boundary conditions) was also implemented as a subclass of `MxFEM`. Thus the implementation follows the standard Diffpack conventions.

In Chapter 7 on numerical experiments the errors of the numerical solutions and the rate of convergence of the numerical solutions are calculated. We see that the implemented Arnold-Winther elements produce errors which agree with the error estimates. This verifies our simulator.

Moreover, this chapter shows plots of three examples with discontinuous elasticity constants. For each of the examples there are two plots for each of the components in the solutions. One plot is the solution using standard Galerkin finite element method with linear elements and one plot is the solution using mixed finite element method with Arnold-Winther elements. We observe that there are visual differences in the details of the stress variable using the two methods. The displacement does not have such visual differences.

From the introductory example we know that Galerkin method results in a smearing effect in the pressure in fluid flow when there is a discontinuous permeability field, as opposed to the mixed finite element method which solves the pressure well. Using this knowledge in addition to considering the details we see in the plots of the elasticity examples, leads us to a tentative conclusion of more correct details using MxFEM in elasticity as well. This tentative conclusion is also based on the fact that MxFEM is generally more precise for solutions of the stress than Galerkin FEM. This conclusion thus answers the research question of this thesis, that is that mixed finite element method

does indeed reveal more details in the simulation of elasticity models when discontinuous Lamé constants are involved, than what the standard Galerkin finite element method does.

8.2 Further work

Implementing the Arnold-Winther elements in Diffpack opens the way to several simulations and further numerical experiments. Only a few experiments have been done in this thesis. Naturally there are further experiments and work to be done using these Arnold-Winther elements.

Firstly, more extensive simulations of the discontinuous case of elasticity could be done. More underlying discontinuity fields could be used, in addition to the variation of Lamé constants could be chosen from physical examples. Especially an investigation into the physical phenomenon of having discontinuous elasticity constants would help to interpret the observed details and differences in plots with discontinuities. The physical facts would be valuable additions for the simulations of models without analytical solutions. Simulations of physical examples would also add more relevant and interesting examples, than purely academic ones.

Next, showing numerically the locking phenomenon would also be a natural task in confirming that MxFEM is preferred over FEM in certain cases. Having implemented MxFEM would in fact help us avoid the problem of locking and simulations of nearly incompressible materials would be possible.

Concerning the implementation issue, implementation of higher order Arnold-Winther elements would also be a natural extension to the current implementation. Likewise, an implementation of more general elasticity models, with both kinds of boundary conditions is also a natural extension. Also implementation of more complex elasticity models, like elasto-viscoplastic models which must be solved with mixed finite element method, that is being naturally on mixed form, would be an interesting extension.

These suggestions are the most obvious and natural extensions to the work that has been done in this thesis. Apart from these suggestion there is a vast amount of research to be done concerning mixed elements and elasticity, among others are extensions of the Arnold-Winther elements to three dimensions and preconditioners for the mixed formulations.

Bibliography

[1] Douglas N. Arnold and Ragnar Winther. Mixed finite elements for elasticity. *Numerische Mathematik*, 2002.

[2] Dietrich Braess. *Finite elements: Theory, fast solvers, and applications in solid mechanics*. Cambridge University Press, 1997.

[3] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Number 15 in Texts in Applied Mathematics. Springer-Verlag, 1994.

[4] P. G. Ciarlet. *The Finite Element Method for Elliptic Problems*. North-Holland, Amsterdam, New York, Oxford, 1978.

[5] P. G. Ciarlet and J. L. Lions, editors. *Handbook of Numerical Analysis - Numerical Methods for Solids (Part 1)*, volume III. North-Holland, 1994.

[6] World Wide Web documentation. *Diffpack Kernel and Toolboxes Documentation Version 4.0.00*. URL: http://www.nobjects.com.

[7] Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, 2002.

[8] Robert C. Kirby. FIAT: A new paradigm for computing finite element basis functions. *ACM Trans. Math. Software*, 2004.

[9] Runhild Aae Klausen. *On Locally Conservative Numerical Methods for Elliptic Problems; Application to Reservoir Simulation*. PhD thesis, University of Oslo, 2003.

[10] Hans Petter Langtangen. When the Physics Gets Mixed. September 13, 2003.

[11] Hans Petter Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*. Springer, 2003.

[12] Kent-André Mardal. *Software and Numerical Methods for the Incompressible Navier-Stokes Equations*. PhD thesis, University of Oslo, 2003.

[13] Yanqiu Wang. Implementation of the Arnold-Winther Element. Unpublished documentation.

[14] Yanqiu Wang. Overlapping Schwarz preconditioner for the mixed formulation of plane elasticity. *Applied Numerical Mathematics*, 2003.

[15] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method*, volume 1: The Basis. Butterworth Heinemann, 5th edition, 2000.