Kevin Ayrton Pendl, BSc

Development of an Implicit Cell-Centred Finite Volume Solver for Nonlinear Solid Mechanics

Master's Thesis

to achieve the university degree of
Diplom–Ingenieur

Master's degree programme: Mechanical Engineering and Business Economics
Computational Engineering and Mechatronics

submitted to

Graz University of Technology
Faculty of Mechanical Engineering and Economic Sciences

Supervisor

Univ.-Prof. Dipl.-Math.techn. Dr.-Ing. Thomas Hochrainer
Institute of Strength of Materials

Second Supervisor

Dipl.-Ing. Benedikt Weger, BSc
Institute of Strength of Materials

Graz, June 2020

With your mind power,
your determination,
your instinct,
and the experience as well,
you can fly very high.

AYRTON SENNA

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.
Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.
The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Graz, ...
date (signature)

Abstract

The Finite Element (FE) method is a common tool used for calculations in solid mechanics. In contrast, the Finite Volume (FV) method is primarily used for fluid flow problems. However, in the last few decades about 400 scientific articles on FV approaches for solid mechanics have been published, including books, theses and papers. The regular FV approach for solids is oriented on the fluid flow implementations and incorporates a deferred correction approach, especially in the context of nonorthogonal grids. So-called cross diffusion terms are formed within the diffusion term, which are incorporated into the right-hand side of the system of equations and are explicitly treated in the segregated solution process.

In contrast, it is common practice in solid mechanics with FE discretisation to pursue an implicit solution process, where the governing equations are assembled in a block matrix. In large deformation theory, a linearisation process using directional derivatives in combination with the *Newton-Raphson* method may be performed. The equations of large deformation solid mechanics are generally nonlinear due to the nonlinear kinematics and material law, hence the linearisation. This approach is described in detail in the literature.

The purpose of this thesis is to develop a FV solution algorithm analogous to the FE approach for nonlinear solid mechanics. All terms are treated implicitly and lead to a set of linear algebraic equations as a result of the linearisation and discretisation process. After the theoretical introduction, including nonlinear solid mechanics and the FV method, the implicit solution algorithm is developed step by step. A generic example, which is discussed in detail, contributes to the understanding of the introduced equations. Eventually, simulation results of the implemented MATLAB© code are presented. Results from the arising code called SOOFVAM are compared to the in-house solver SOOFEAM, a MATLAB© FE implementation for nonlinear solid mechanics. In addition, the results are compared to the open-source FV solvers OpenFOAM© and foam-extend, which comprise linear and nonlinear solid mechanics solution algorithms, respectively.

Kurzfassung

Während die Finite Elemente (FE) Methode ein wesentliches Werkzeug zur Berechnung in der Festkörpermechanik darstellt, wird die Finite Volumen (FV) Methode hauptsächlich für Strömungsprobleme herangezogen. In den letzten Jahrzehnten wurden jedoch in etwa 400 wissenschaftliche Beiträge zur Anwendung von FV Methoden in der Festkörpermechanik veröffentlicht. Der allgemeine Ansatz orientiert sich dabei an der Verwendung der Methode für Strömungsprobleme, die im Besonderen im Zusammenhang mit nicht-orthogonalen Gittern einen *Ansatz der verzögerten Korrektur* (deferred correction approach) vorsieht. Dabei werden im Diffusionsterm Terme sogenannter *Kreuzdiffusion* (cross diffusion) gebildet, die explizit in den *segregierten Lösungsansatz* (segregated solution process) einfließen.

Im Gegensatz dazu ist es in der Festkörpermechanik in Kombination mit der FE Diskretisierung gang und gäbe, einen impliziten Lösungsansatz zu verfolgen. Die resultierenden linearen Gleichungen werden dabei in eine Blockmatrix assembliert und implizit gelöst. Im Fall der Festkörpermechanik für große Deformationen ist es dabei üblich, die infolge der nichtlinearen Kinematikzusammenhänge und des im allgemeinen nichtlinearen Materialgesetzes nichtlinearen Lösungsgleichungen zunächst zu linearisieren und dann zu diskretisieren. Dabei wird sich die Methode von Richtungsableitungen und das *Newton*-Verfahren (auch bekannt als *Newton-Raphson*-Methode) zunutze gemacht. Für FE Methoden ist dieser Vorgang ausführich in der Literatur beschrieben.

Ziel dieser Arbeit ist es, einen zur FE Methode für nichtlineare Festkörpermechanik analogen Lösungsalgorithmus für die FV Methode zu entwickeln. Hierbei sollen ebenfalls sämtliche Terme implizit in das zu lösende Gleichungssystem eingearbeitet werden. Nachdem die benötigten theoretischen Inhalte zur nichtlinearen Festkörpermechanik und der FV Methode geklärt sind, wird der implizite Lösungsalgorithmus sukzessive entwickelt. Anhand eines generischen Beispiels, welches als Gedankenexperiment Schritt für Schritt erläutert wird, soll der Algorithmus zugänglicher gemacht werden. Abschließend werden Simulationsergebnisse des in MATLAB© implementierten Codes präsentiert. Die Ergebnisse des aus der Implementierung resultierenden Programms SOOFVAM werden dabei mit der hauseigenen Software SOOFEAM, welches ein MATLAB© FE Programm für nichtlineare Festkörpermechanik darstellt, sowie mit den beiden Open-Source Paketen OpenFOAM© und foam-extend, die jeweils FV Solver für lineare und nichtlineare Festkörpermechanik darstellen, verglichen.

Danksagung

Ich möchte mich an dieser Stelle bei all jenen bedanken, die zum Gelingen dieser Arbeit, sei es durch ihre fachliche oder persönliche Unterstützung, beigetragen haben.

Zunächst möchte ich Herrn Univ.-Prof. Dipl.-Math.techn. Dr.-Ing. Thomas Hochrainer dafür danken, mir dieses durchaus fordernde, aber interessante Thema anvertraut zu haben. Dabei freut es mich, dass ich die Möglichkeit hatte, neue Erkenntnisse für diese zum Teil neue Thematik liefern zu können. Weiters möchte ich mich hiermit auch für das rege Interesse am Fortschritt der Arbeit bedanken.

Für alle anregenden Diskussionen, fachlichen Inputs und der im Allgemeinen ausgezeichneten Betreuung möchte ich mich im Besonderen bei Dipl.-Ing. Benedikt Weger, BSc, bedanken. Ohne dein Zutun wäre mir das Bewältigen der Aufgaben sicher um einiges schwerer gefallen.

Vor allem gilt mein Dank aber meiner Familie, die mich bisher auf meiner akademischen Laufbahn, sowie bei allen anderen Entscheidungen, vollends unterstützt hat. Als mein persönlicher Motivator und Antrieb dient für mich weiterhin das Versprechen, das ich meinem verstorbenen Vater gemacht habe, nämlich „stets so zielstrebig zu bleiben, wie bisher“.

Nichtsdestoweniger möchte ich mich bei allen bedanken, die sich die Mühe gemacht haben, diese Arbeit Korrektur zu lesen. Auch für jegliche Form von Kritik, sei sie von inhaltlicher oder stilistischer Natur, die in diesem Zusammenhang an mich herangetragen wurde, bin ich dankbar. Ohne euch wäre diese Arbeit nicht in dieser Fassung zustande gekommen.

List of Abbreviations

BC	boundary condition
CFD	Computational Fluid Dynamics
CM	Continuum Mechanics
FE	Finite Element
FEM	Finite Element Method
FV	Finite Volume
FVM	Finite Volume Method
LHS	left-hand side
LS	least-squares
NR	*Newton-Raphson*
NRM	*Newton-Raphson* method
OpenFOAM	Open Source Field Operation and Manipulation
PDE	Partial Differential Equation
RHS	right-hand side
SIMPLE	Semi-Implicit Method for Pressure Linked Equations
SOOFEAM	Software for Object-Oriented Finite Element Analysis in Matlab
SOOFVAM	Software for Object-Oriented Finite Volume Analysis in Matlab
TC	test case
UML	Unified Modeling Language

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Continuum Mechanics (CM) is a branch of mechanics that, inter alia, studies the motion of solids and takes into account the hypothesis of continuous media [14, 19]. The hypothesis is an idealisation in which physical phenomena are explained without a detailed knowledge of the internal microstructure of a body [14, 23].

When a body $\mathcal{B}$ is considered as a continuous medium (as continuum), it is seen as an open subset of the *Euclidian* space [19, 38] with a continuous distribution of matter in space and time [23]. The body $\mathcal{B}$ is imagined as being a composition of a set of continuum (or material) particles, represented by the point $p \in \mathcal{B}$ and its position vector $\underline{x}_p$ shown in Figure 1.1. Properties of the body, e.g. density, temperature, velocity or displacement, are defined as field quantities and assigned to this material points.

If the internal structure of the body $\mathcal{B}$ is neglected, the large number of particles is replaced with a few quantities. This so-called macroscopic system can be a good approximation from a physical point of view. Although the predictions based on a macroscopic system are not exact, they are precise enough for the design of machine elements in mechanical engineering [23].

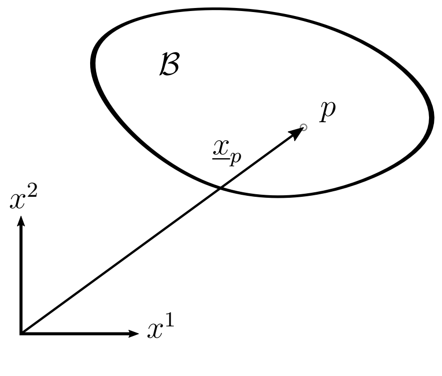


Figure 1.1.: Continuous Body

In solid-based continuum mechanics, one is interested in the calculation of a so-called deformed configuration, which is the result of loading the body $\mathcal{B}$ with external forces. If this deformed condition is an equilibrium state, one needs to solve the Partial Differential Equation (PDE) of the equilibrium of translational momentum. It reads (with neglected body forces):

$$\underline{\nabla} \cdot \underline{\underline{\sigma}} = \underline{0}\,, \;\; \underline{x} \in \mathcal{B}\,. \tag{1.1}$$

The derivation of Equation (1.1) is discussed in detail in Chapter 2. In general, the PDE in Equation (1.1) is solved numerically since there are only few analytical solutions for the complex problems dealt with today.

The common approach in solid mechanics is a discretisation process using the Finite Element Method (FEM). The domain of consideration is divided into so-called *finite elements*, and the discretisation process is based on the concept of virtual work, which is not discussed further here. In the case of nonlinear solid mechanics, one needs to solve Equation (1.1) iteratively since it is in general nonlinear. A regular approach is the linearisation of the nonlinear equation using a directional derivative approach and solving the resulting set of linear equations with the *Newton-Raphson* (NR) method. Another discretisation concept is the Finite Difference Method (FDM), where PDEs are approximated with finite differences. However, the Finite Volume (FV) method works in a different manner.

But similar to the mentioned methods, the FV method transforms PDEs into a set of linear algebraic equations. In general, the following steps need to be performed in order to achieve this [21, 35, 40]:

(I) the computational domain is subdivided into finite cells (with finite volumes),
(II) the PDE, as result of balance principles, is integrated over the volume/surface of the cell,
(III) surface and volume integrals are approximated as discrete sums using an integration quadrature,
(IV) the variation of variables within the cell is approximated and surface values are related to their respective cell values, and
(V) the system of linear algebraic equations is assembled and solved.

This common approach can be divided into two main steps—the discretisation of the computational domain and the discretisation of the governing equations. Step (I), which corresponds to the discretisation of the domain, is introduced in Section 3.1, whereas the further steps, that correspond to the latter, are discussed in Section 3.2 and Section 3.3.

Like other numerical methods, the first approaches of the Finite Volume Method (FVM) were introduced in the last century. First attempts were done by McDonald, MacCormack and Paullay, and just like then the FVM is mainly used for fluid flow problems today [40]. It is the most widely used method for Computational Fluid Dynamics (CFD) [21].

Since then, two main approaches of FVMs have been developed: cell-centred and vertex-centred approaches. They mainly differ in the discretisation of the computational domain. In the cell-centred approach, a cell is the result of the subdivision of the computational domain. Each cell consists of a centroid, where the field variables are defined and stored. In contrast, in a vertex-centred approach the cell is formed after the subdivision of the computational domain. The cells are the result of connecting the centroids of cells that have a vertex in common. Therefore, they form a so-called dual mesh [4, 10, 21, 35]. In a vertex-centred approach, the field variables are assigned to the vertices.

On the left of Figure 1.2, a cell-centred discretisation approach is shown, while on the right an approach of a vertex-centred grid may be identified. Since a cell-centred approach is used in this thesis, vertex-centred grids are not further taken into account.

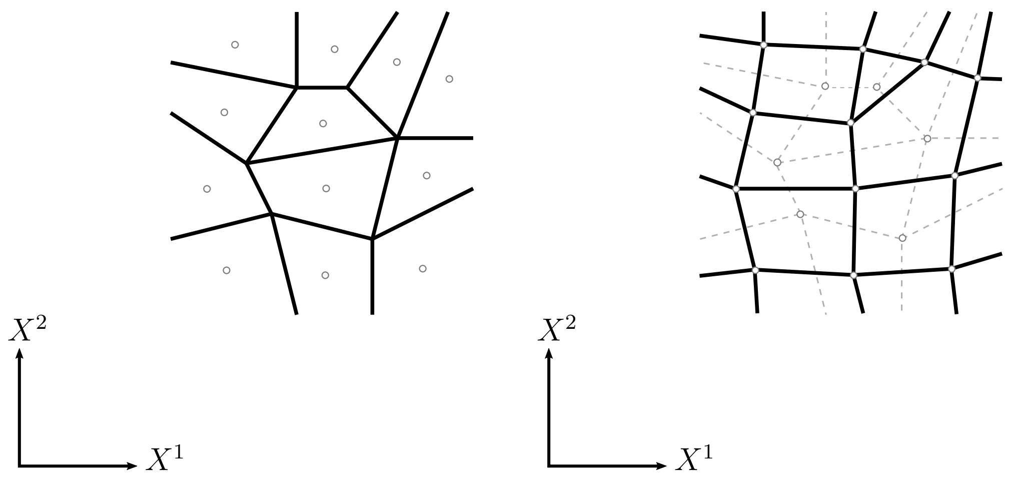


Figure 1.2.: Cell-Centred Grid (left) and Vertex-Centred Grid (right)

The applications of FV approaches are not limited to fluid mechanics. Different FV solvers for solid mechanics have been developed, and in the last decades about 400 scientific papers including books, theses and conference proceedings have been published. *Cardiff and Demirdžić* give an overview of this work in their not yet peer-reviewed paper [7]. The following provides a brief overview of state of the art applications of FV solvers in CM.

1.1. State of the Art of FV Approaches for Solids

The FV approaches for solids are generally based on the implementations of fluid flow problems. Therefore, the governing equations are adapted to the common FV method. Some of these approaches published over the last 15 years are presented below.

In the years 2005 and 2006, *Bijelonja et al.* proposed a FV formulation both for incompressible small and large deformation problems. They both base on a SIMPLE[1] algorithm for displacement-pressure coupling, with the hydrostatic pressure as an additional dependent variable. They assumed an incompressible hyperelastic material, i.e. a *Mooney-Rivlin* material model, as constitutive law for finite deformations. The methods were formulated using the integral conservation equations governing momentum balance in total *Lagrangian* formulation [2, 3].

[1]Semi Implicit Method for Pressure Linked Equations

An approach of a FV module called "`MulPhys-FV`" integrated into the numerical program "`MulPhys`" [36] was carried out by *Limache and Idelsohn* in 2007. They formulated a FV code for nonlinear solid mechanics including simulations for large deformations, rotations and displacements. Using a vertex-centred approach, they assumed a hyperelastic *St. Venant-Kirchhoff* material model [31].

In 2007, *Tukovic et al.* developed a selfcontained FVM Fluid-Structure Interaction solver to simulate dynamic interactions between an incompressible *Newtonian* fluid and an elastic solid with the assumption of large structural deformation. Their simulations were realised with the open-source toolbox OpenFOAM©[2]. Their cell-centred approach comprised a *St. Venant-Kirchhoff* material model and the solver was validated on the vibration of a cantilevered elastic beam used to simulate the dynamic response of an axial turbine blade [45].

A matrix-free vertex-centred FV approach was discussed by *Suliman et al.* in 2014. In their simulations, they investigated linear elastic structures undergoing bending. The results were compared with the traditional isoparametric Finite Element (FE) method. They determined similar disadvantages as in the linear FE formulation for their FV formulation. In order to circumvent the issues, an enhanced FV approach was proposed, which provides an alternative for the analysis of bending problems [42].

In 2017, *Haider* introduced a new computational framework for the numerical analysis of large strain explicit solid dynamic problems. His proposed algorithm is entitled TOUCH[3]. The basis of his work was the development of a new set of first order conservation laws, where the unknown variables are linear momentum and the extended set of geometric strain measures. In his dissertation isothermal hyperelastic and elastoplastic constitutive models were applied. The proposed framework was implemented from scratch in OpenFOAM©. Finally, the code was compared against other in-house numerical methods, including the FEM and meshless methods [20].

Cardiff et al. proposed a new cell-centred FV approach for simulations of metal forming processes in 2017. The governing equations were formulated in updated *Lagrangian* formulation, and a hyperelastoplastic constitutive relation was assumed. The results of the simulation were compared to a number of benchmark test cases, where good agreement with available analytical and FE solutions was achieved [8].

1.2. Comparison of the FVM and the FEM

Although some FV approaches have been developed for solid mechanics, it is still common practice to use the FE method. As it was mentioned in the previous section, the FEM may be used as benchmark for the proposed FV solvers. A brief overview of particular similarities and differences between FV and FE methods is given below.

[2]Open Source Field Operation and Manipulation

[3]Total Lagrangian Upwind Cell Centred Finite Volume Method for Hyperbolic conservation laws

In general, both methods may be considered as methods of weighted residuals, but they differ in the choice of the weighting function. While the Galerkin method used for FE applies shape functions as weighting functions, in the FVM the weighting functions are chosen as unity [15, 42, 43]. Therefore, the FVM may be considered as a particular case of the FEM with non-Galerkin weighting [15, 39, 41].

Nevertheless, these two numerical schemes differ in their mathematical background: FE methods base on the variational principle, where the governing equations in strong form are transformed into their equivalent weak form [8]. The method can be easily extended to higher-order discretisation. The FVM, on the other hand, is usually second-order accurate, and it is based on the strong integral formulation of the conservation laws. Therefore, the equations automatically satisfy local conservation properties at the discrete level [4, 20, 21, 46].

Moreover, there are differences regarding the residence of the primary variables. While in a cell-centred approach the values, e.g. displacements, are stored in the centroids of the cells, in FE methods the numerical quantities are stored in the vertices of the element[4] [20, 21].

In contrast to FE methods where pre-defined shape functions are used depending on the topology of the element, the discretisation applied in the FVM allows to construct a second-order accurate discretisation independent of the shape of the cell. It can be easily applied to any arbitrary grid. Therefore, it is possible that different cell shapes are *mixed and matched* at will [26].

There are also differences between FV and FE methods in the solution process of the system of algebraic equations. On the one hand, the FVM generally uses a segregated solution procedure, in which coupling and nonlinearity are treated in an iterative manner [15, 26]. On the other hand, for the solution process in the FEM, the matrices are normally given in block-coupled form and solved implicitly [26].

1.3. Scope and Outline

The objective of this thesis is the development of an implicit cell-centred FV solver for nonlinear elasticity analogous to the approach used for FE methods. Therefore, the task is not to adapt the governing equations for large deformations to the solution process of the general FV approaches (like in state of the art approaches, cf. Section 1.1), but to adapt the FV methodology to the approach including the linearisation via directional derivatives. A major difference between common FV approaches and the proposed solution is the treatment of non-orthogonal terms, so-called cross diffusion. In the proposed algorithm, the treatment is done in a different way, while the deferred correction approach distinguishes between implicit (orthogonal) and explicit (nonorthogonal) terms.

[4]The element is the equivalent to the cell in the FVM.

The proposed solution algorithm is formulated based on the equilibrium of internal forces in total *Lagrangian* formulation, neglecting transient terms and body forces. This thesis only focuses on compressible isotropic hyperelastic constitutive models in order to close the equation system. The arising MATLAB© code is entitled SOOFVAM and is discussed in Chapter 5.

In general, this thesis is divided into two main parts. The first part introduces continuum mechanics for solids (Chapter 2), including both small and large deformation theory, as well as a description of the FVM (Chapter 3). These two topics are combined in Chapter 4, where the governing implicit equations are developed.

The second part starts with a detailed discussion (Chapter 5) of the equations derived in Chapter 4, and illustrates extracts of the implemented code. A main focus is set on the derivation of the explicit and implicit forms of cell gradients with the least-squares method and the treatment of Dirichlet boundary conditions in the implicit (and iterative) solution procedure. Results of the simulation, including a comparison to OpenFOAM© and foam-extend, are presented in Chapter 6. Finally, the findings are summarized and an outlook of possible further steps is given in Chapter 7.

Since this thesis depends on tensorial mathematical formulations, the discerning reader may have a look in any textbook related to tensor algebra. Moreover, the concept of continuum mechanics is described in more detail in e.g. [5, 14, 19, 23, 32]. For a detailed description of the FVM the author recommends literature including [4, 16, 21, 35, 46], since this thesis only deals with the fundamentals needed for the proposed solution.

1.4. Mathematical Preliminaries

In the subsequent chapters the reader will encounter different types of tensors. In order to get in touch with the notation of tensors used in this thesis, a few coherences regarding tensor algebra are described below. The following terms and subjects are presented from a more practical (i.e. engineering) point of view without any excessive use of mathematical language.

In general, most of the physical coherences in CM are expressed in form of tensors [22]. The basis of tensorial objects is a so-called vector space $\mathcal{V}$ with n dimensions, whereas in this thesis it holds $n = 3$ or $n = 2$[5].

Forms of Notation

In general, one can distinguish between the symbolic and index notation of tensors. Since both forms of notation are used in this thesis, they are briefly introduced.

[5]Most of the simulated test cases in Chapter 6 deal with a two-dimensional domain.

Each vector $\underline{v} \in \mathcal{V}$ can be defined as the linear combination of the vector space's basis vectors

$$\underline{v} = \sum_{i=1}^{3} v^i \, \underline{b}_i \, , \tag{1.2}$$

where the contravariant, i.e. superscript, term v^i refers to the components of the vector $\underline{v}$ and the subscript term $\underline{b}_i$ represents the basis vectors of $\mathcal{V}$. The left-hand side (LHS) of Equation (1.2) refers to the symbolic notation and as may be observed, vectors are identified with a line underneath the variable in this thesis. The right-hand side (RHS) refers to the index notation, whereas in this thesis the distinction is made between superscript and subscript indices.

As one can imagine, if there are many vectors (tensors) involved in an equation, one would have to write many summation signs. In order to circumvent this issue, the summation convention for general curvilinear coordinate systems is introduced (also called *Einstein summation convention*) [14]: if an index appears once as superscript and once as subscript in a term, one has to sum over this index and the term is written without the summation sign, i.e.

$$\underline{v} = \sum_{i=1}^{3} v^i \, \underline{b}_i = v^i \, \underline{b}_i \, . \tag{1.3}$$

Within this context, it is not possible that the same index appears more than once as superscript or subscript. Since in the FV discretisation process integrals are replaced by summation signs, every summation sign that appears in an equation will be explained from now on.

Higher-order tensors are notated in an analogous way, e.g.

$$\begin{aligned} \underline{\underline{A}} &= A^{ij} \, \underline{b}_i \otimes \underline{b}_j \, , \\ \underline{\underline{\underline{B}}} &= B_{ijk} \, \underline{b}^i \otimes \underline{b}^j \otimes \underline{b}^k \, , \\ \underline{\underline{\underline{\underline{C}}}} &= C^i_{jkl} \, \underline{b}_i \otimes \underline{b}^j \otimes \underline{b}^k \otimes \underline{b}^l \, , \end{aligned} \tag{1.4}$$

where $\underline{\underline{A}}, \underline{\underline{\underline{B}}}$ and $\underline{\underline{\underline{\underline{C}}}}$ are arbitrary tensors of order two, three and four, respectively. The number of indices defines the order of the tensor, e.g. C^i_{jkl} represents the components of the fourth-order tensor $\underline{\underline{\underline{\underline{C}}}}$. Especially in Chapter 2 and Chapter 4, the reader will encounter the index notation of tensors.

In the introduction of the FVM (Chapter 3), the symbolic notation is sufficient. However, subscripts will be needed in order to define if a term relates to e.g. a cell or face. This must not be confused with the indices as described above. Therefore, these subscripts often will be separated from the terms with parantheses, e.g. $(.)_\mathrm{f}$ refers to an arbitrary term of a face. Moreover, the subscripts $(.)_\mathrm{C}$, $(.)_\mathrm{F}$ and $(.)_\mathrm{b}$ relate to owner cells, neighbour cells and boundary faces, respectively.

As can be observed in Equation (1.4), in symbolic notation the number of lines underneath the tensor defines the order for higher-order tensors.

Special Forms

Two special forms used in tensor algebra are finally defined below. On the one hand, there is the *Kronecker delta* δ^i_j, which is defined as [22, 23]:

$$\delta^i_j = \begin{cases} 1\,, & \text{if } i = j \\ 0\,, & \text{if } i \neq j\,. \end{cases} \tag{1.5}$$

In symbolic notation, Equation (1.5) may be read as $\underline{\underline{I}}$, which is the second-order identity matrix.

On the other hand, there is the *metric tensor* $\underline{\underline{g}}$ that can raise and lower the indices of tensors, i.e.

$$A_i = g_{ij}\ B^j\,, \tag{1.6}$$

where the metric tensor g_{ij} is used to lower the index of the first-order contravariant tensor B^j to form the first-order covariant tensor A_i.

In general, the metric tensor is needed to describe lengths and angles between vectors in arbitrary coordinate systems and is defined as

$$\underline{u} \cdot \underline{w} := g[\underline{u}\,, \underline{w}] = g_{ij}\ u^i\ w^j\,, \tag{1.7}$$

which refers to the *dot product* of two vectors in an arbitrary coordinate system. The length of $\underline{u}$ can be calculated with the metric tensor as

$$\|\underline{u}\| = \sqrt{\underline{u} \cdot \underline{u}} = \sqrt{g_{ij}\ u^i\ u^j}\,, \tag{1.8}$$

and the angle φ between $\underline{u}$ and $\underline{w}$ is calculated as

$$\cos(\varphi) = \frac{\underline{u} \cdot \underline{w}}{\|\underline{u}\| \cdot \|\underline{w}\|}\,. \tag{1.9}$$

For Cartesian coordinate systems it holds $g_{ij} = \delta_{ij}$ (*Kronecker* delta, cf. Equation (1.5)) [22]. This is also true for the other variant form of the metric tensor, i.e. $g^{ij} = \delta^{ij}$.

2. Continuum Mechanics

In this chapter, both the small and large deformation theory of continuum mechanics are introduced. Thus, topics including kinematics for finite and infinite deformations (Section 2.1), the concept of stress (Section 2.2), the equilibrium state and the resulting boundary value problem (Section 2.3) as well as constitutive equations (Section 2.4) are discussed. The last section of this chapter addresses the *Newton-Raphson* method, which is a tool that is not solely used in CM. The *Newton* algorithm is explained for one-dimensional functions $f(x)$ and is then applied for arbitrary functionals $\mathcal{F}(\underline{x})$.

In the context of this thesis the term *continuum mechanics* refers to continuum mechanics for *solids*.

2.1. Kinematics

2.1.1. Configurations, Mapping and Displacement

If the body $\mathcal{B}$ in Figure 1.1 moves through time and space (e.g. as the result of external forces), it occupies a continuous sequence of geometrical regions. Two specific geometrical regions of this motion are denoted as $\mathcal{B}_0$ and $\mathcal{B}_t$ and are shown in Figure 2.1. The *reference configuration*[1] $\mathcal{B}_0$ refers to an initial state of the body at a reference time $t = 0$. As may be observed in Figure 2.1, a particle point $P \in \mathcal{B}_0$ is identified by its position vector $\underline{X}$. In contrast, the *current configuration*[2] $\mathcal{B}_t$ refers to the current state of the body at an arbitrary time $t = \bar{t} > 0$. As a result of the motion, the point P gets transferred, i.e. mapped, to the point $p \in \mathcal{B}_t$ and is identified by its position vector $\underline{x}$. Alternatively, the reference and current configuration may be also referred to as *Lagrangian* and *Eulerian* description, respectively [5, 14, 22, 23].

The discerning reader may notice that an uppercase letter is used for the position vector of the reference configuration and that the position vector of the current configuration is written in lowercase, cf. Figure 2.1. In general, this thesis refers to the reference configuration with uppercase letters and the current configuration with lowercase letters. This notation holds also true for e.g. the index notation of tensors.

In the following, only the two mentioned configurations are of interest and not the motion itself, therefore the transition from the reference to the current configuration is

[1]It is also called *undeformed*, *initial* or *material configuration*.
[2]It is also called *deformed* or *spatial configuration*.

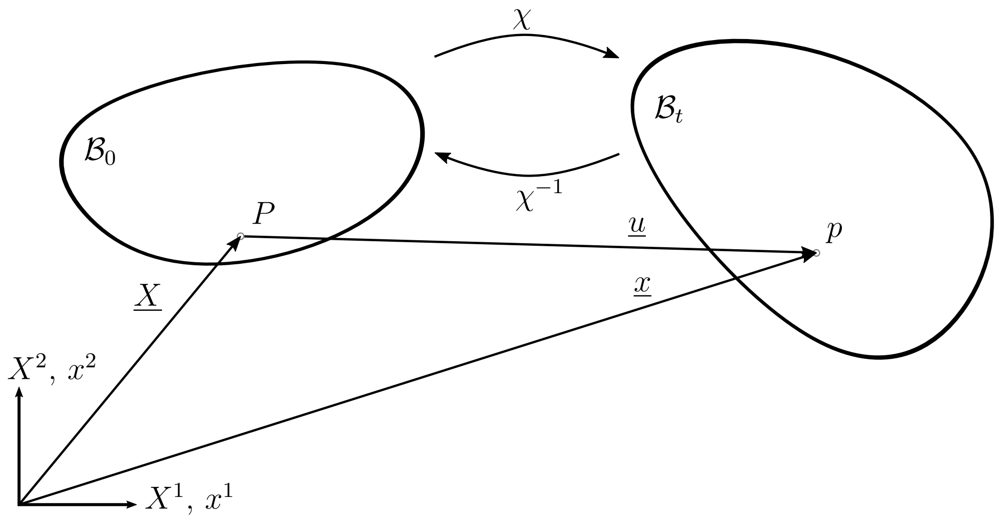


Figure 2.1.: Configurations and Mapping of a Body

further called mapping χ. In general, the coherences described above yield [5, 23]

$$\underline{x} = \chi(\underline{X}, t) \iff \underline{X} = \chi^{-1}(\underline{x}, t)\,, \tag{2.1}$$

whereby in this thesis only the state of equilibrium as the result of a load is of interest. The time-dependency in Equation (2.1) can therefore be neglected, yielding

$$\underline{x} = \chi(\underline{X}) \iff \underline{X} = \chi^{-1}(\underline{x})\,. \tag{2.2}$$

Note that the mapping will change the position, orientation and shape of the body $\mathcal{B}$ in general, cf. Figure 2.1. This so-called deformation mapping χ maps each point from the reference ($P \in \mathcal{B}_0$) into the current configuration ($p = \chi(P) \in \mathcal{B}_t$). The difference between the position of a particle in the undeformed and deformed configuration can be described as

$$\underline{u} = \underline{x} - \underline{X}\,. \tag{2.3}$$

The *displacement* vector $\underline{u}$ (in the deformed configuration) holds for all particles of the continuum [23] and is the primary variable to be calculated.

2.1.2. Deformation Gradient

As illustrated in Figure 2.1, the mapping χ causes a general alteration of the body $\mathcal{B}$ and therefore also changes the orientation of tangent vectors in particle points. Figure 2.2 shows the reference configuration with an arbitrary point $P \in \mathcal{B}_0$. The tangent vectors of P with respect to the coordinate directions are illustrated as well. Moreover, the effect of the deformation is emphasized graphically with a structured, orthogonal mesh that is aligned with the coordinate directions of the reference configuration. Due to the mapping of the point P into $p = \chi(P)$, the tangent vectors as well as the mesh do not resemble the initial state, cf. Figure 2.2.

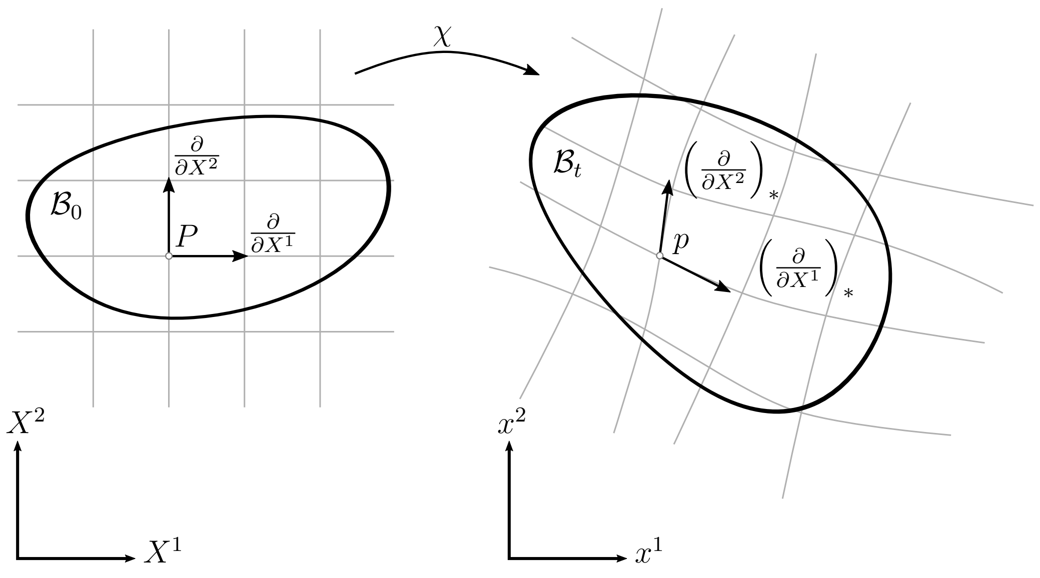

χ
$\mathcal{B}_0$
$\frac{\partial}{\partial X^2}$
P
$\frac{\partial}{\partial X^1}$
$\mathcal{B}_t$
$\left(\frac{\partial}{\partial X^2}\right)_*$
p
$\left(\frac{\partial}{\partial X^1}\right)_*$
X^2
X^1
x^2
x^1

Figure 2.2.: Mapping of Tangent Vectors

Local deformations in the surroundings of a point are characterized with the mapping of the tangent vectors. This is described with the mapping $F = D\chi$, which is a linearisation, and is called *deformation gradient.* It establishes the fundamental relationship [5, 23]

$$d\underline{x} = \underline{\underline{F}}\, d\underline{X}\,, \tag{2.4}$$

in which the definition of the deformation gradient is used that can be described as [22, 23]

$$\underline{\underline{F}} = \frac{\partial \underline{x}}{\partial \underline{X}} \quad \text{or} \quad F^i_I = \frac{\partial x^i}{\partial X^I}\,. \tag{2.5}$$

As one notices, Equation (2.4) describes a linear mapping between two vectors. The deformation gradient is also called a *two-point tensor* because it connects points from two different configurations [23]. As a result, the deformation gradient can be used to describe the mapping of the basis vectors from the reference into the current configuration. This mapping reads in index notation [22]

$$\left(\frac{\partial}{\partial X^I}\right)_* = \frac{\partial p}{\partial X^I} = \frac{\partial p}{\partial x^i} \cdot \frac{\partial x^i}{\partial X^I} = F^i_I\, \frac{\partial}{\partial x^i} \tag{2.6}$$

and is the so-called *push-forward* (denoted with a subscript asterisk). The inverse mapping of e.g. the basis vectors of the current configuration reads [22]

$$\left(\frac{\partial}{\partial x^i}\right)^* = \check{F}^I_i\, \frac{\partial}{\partial X^I} \tag{2.7}$$

and is the so-called *pull-back* (denoted with a superscript asterisk). $\check{F}^I_i$ relates to the components of $\underline{\underline{F}}^{-1}$, which is the inverse of $\underline{\underline{F}}$, because it holds true that

$$\delta^I_J = \frac{\partial X^I}{\partial X^J} = \frac{\partial X^I}{\partial x^i} \cdot \frac{\partial x^i}{\partial X^J} = \check{F}^I_i \, F^i_J \,. \tag{2.8}$$

The push-forward and pull-back operations exist not only for the basis vectors but also for all other tensorial objects, as can be seen below.

2.1.3. Strain

In contrast to the theory of small deformations, there is no unique way to define a strain measure in nonlinear elasticity. Unlike displacements, which are measurable quantities, strains base on a *concept* to simplify analysis. Numerous possibilities for the definition of strain tensors have been proposed in the literature [23]. Strain tensors compare deformation tensors, that describe the alteration of lengths and angles, with the corresponding metric tensor of the reference or current configuration. The strain tensor used to derive the governing equations in this thesis is the *Green-Lagrange strain* tensor $\underline{\underline{E}}$, which measures changes in lengths and angles with respect to the reference configuration. It is defined as [22, 23]

$$\underline{\underline{E}} = \frac{1}{2} \left(\underline{\underline{C}} - \underline{\underline{G}} \right), \tag{2.9}$$

with the second-order metric tensor of the reference configuration $\underline{\underline{G}}$ and the right *Cauchy-Green deformation* tensor $\underline{\underline{C}}$, which is defined as the pull-back of the metric tensor of the current configuration $\underline{\underline{g}}$ [22, 23]:

$$\underline{\underline{C}} = \underline{\underline{g}}^* = \underline{\underline{F}}^T \, \underline{\underline{g}} \, \underline{\underline{F}} \,. \tag{2.10}$$

If Equation (2.10) is substituted in Equation (2.9), one obtains

$$\underline{\underline{E}} = \frac{1}{2} \left(\underline{\underline{F}}^T \, \underline{\underline{g}} \, \underline{\underline{F}} - \underline{\underline{I}} \right) \quad \text{or} \quad E_{IJ} = \frac{1}{2} \left(F^i_I \, g_{ij} \, F^j_J - \delta_{IJ} \right), \tag{2.11}$$

where $\underline{\underline{G}} = \underline{\underline{I}}$ and δ_{IJ} refers to the *Kronecker* delta.

The deformation gradient $\underline{\underline{F}}$ can be calculated in terms of the gradient of the displacement vector $\underline{u}$ and reads

$$\underline{\underline{F}} = \frac{\partial \underline{x}}{\partial \underline{X}} = \frac{\partial \underline{x}}{\partial \underline{X}} + \frac{\partial \underline{X}}{\partial \underline{X}} - \frac{\partial \underline{X}}{\partial \underline{X}} = \frac{\partial \underline{x} - \partial \underline{X}}{\partial \underline{X}} + \frac{\partial \underline{X}}{\partial \underline{X}} = \frac{\partial \underline{u}}{\partial \underline{X}} + \underline{\underline{I}} \,, \tag{2.12}$$

where $\frac{\partial \underline{u}}{\partial \underline{X}}$ relates to the displacement gradient. Equation (2.12) reads in index notation

$$F^i_I = u^i_{,I} + \delta^i_I \,, \tag{2.13}$$

where the comma in $u^i_{,I}$ refers to the partial derivative of the displacement vector $\underline{u}$ with respect to $\underline{X}$. If Equation (2.13) is substituted in Equation (2.11), one obtains

$$\begin{aligned} E_{IJ} &= \frac{1}{2}\,(F^i_I\; g_{ij}\; F^j_J - \delta_{IJ}) = \frac{1}{2}\,(g_{ij}\;(u^i_{,I} + \delta^i_I)(u^j_{,J} + \delta^j_J) - \delta_{IJ}) = \\ &= \frac{1}{2}\,(g_{ij}\;\delta^i_I\;\delta^j_J + g_{ij}\;u^i_{,I}\;\delta^j_J + g_{ij}\;u^j_{,J}\;\delta^i_I + g_{ij}\;u^i_{,I}\;u^j_J - \delta_{IJ})\,. \end{aligned} \tag{2.14}$$

With the equality $g_{ij} = \delta_{ij}$, one obtains

$$\begin{aligned} E_{IJ} &= \frac{1}{2}\,(\delta_{ij}\;\delta^i_I\;\delta^j_J + \delta_{ij}\;u^i_{,I}\;\delta^j_J + \delta_{ij}\;u^j_{,J}\;\delta^i_I + \delta_{ij}\;u^i_{,I}\;u^j_J - \delta_{IJ}) = \\ &= \frac{1}{2}\,(\delta_{jI}\;\delta^j_J + u^i_{,I}\;\delta_{iJ} + u^j_{,J}\;\delta_{jI} + \delta_{ij}\;u^i_{,I}\;u^j_{,J} - \delta_{IJ}) = \\ &= \frac{1}{2}\,(u^i_{,I}\;\delta_{iJ} + u^j_{,J}\;\delta_{jI} + \delta_{ij}\;u^i_{,I}\;u^j_{,J})\,. \end{aligned} \tag{2.15}$$

Note that the last term in Equation (2.15) is nonlinear, which yields a *nonlinear* kinematic relationship for finite deformations.

2.1.4. Strain for Small Deformations

Since some of the test cases in Chapter 6 deal with linear elasticity, some of the essential aspects of the theory of small deformations in relation to kinematics are presented below.

If the displacement gradients in Equation (2.15) are *small* in comparison to unity, then products of this terms are negligible [18]. For such cases, the *Green-Lagrange* strain tensor in Equation (2.15) simplifies to

$$E_{ij} = \frac{1}{2}\,(u_{i,j} + u_{j,i}) =: \varepsilon_{ij}\,, \tag{2.16}$$

whereby ε_{ij} relates to the components of the second-order *infinitesimal* or *small strain* tensor $\underline{\underline{\varepsilon}}$. All the indices of the tensors in Equation (2.16) are written in lowercase letters because for small deformations one assumes that there is no (finite) difference between reference and current configuration. As can be observed, Equation (2.16) yields a *linear* kinematic relationship for small deformations.

2.2. Stress Tensors

The deformation of the continuum body due to external loading gives rise to interactions between neighbouring particles, resulting in internal forces [14, 23]. The relation of forces to areas leads to the term *stress* with the physical dimension force per unit area. The concept of stress is linked to *Cauchy's* stress theorem.

In order to develop the concept of stress, a body $\mathcal{B}$ in its current configuration as shown in Figure 2.3 is investigated. It contains two regions Ω_1 and Ω_2 that are in contact. On

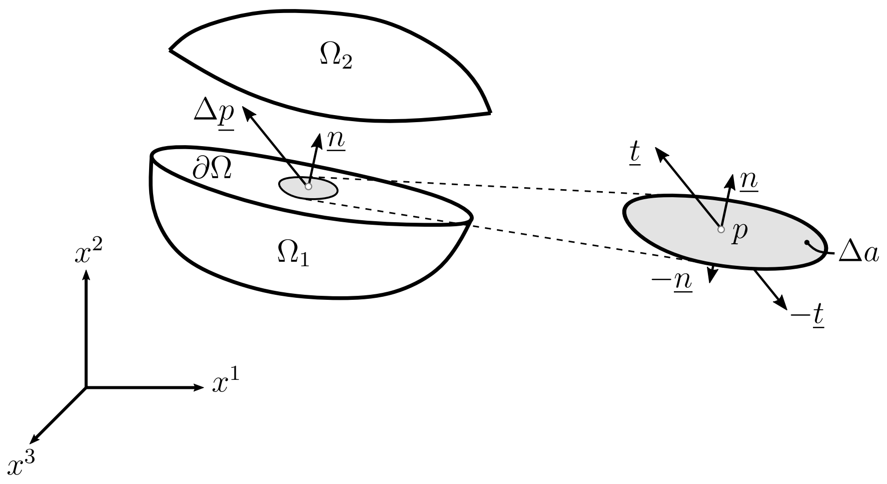


Figure 2.3.: Traction Vector (inspired by [5])

the contact surface $\partial\Omega$, an arbitrary particle point p is chosen, whereas the area Δa normal to $\underline{n}$ defines the neighbourhood of the point. If $\Delta \underline{p}$ is the resultant force, the traction vector $\underline{t}$ at p is then defined as [5]

$$\underline{t}(\underline{n}) = \lim_{\Delta a \to 0} \frac{\Delta \underline{p}}{\Delta a}\,, \tag{2.17}$$

whereas *Newton's* third law of action and reaction must hold, $\underline{t}(-\underline{n}) = -\underline{t}(\underline{n})$. In general, there is a linear relation between the traction vector $\underline{t}$ and the normal vector $\underline{n}$[3], namely

$$\underline{t} = \underline{\underline{\sigma}}\ \underline{n} \quad \text{or} \quad t^i = \sigma^{ij}\ n_j\,, \tag{2.18}$$

with the second-order *Cauchy stress* tensor $\underline{\underline{\sigma}}$. Equation (2.18) is *Cauchy's* stress theorem [23].

Alternative Stress Tensors

Similar to strain tensors, numerous definitions of stress tensors have been proposed in the literature [23]. Since the deformed configuration is not known in advance, it is not convenient to work with stress tensors defined on the current configuration, e.g. the *Cauchy* stress tensor [23]. Therefore, the derivation of the *second Piola-Kirchhoff stress* tensor $\underline{\underline{S}}$, which is solely defined on the reference configuration, is developed[4].

The pull-back operation with respect to the second index of the *Cauchy* stress tensor $\underline{\underline{\sigma}}$ (via *Piola* transformation) yields the *first Piola-Kirchhoff stress* tensor $\underline{\underline{P}}$ [14, 22, 23]:

$$\underline{\underline{P}} = J\ \underline{\underline{\sigma}}\ \underline{\underline{F}}^{-T} \quad \text{or} \quad P^{iJ} = J\ \sigma^{ij}\ \check{F}^{J}_{j}\,, \tag{2.19}$$

[3]if $\underline{t}$ is dependent on $\underline{n}$ [23]

[4]The second *Piola-Kirchhoff* stress tensor is the work conjugate stress to the *Green-Lagrange* strain tensor, cf. Section 2.4.

where J is defined as the determinant of the deformation gradient $\underline{\underline{F}}$, $J = \det(\underline{\underline{F}})$. The first *Piola-Kirchhoff* stress tensor $\underline{\underline{P}}$ relates forces on the deformed configuration to areas on the reference configuration. The pull-back operation with respect to the first index of the first *Piola-Kirchhoff* stress tensor $\underline{\underline{P}}$ or the *full* pull-back of the *Cauchy* stress tensor $\underline{\underline{\sigma}}$ yields the second *Piola-Kirchhoff* stress tensor $\underline{\underline{S}}$, which reads [22, 23]

$$\underline{\underline{S}} = \underline{\underline{F}}^{-1}\ \underline{\underline{P}} = J\ \underline{\underline{F}}^{-1}\ \underline{\underline{\sigma}}\ \underline{\underline{F}}^{-T} \quad \text{or} \quad S^{IJ} = \check{F}^I_i\ P^{iJ} = J\ \check{F}^I_i\ \sigma^{ij}\ \check{F}^J_j\,, \tag{2.20}$$

and relates forces to areas both on the undeformed configuration.

2.3. Equilibrium and Boundary Value Problem

The governing equations are derived from the balance of translational momentum for a body in its current configuration $\mathcal{B}_t$, which reads [14, 19, 23]

$$\int\limits_{\mathcal{B}_t} \rho \ddot{\underline{u}}(\underline{x})\,\mathrm{d}v = \int\limits_{\partial\mathcal{B}_t} \underline{t}(\underline{x})\,\mathrm{d}a + \int\limits_{\mathcal{B}_t} \underline{b}(\underline{x})\,\mathrm{d}v\,. \tag{2.21}$$

The term $\ddot{\underline{u}}(\underline{x}) = \frac{\partial^2 u}{\partial(t)^2}$ refers to the acceleration and is neglected for further calculations, i.e. no time-dependency as mentioned, $\frac{\partial}{\partial t} = \frac{\partial^2}{\partial(t)^2} = 0$. Only the equilibrium state of the body $\mathcal{B}$ is of interest. It reads

$$\int\limits_{\partial\mathcal{B}_t} \underline{t}(\underline{x})\,\mathrm{d}a + \int\limits_{\mathcal{B}_t} \underline{b}(\underline{x})\,\mathrm{d}v = \underline{0}\,, \tag{2.22}$$

with the traction forces $\underline{t}(\underline{x})$ per unit area and body forces $\underline{b}(\underline{x})$ per unit volume. If Equation (2.18) is substituted into Equation (2.22) and one applies the *divergence theorem of Gauss* for the surface integral, one obtains

$$\begin{aligned}\int\limits_{\partial\mathcal{B}_t} \underline{t}\,\mathrm{d}a + \int\limits_{\mathcal{B}_t} \underline{b}\,\mathrm{d}v &= \int\limits_{\partial\mathcal{B}_t} \underline{\underline{\sigma}}\ \underline{n}\,\mathrm{d}a + \int\limits_{\mathcal{B}_t} \underline{b}\,\mathrm{d}v = \\ = \int\limits_{\mathcal{B}_t} \underline{\nabla}\cdot\underline{\underline{\sigma}}\,\mathrm{d}v + \int\limits_{\mathcal{B}_t} \underline{b}\,\mathrm{d}v &= \int\limits_{\mathcal{B}_t} (\underline{\nabla}\cdot\underline{\underline{\sigma}} + \underline{b})\,\mathrm{d}v = \underline{0}\,,\end{aligned} \tag{2.23}$$

where $\underline{\nabla}$ relates to the *del* (or *nabla*) operator. Considering that the equation above can be applied to any computational domain, the integrand of Equation (2.23) must vanish and results in

$$\underline{\nabla}\cdot\underline{\underline{\sigma}} + \underline{b} = \underline{0}\,. \tag{2.24}$$

Due to simplicity, body forces are neglected ($\underline{b} = \underline{0}$) for further calculations:

$$\underline{\nabla}\cdot\underline{\underline{\sigma}} = \underline{0}\,. \tag{2.25}$$

Equation (2.25) is the PDE of the equilibrium of internal forces to be solved. The angular momentum of the body results in the symmetry of the *Cauchy* stress tensor $\underline{\underline{\sigma}}$ [14], $\sigma^{ij} = \sigma^{ji}$, and is not discussed further here. Within this context, the boundary value problem to solve is introduced as follows [22]:

From a given body $\mathcal{B}$ in its reference configuration $\mathcal{B}_0$, the deformed configuration $\mathcal{B}_t = \chi(\mathcal{B}_0)$ is to be calculated in the form of the deformation mapping $\underline{x}(\underline{X}) = \chi_t(\underline{X})$. The body $\mathcal{B}$ is prescribed on a part of the boundary, i.e. the *Dirichlet* boundary $\partial_D\mathcal{B}_0$ and the *Neumann* boundary $\partial_N\mathcal{B}_t := \partial\mathcal{B}_t \backslash \partial_D\mathcal{B}_0$. As the body is loaded, the equilibrium state of the resulting deformed configuration is to be calculated, i.e.

$$\begin{aligned} \underline{\nabla} \cdot \underline{\underline{\sigma}} &= \underline{0}\,, & \underline{x} &\in \mathcal{B}_t \\ \underline{\underline{\sigma}} \cdot \underline{n} &= \underline{t}\,, & \underline{x} &\in \partial_N\mathcal{B}_t \\ \chi_t(\underline{X}) &= \overline{\chi}_t(\underline{X})\,, & \underline{X} &\in \partial_D\mathcal{B}_0\,, \end{aligned} \tag{2.26}$$

where the second line in Equation (2.26) refers to the prescribed traction or Neumann boundary condition (BC) and the third line relates to the prescribed displacement or Dirichlet BC.

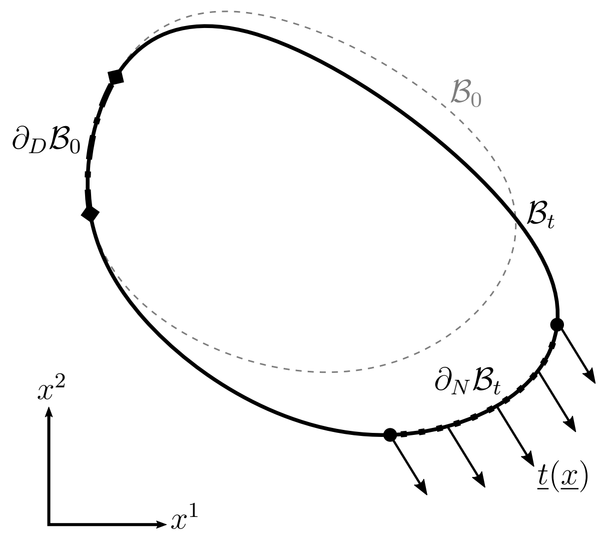

$\mathcal{B}_0$
$\partial_D\mathcal{B}_0$
$\mathcal{B}_t$
x^2
$\partial_N\mathcal{B}_t$
$\underline{t}(\underline{x})$
x^1

Figure 2.4.: Boundary Value Problem of Nonlinear Elasticity

The coherences described above are illustrated in Figure 2.4. The reference configuration $\mathcal{B}_0$ is sketched with a dashed grey line, while the current configuration $\mathcal{B}_t$ is shown as solid black. The Dirichlet and Neumann boundaries are both highlighted. While the Dirichlet boundary $\partial_D\mathcal{B}_0$ is emphasized with a dotdashed line and ends in a diamond shape, the Neumann boundary $\partial_N\mathcal{B}_t$ with the prescribed traction $\underline{t}$ is highlighted with a dotted line and dotted ends. The leftover boundary also refers to a Neumann boundary, but with a traction force $\underline{t} = \underline{0}$[5].

[5]It is also called implicit Neumann boundary.

For the FV discretisation process, Equation (2.25) is integrated over the computational domain, which reads

$$\int_{\mathcal{B}_t} \underline{\nabla} \cdot \underline{\underline{\sigma}} \, \mathrm{d}v = \underline{0} \,. \tag{2.27}$$

The linearisation (cf. Section 2.5) of the nonlinear equations in Equation (2.27) is executed for the reference configuration. Therefore, transformations including pull-back operations, the divergence theorem of *Gauss* as well as the substitution of Equation (2.19) result in

$$\int_{\mathcal{B}_t} \underline{\nabla} \cdot \underline{\underline{\sigma}} \, \mathrm{d}v = \int_{\partial\mathcal{B}_t} \underline{\underline{\sigma}} \, \mathrm{d}\underline{a} = \int_{\partial\mathcal{B}_0} \underline{\underline{\sigma}} \; J \; \underline{\underline{F}}^{-T} \, \mathrm{d}\underline{A} = \int_{\partial\mathcal{B}_0} \underline{\underline{P}} \, \mathrm{d}\underline{A} = \underline{0} \,. \tag{2.28}$$

With the relationship $\underline{\underline{S}} = \underline{\underline{F}}^{-1} \; \underline{\underline{P}} \implies \underline{\underline{P}} = \underline{\underline{F}} \; \underline{\underline{S}}$ from Equation (2.20), one obtains

$$\int_{\partial\mathcal{B}_0} \underline{\underline{P}} \, \mathrm{d}\underline{A} = \int_{\partial\mathcal{B}_0} \underline{\underline{F}} \; \underline{\underline{S}} \, \mathrm{d}\underline{A} = \int_{\partial\mathcal{B}_0} \underline{\underline{F}} \; \underline{\underline{S}} \; \underline{N} \, \mathrm{d}A = \underline{0} \,. \tag{2.29}$$

As may be observed, the differential surface vector $\mathrm{d}\underline{A} = \underline{N} \; \mathrm{d}A$ is split into normal vector (of reference configuration) $\underline{N}$ and differential area $\mathrm{d}A$. Equation (2.29) refers to the set of equations used for the derivation of the implicit governing equations in Chapter 4.

2.4. Constitutive Equations

Although the equations introduced in the previous sections do hold for a continuous body, they do not differentiate between individual materials. This information is set with the material (or constitutive) law. The constitutive law approximates the physical behaviour of a material under certain conditions of interest [23].

There are a lot of different constitutive theories, and they establish a relationship between stresses and strains. A specific form of such a theory is briefly discussed below: the finite hyperelasticity theory [23], in which stresses are derived from stored elastic energy functions [5].

All structural materials inherit to a certain degree the property of *elasticity*, i.e. induced deformations disappear when the external forces of a loaded body are removed [44]. Only if a material is loaded to a level below the *yield strain* and if the unloading follows the same path, the material can be considered *elastic* [18, 44]. For the finite deformations dealt with in this thesis, the materials exhibit a *nonlinear* elastic behaviour which manifests itself in a curved loading and unloading path.

An elastic material model generally predicts a material behaviour independent of material history. Stresses are *univocally* determined by strains and vice versa [14]. As a consequence, elastic material models cannot be used to model effects such as permanent deformation, material damage or creep [14].

A so-called hyperelastic material postulates the existence of a scalar *elastic potential* function Ψ, which is defined per unit reference volume. It holds that the elastic potential is dependent on the current deformation or strain condition [22]. Therefore, it is a function of an arbitrary strain measure $\underline{\underline{\epsilon}}$, i.e. $\Psi = \Psi(\underline{\underline{\epsilon}})$.

If the work done by stresses during a deformation process is only dependent on the reference and current configuration, the behaviour of a material is said to be path-independent and implies a potential [5]. As a result, the stored elastic potential function can be established as the work done by the stresses and reads [22]

$$\Psi(\underline{\underline{\epsilon}}) = \int_0^{\underline{\underline{\epsilon}}} \underline{\underline{\Sigma}}(\underline{\underline{\tilde{\epsilon}}})\,\mathrm{d}\underline{\underline{\tilde{\epsilon}}} \quad \Longleftrightarrow \quad \underline{\underline{\Sigma}}(\underline{\underline{\epsilon}}) = \frac{\partial \Psi(\underline{\underline{\epsilon}})}{\partial \underline{\underline{\epsilon}}}\,, \tag{2.30}$$

where $\underline{\underline{\Sigma}}$ refers to an arbitrary stress tensor which is work conjugate to the corresponding strain measure $\underline{\underline{\epsilon}}$. In this thesis, the work conjugate pair consisting of the second *Piola-Kirchhoff* stress tensor and the *Green-Lagrange* strain tensor ($\underline{\underline{S}}$, $\underline{\underline{E}}$) is used for the material law description. The relation in Equation (2.30) then becomes

$$\underline{\underline{S}}(\underline{\underline{E}}) = \frac{\partial \Psi(\underline{\underline{E}})}{\partial \underline{\underline{E}}} \quad \text{or} \quad S^{IJ} = \frac{\partial \Psi}{\partial E_{IJ}}\,. \tag{2.31}$$

In addition, for a work conjugate pair $(\underline{\underline{\Sigma}}\,, \underline{\underline{\epsilon}})$ the *elasticity* tensor $\underline{\underline{\underline{\underline{\mathcal{C}}}}}$ is defined as [22]

$$\underline{\underline{\underline{\underline{\mathcal{C}}}}}(\underline{\underline{\epsilon}}) = \frac{\partial \underline{\underline{\Sigma}}}{\partial \underline{\underline{\epsilon}}} = \frac{\partial^2 \Psi}{\partial \underline{\underline{\epsilon}} \partial \underline{\underline{\epsilon}}}\,. \tag{2.32}$$

The elasticity tensor for the work conjugate pair $(\underline{\underline{S}}\,, \underline{\underline{E}})$ is the *material elasticity* tensor $\underline{\underline{\underline{\underline{\mathbb{C}}}}}$, which reads

$$\underline{\underline{\underline{\underline{\mathbb{C}}}}}(\underline{\underline{E}}) = \frac{\partial \underline{\underline{S}}}{\partial \underline{\underline{E}}} = \frac{\partial^2 \Psi}{\partial \underline{\underline{E}} \partial \underline{\underline{E}}} \quad \text{or} \quad \mathbb{C}^{IJKL} = \frac{\partial S^{IJ}}{\partial E_{KL}} = \frac{\partial \Psi}{\partial E_{IJ} \partial E_{KL}}\,, \tag{2.33}$$

and it inherits the symmetry of the strain tensor E^{IJ} in its first two and second two indices, $\mathbb{C}^{IJKL} = \mathbb{C}^{JIKL}$ and $\mathbb{C}^{IJKL} = \mathbb{C}^{IJLK}$. Additionally, *Schwarz'* theorem holds: $\mathbb{C}^{IJKL} = \mathbb{C}^{KLIJ}$.

2.4.1. Compressible Isotropic Hyperelastic Material Models

In the following, two material models of isotropic hyperelastic material behaviour are presented. An elastic potential function Ψ is called *isotropic*[6] if it is a function not only

[6]Isotropy is defined by the requirement that the material behaviour is identical in every material direction [5].

of a strain measure, but of the invariants of a strain measure, i.e.

$$\Psi = \Psi(\underline{\underline{\epsilon}}) = \Psi(I_{\underline{\underline{\epsilon}}}, II_{\underline{\underline{\epsilon}}}, III_{\underline{\underline{\epsilon}}}) \,, \tag{2.34}$$

as the relationship between the potential function Ψ and the strain measure $\underline{\underline{\epsilon}}$ is independent of the material axes chosen [5]. The invariants of the strain measure $\underline{\underline{\epsilon}}$ can be calculated as

$$\begin{aligned} I_{\underline{\underline{\epsilon}}} &= \mathrm{tr}(\underline{\underline{\epsilon}}) \,, \\ II_{\underline{\underline{\epsilon}}} &= \underline{\underline{\epsilon}} : \underline{\underline{\epsilon}} \,, \\ III_{\underline{\underline{\epsilon}}} &= \det(\underline{\underline{\epsilon}}) \,, \end{aligned} \tag{2.35}$$

where $\mathrm{tr}(\underline{\underline{\epsilon}})$ refers to the trace, (:) to the double dot product operator and $\det(\underline{\underline{\epsilon}})$ to the determinant of the strain measure $\underline{\underline{\epsilon}}$.

The work conjugate strain measure for the second *Piola-Kirchhoff* stress tensor $\underline{\underline{S}}$ is the *Green-Lagrange* strain tensor $\underline{\underline{E}}$. The elastic potential is defined as a function of the invariants of the right *Cauchy-Green* tensor $\underline{\underline{C}}$[7] because the potential function must remain invariant, even if the current configuration undergoes a rigid body transformation [5]. This implies that Ψ is only a function of the stretch component $\underline{\underline{U}}$ (and independent of the rotation component $\underline{\underline{R}}$) of the deformation gradient $\underline{\underline{F}} = \underline{\underline{R}}\,\underline{\underline{U}}$, and $\underline{\underline{C}} = \underline{\underline{U}}^2$ [5]. Therefore, the elastic potential Ψ can be defined as a function of the invariants of $\underline{\underline{C}}$, i.e. $\Psi = \Psi(I_{\underline{\underline{C}}}, II_{\underline{\underline{C}}}, III_{\underline{\underline{C}}})$. According to Equation (2.31), the second *Piola-Kirchhoff* stress tensor $\underline{\underline{S}}$ is the first derivative of the elastic potential with respect to the *Green-Lagrange* strain tensor $\underline{\underline{E}}$. In terms of the invariants of $\underline{\underline{C}}$ it can be calculated as [5]

$$\underline{\underline{S}} = \frac{\partial \Psi}{\partial \underline{\underline{E}}} = 2\,\frac{\partial \Psi}{\partial \underline{\underline{C}}} = 2\,\frac{\partial \Psi}{\partial I_{\underline{\underline{C}}}} \cdot \frac{\partial I_{\underline{\underline{C}}}}{\partial \underline{\underline{C}}} + 2\,\frac{\partial \Psi}{\partial II_{\underline{\underline{C}}}} \cdot \frac{\partial II_{\underline{\underline{C}}}}{\partial \underline{\underline{C}}} + 2\,\frac{\partial \Psi}{\partial III_{\underline{\underline{C}}}} \cdot \frac{\partial III_{\underline{\underline{C}}}}{\partial \underline{\underline{C}}} \,. \tag{2.36}$$

St. Venant-Kirchhoff Material Model

The simplest approach to describe a hyperelastic material is the *St. Venant-Kirchhoff* material model. Basically it is only the transfer of the equations used for small deformations to the finite deformation range, cf. Section 2.4.2. Hence, the strain tensor $\underline{\underline{\varepsilon}}$ is replaced by the *Green-Lagrange* strain tensor $\underline{\underline{E}}$ and the *Cauchy* stress tensor $\underline{\underline{\sigma}}$ becomes the second *Piola-Kirchhoff* stress tensor $\underline{\underline{S}}$. The elastic potential is defined as [5]

$$\Psi = \frac{\lambda}{2}\,\mathrm{tr}(\underline{\underline{E}})^2 + \mu\,\underline{\underline{E}} : \underline{\underline{E}} \,, \tag{2.37}$$

where μ and λ are material parameters. Applying Equation (2.36), one obtains for the second *Piola-Kirchhoff* stress tensor [5]

$$\underline{\underline{S}} = \lambda\;\mathrm{tr}(\underline{\underline{E}})\;\underline{\underline{I}} + 2\,\mu\,\underline{\underline{E}} \,. \tag{2.38}$$

[7]For convenience, it holds true that $\underline{\underline{C}} = 2\underline{\underline{E}} + \underline{\underline{I}}$.

The material elasticity tensor then can be calculated as [5]

$$\underline{\underline{\underline{\underline{\mathbb{C}}}}} = \lambda \, \underline{\underline{I}} \otimes \underline{\underline{I}} + \mu \, (\underline{\underline{\underline{\underline{I}}}} + \underline{\underline{\underline{\underline{I}}}}^T) \,, \tag{2.39}$$

with the second-order and fourth-order identity tensors $\underline{\underline{I}}$ and $\underline{\underline{\underline{\underline{I}}}}$, respectively.

Compressible Neo-Hookean Material Model

Another common and rather simple model is the compressible *Neo-Hookean* material model. It exhibits characteristics that can be identified with material parameters found in linear elasticity [5]. The elastic potential as a function of the invariants of the right *Cauchy-Green* tensor $\underline{\underline{C}}$ reads [5]

$$\Psi = \frac{\mu}{2} \, (I_{\underline{\underline{C}}} - 3) - \mu \, \ln(\sqrt{III_{\underline{\underline{C}}}}) + \frac{\lambda}{2} \, \ln(\sqrt{III_{\underline{\underline{C}}}})^2 \,, \tag{2.40}$$

where μ and λ are again material constants and ln(.) refers to the natural logarithm. Applying Equation (2.36), the second *Piola-Kirchhoff* stress tensor $\underline{\underline{S}}$ becomes [5]

$$\underline{\underline{S}} = \mu \, (\underline{\underline{I}} - \underline{\underline{C}}^{-1}) + \lambda \, \ln(\sqrt{III_{\underline{\underline{C}}}}) \, \underline{\underline{C}}^{-1} \,. \tag{2.41}$$

The material elasticity tensor $\underline{\underline{\underline{\underline{\mathbb{C}}}}}$ then can be calculated as [5]

$$\underline{\underline{\underline{\underline{\mathbb{C}}}}} = \lambda \, \underline{\underline{C}}^{-1} \otimes \underline{\underline{C}}^{-1} + 2 \, (\mu - \lambda \, \ln(\sqrt{III_{\underline{\underline{C}}}})) \, \underline{\underline{\underline{\underline{\mathcal{I}}}}} \,, \tag{2.42}$$

with the fourth-order tensor $\underline{\underline{\underline{\underline{\mathcal{I}}}}}$ defined as [5]

$$\underline{\underline{\underline{\underline{\mathcal{I}}}}} = -\frac{\partial(\underline{\underline{C}}^{-1})}{\partial \underline{\underline{C}}} \,. \tag{2.43}$$

2.4.2. Generalized Hooke's Law for Small Deformations

The generalized form of *Hooke's law* reads

$$\underline{\underline{\sigma}} = \underline{\underline{\underline{\underline{\mathbb{C}}}}} : \underline{\underline{\varepsilon}} \,, \tag{2.44}$$

with the *Cauchy* stress tensor $\underline{\underline{\sigma}}$, the infinitesimal strain tensor $\underline{\underline{\varepsilon}}$ and the fourth-order elasticity tensor $\underline{\underline{\underline{\underline{\mathbb{C}}}}}$ with 81 coefficients called *elastic constants* [18]. For the isotropic case, the number of coefficients reduces to 21 nonzero elements, which can be expressed with two constants, namely

$$\underline{\underline{\underline{\underline{\mathbb{C}}}}} = \lambda \, \underline{\underline{I}} \otimes \underline{\underline{I}} + \mu \, (\underline{\underline{\underline{\underline{I}}}} + \underline{\underline{\underline{\underline{I}}}}^T) \,, \tag{2.45}$$

with λ and μ being the *Lamé parameters* that can be generally calculated as

$$\begin{aligned}\lambda &= \frac{\nu E}{(1+\nu)(1-2\nu)}\,,\\ \mu &= \frac{E}{2\,(1+\nu)}\end{aligned} \tag{2.46}$$

The term E refers to the *Young modulus* and ν to the *Poisson ratio.* For the two-dimensional plane stress case, λ has to be modified and reads

$$\lambda = \frac{\nu E}{(1+\nu)(1-\nu)}\,. \tag{2.47}$$

2.5. Newton-Raphson Method

The PDE in Equation (2.29) is generally nonlinear. There are nonlinear kinematics, and the constitutive equations introduced in Section 2.4.1 are assumed to be nonlinear as well. In order to solve nonlinear equations, one can use the *Newton-Raphson* method (NRM), which is first introduced for scalar equations.

Assuming a given nonlinear, differentiable scalar function $f(x)$, one is interested in calculating the root of this function. But instead of solving the nonlinear equation $f(x) = 0$, the linear equation

$$f(x_t) + (x_{t+1} - x_t)f'(x_t) = 0 \iff x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} \tag{2.48}$$

is iteratively solved for each iteration step t. Equation (2.48) is the result of a truncated Taylor series expansion and $f'(x_t) = \frac{df}{dx}\Big|_{x_t}$ refers to the first derivative of $f(x)$ evaluated at the point x_t. In order to find the root, one has to start with an initial guess x_0. This leads to the first so-called *residuum* $f(x_0)$, and the *tangent* in this point is calculated as $\frac{\partial f}{\partial x}\Big|_{x_0}$. As a result, the increment Δx_1 can be calculated and reads

$$\Delta x_1 = -\frac{f(x_0)}{f'(x_0)} \implies x_1 = x_0 + \Delta x_1\,, \tag{2.49}$$

where x_1 is the next evaluation point of the iterative process. Repeating Equation (2.49) for each iteration step t up to a desired convergence limit, which is a measure of the deviation from the "real" root, results in an approximated value of the root of the nonlinear function $f(x)$. Therefore, one applies

$$\Delta x_{t+1} = -\frac{f(x_t)}{f'(x_t)} \implies x_{t+1} = x_t + \Delta x_{t+1}\,. \tag{2.50}$$

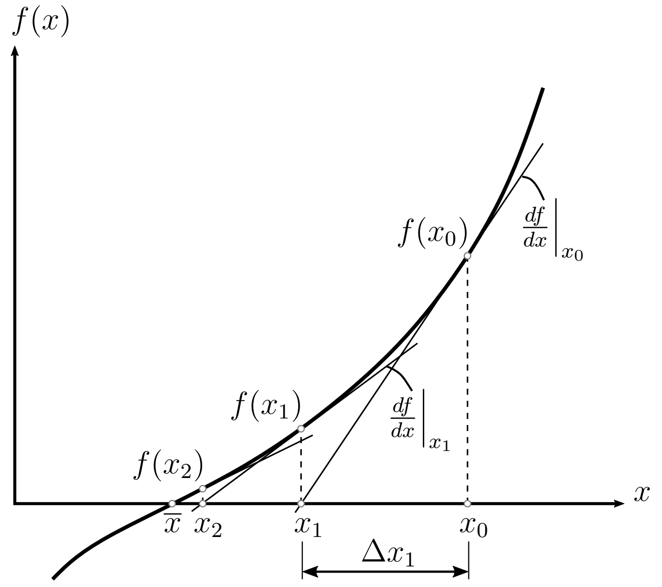


Figure 2.5.: Newton-Raphson Method for a Scalar Function $f(x)$

The iterative solution algorithm of the NRM is illustrated in Figure 2.5.

There is no limitation to scalar functions, but it is possible to apply the NRM for arbitrary functionals $\mathcal{F}(\underline{x})$ with arbitrary domain as well. Again, one is interested in the root, i.e. $\mathcal{F}(\underline{x}) = 0$. Hence, the nonlinear equation is linearised to

$$\mathcal{F}(\underline{x}) \approx \mathcal{F}(\underline{x}) + D\mathcal{F}(\underline{x})[\underline{\Delta x}] = 0\,, \tag{2.51}$$

where $D\mathcal{F}(\underline{x})[\underline{\Delta x}]$ refers to the directional derivative of $\mathcal{F}(\underline{x})$ in the direction of the increment vector $\underline{\Delta x}$. The directional derivative can be calculated as

$$D\mathcal{F}(\underline{x})[\underline{\Delta x}] = \frac{d}{ds}\bigg|_{s=0} \Big(\mathcal{F}(\underline{x} + s\underline{\Delta x})\Big)\,, \tag{2.52}$$

where s is a parameter to scale the increment $\underline{\Delta x}$. Equation (2.52) can be interpreted as follows: It describes the change[8] of the functional $\mathcal{F}(\underline{x})$ if the variable $\underline{x}$ is slightly altered[9]. However, the change must be evaluated at the point $\underline{x}$, so the parameter s is set to zero after the derivative has been performed.

While the functional $\mathcal{F}(\underline{x})$ is nonlinear in $\underline{x}$, the directional derivative $D\mathcal{F}(\underline{x})[\underline{\Delta x}]$ is linear in $\underline{\Delta x}$. Therefore, it can be said it has been linearised with respect to the increment $\underline{\Delta x}$ [5].

[8]The change is represented by the derivative with respect to s.
[9]This is expressed by the term $s\underline{\Delta x}$.

3. Finite Volume Method

This chapter introduces basics of the FVM, whereas the discretisation of the computational domain is presented first (Section 3.1), and then the discretisation of the transport equation for the general scalar variable ϕ is performed (Section 3.2). This derivation forms the basis for the development of the governing equations presented in Chapter 4. The common solution process and the so-called deferred correction approach are introduced in Section 3.3, illustrating the differences to the proposed implicit approach.

Due to simplicity in the presentation of sketches, the topics covered in this chapter are discussed for a two-dimensional domain only. Note that they also hold true for three-dimensional cases.

3.1. Mesh Generation

As mentioned in Chapter 1, the first step of the FVM is the generation of a discrete mesh comprising a collection of nonoverlapping cells. In general, each cell[1] is defined by a set of faces, while each face consists of a certain number of nodes and the nodes are defined by a position vector [37], cf. Figure 3.1.

Besides the distinction of the geometric shape[2] of the cells, one can generally distinguish between structured and unstructured grids. A structured mesh offers advantages regarding the topological information. In contrast, a major advantage of unstructured grids is their use for complex geometries. However, with the same regular distribution of points, the results obtained with an unstructured grid tend to be less accurate than with a structured grid [21]. While the cells in a structured grid can be both orthogonal and nonorthogonal, an unstructured grid generally comprises nonorthogonal cells.

3.1.1. Structured Grids

When the computational domain comprises a *structured* grid, each interior cell has the same number of neighbours [35]. Such a grid is created by dividing the domain into e.g. squares, cf. Figure 3.2. Each cell in the structured grid shown in Figure 3.2 is

[1] also referred to as *control volume*

[2] different types of polygons, e.g. triangles, quadrilaterals, tetrahedrals, hexahedrals, etc.

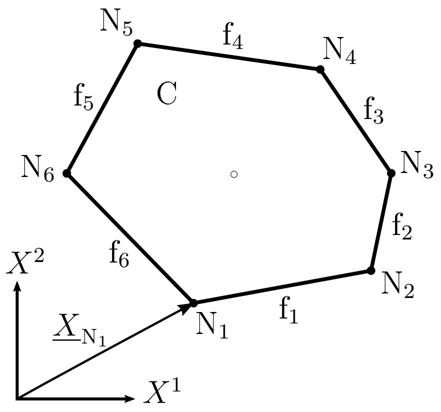


Figure 3.1.: Cell C Consisting of Faces f$_i$ and Nodes N$_i$

easily accessible because the *global indices* [35] i and j provide access to a cell via its respective coordinates. All the neighbours of the cell can be addressed by incrementing or decrementing the respective indices. Consequently, the topological information is embedded in the mesh structure of the grid [35].

However, with arbitrary geometries of the computational domain, a structured mesh reaches its limits.

3.1.2. Unstructured Grids

As mentioned above, *unstructured* grids are far more flexible for complex geometries [4, 21]. Although, this flexibility is accompanied by a disadvantage. In contrast to structured grids, a cell in an unstructured mesh cannot be accessed simply via global indices because the nodes are arbitrarily ordered [4, 35]. As a result, although the grid is not restricted by the geometry of the computational domain, the topological information must be created explicitly.

Due to the shape, an unstructured grid usually consists of nonorthogonal cells, cf. Figure 3.3. In general, the generation of an unstructured mesh is based on the *Delaunay triangulation* [4], which is not discussed further here.

Topological Information

Figure 3.3 comprises the same computational domain as Figure 3.2, but with nonorthogonal quadrilaterals. The global index of each cell is shown in the bottom left corner of the cell. Moreover, an arbitrary cell C and its neighbours F$_i$, where i is the local index of the neighbour cell, are shaded. In order to address face f$_1$, which is in-between C and F$_1$, the use of two indices is not sufficient. As mentioned above, topological information must be created explicitly, and therefore connectivities between

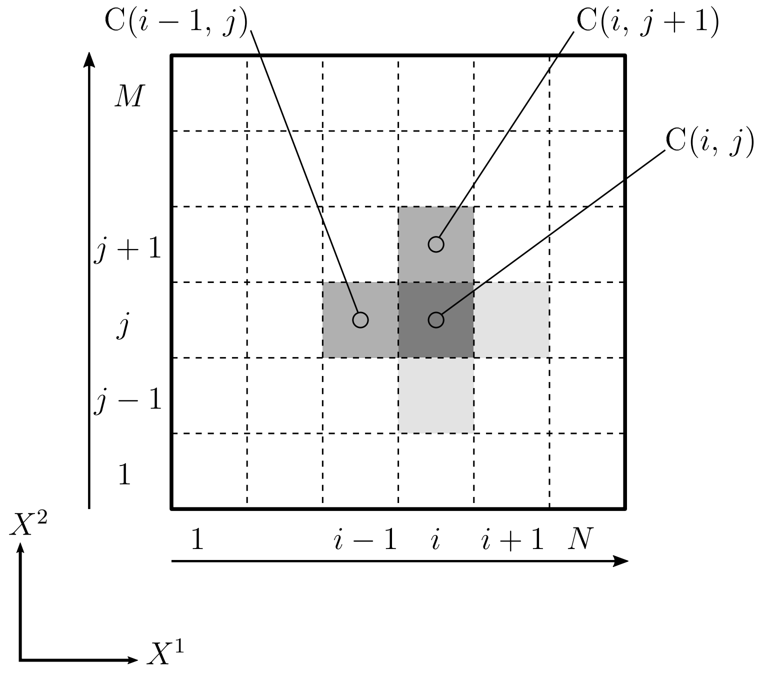


Figure 3.2.: Structured Mesh and its Topological Information (inspired by [35])

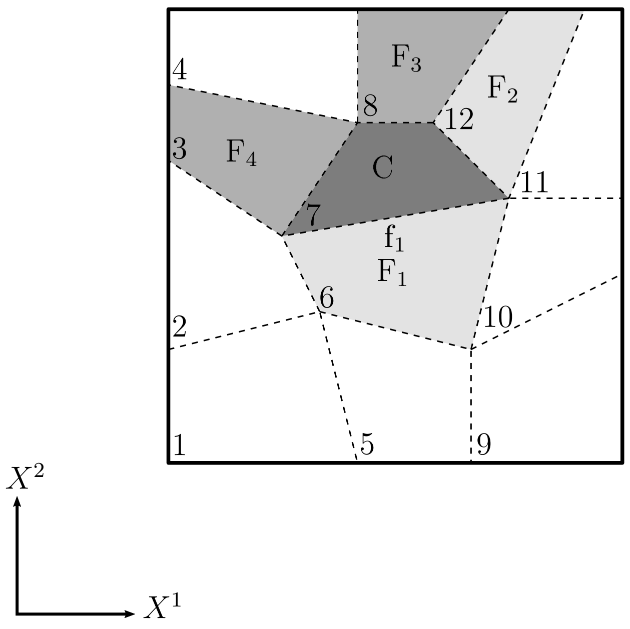


Figure 3.3.: Unstructured Mesh with Global and Local Indices

▷ cells,
▷ cells and faces,
▷ cells and nodes
▷ as well as between faces and nodes

are determined for each cell, face and node of the computational domain.

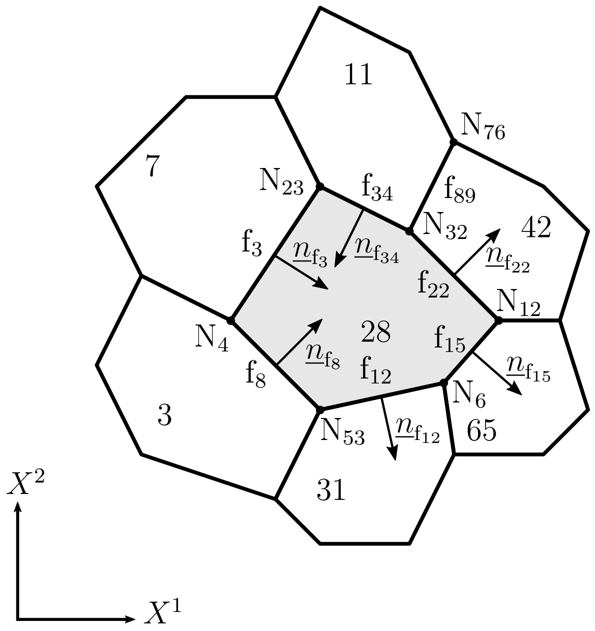


Figure 3.4.: Arbitrary Cell in an Unstructured Mesh (inspired by [35])

Figure 3.4 shows an unstructured mesh with topological information. Moreover, Table 3.1-Table 3.3 present corresponding connectivities for an arbitrary cell[3], face[4] and node[5] from Figure 3.4. The global indices of the elements are arbitrarily chosen and are for explanatory purposes only. Note that also for each other element in Figure 3.4 such information is determined to address all cells, faces and nodes in the unstructured, nonorthogonal grid properly.

As can be observed in the tables below, each cell possesses a specific number of (nearest) neighbours and is composed of a certain number of faces and nodes, cf. Table 3.1. In addition, a face consists of two nodes in the 2D case and generally[6] has an *owner* and *neighbour*. Usually, the owner is the cell with the smaller global index [35]. Moreover, the orientation of the normal vector is defined so that it points to the cell with the higher global index, i.e. to the neighbour cell, cf. Figure 3.4 and Table 3.2.

[3]cell with global index 28
[4]face with global index 8
[5]node with global index 32
[6]Boundary faces do not possess a neighbour.

Table 3.1.: Topological Information of Cell 28 from Figure 3.4

local indices	global indices		
	neighbour	face	node
1	3	3	4
2	7	8	6
3	11	12	12
4	31	15	23
5	42	22	32
6	65	34	53

Table 3.2.: Topological Information of Face f_8 from Figure 3.4

local indices	global indices		
	owner	neighbour	node
1	3	-	4
2	-	28	53

Table 3.3.: Topological Information of Node N_{32} from Figure 3.4

local indices	global indices		
	cell	face	node
1	11	22	12
2	28	34	23
3	42	89	76

Geometric Quantities

Geometric quantities of cells and faces are relevant for the discretisation of the governing equations. A general approach to calculate these quantities for arbitrary polygons in two dimensions is presented below.

For the 3D case, the discerning reader may have a look in any geometry textbook or in [4, 21, 35].

Computation of the Volume and Centroid of a Cell

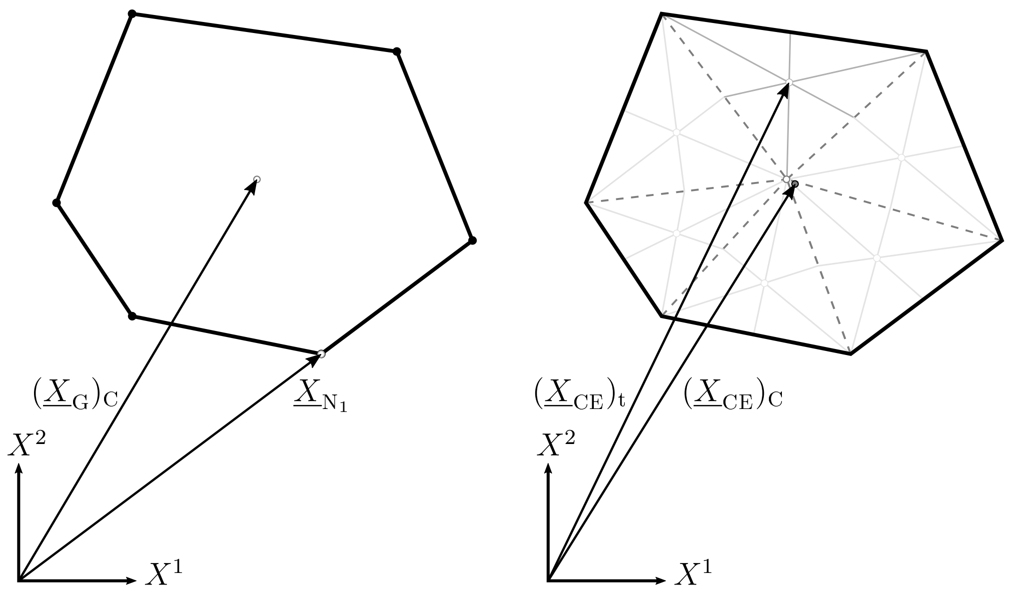

$(\underline{X}_{\mathrm{G}})_{\mathrm{C}}$
$\underline{X}_{\mathrm{N}_1}$
X^2
X^1
$(\underline{X}_{\mathrm{CE}})_{\mathrm{t}}$
$(\underline{X}_{\mathrm{CE}})_{\mathrm{C}}$
X^2
X^1

Figure 3.5.: Geometric Centre and Centroid of an Arbitrary Polygonal Cell (inspired by [35])

Figure 3.5 shows an arbitrary polygonal cell of a two-dimensional unstructured mesh. In general, for an arbitrary polygon the geometric centre $\underline{X}_{\mathrm{G}}$, which is the arithmetic average of all its surrounding points, does not coincide with its centroid $\underline{X}_{\mathrm{CE}}$. Only for some special cases this is fulfilled, e.g. for triangles. Nevertheless, to calculate the centroid, one needs to calculate the geometric centre of the cell $(\underline{X}_{\mathrm{G}})_{\mathrm{C}}$ which reads [35]

$$(\underline{X}_{\mathrm{G}})_{\mathrm{C}} = \frac{1}{n}\sum_{i=1}^{n} \underline{X}_{\mathrm{N}_i}\,, \tag{3.1}$$

with the sum over the number of vertices, i.e. nodes, n of the cell and $\underline{X}_{\mathrm{N}_i}$ referring to the position vectors of the nodes, cf. Figure 3.5 left. The subscript on the LHS, i.e. $(.)_{\mathrm{C}}$, refers to *cell*. In order to calculate the centroid of a cell C with k edges, one has to subdivide it into k subtriangles with the geometric centre $(\underline{X}_{\mathrm{G}})_{\mathrm{C}}$ being the apex of each triangle, cf. Figure 3.5 right. The centroid of the cell $(\underline{X}_{\mathrm{CE}})_{\mathrm{C}}$ can be calculated as [35]

$$(\underline{X}_{\mathrm{CE}})_{\mathrm{C}} = \frac{1}{A_{\mathrm{C}}} \sum_{\mathrm{t}} (\underline{X}_{\mathrm{CE}})_{\mathrm{t}} \; A_{\mathrm{t}} \,, \tag{3.2}$$

with the sum over the formed subtriangles t. $(\underline{X}_{\mathrm{CE}})_{\mathrm{t}}$ is the centroid of the respective subtriangle, which reads

$$(\underline{X}_{\mathrm{CE}})_{\mathrm{t}} = (\underline{X}_{\mathrm{G}})_{\mathrm{t}} = \frac{1}{n} \sum_{i=1}^{n} \underline{X}_{\mathrm{N}_i} \,, \tag{3.3}$$

with the sum over the number vertices n of the triangle and the subscript on the LHS, i.e. $(.)_{\mathrm{t}}$, referring to *triangle*. A_{t} is the area of the subtriangle, which can be calculated with e.g. Heron's formula:

$$A_{\mathrm{t}} = \frac{1}{4} \sqrt{(a+b+c) \cdot (a+b-c) \cdot (a-b+c) \cdot (-a+b+c)} \,, \tag{3.4}$$

where a, b and c relate to the length of the edges of the triangle. They may be calculated as

$$\begin{aligned} a &= \|\underline{X}_{\mathrm{N}_2} - \underline{X}_{\mathrm{N}_1}\| \,, \\ b &= \|\underline{X}_{\mathrm{N}_3} - \underline{X}_{\mathrm{N}_2}\| \,, \\ c &= \|\underline{X}_{\mathrm{N}_1} - \underline{X}_{\mathrm{N}_3}\| \,, \end{aligned} \tag{3.5}$$

with the position vectors of the nodes of the respective triangle. The volume of the cell A_{C}, which is indeed an area for the 2D case, can then be calculated with the sum of areas of the subtriangles t:

$$A_{\mathrm{C}} = \sum_{\mathrm{t}} A_{\mathrm{t}} \,. \tag{3.6}$$

Computation of the Area, Centroid and Normal Vector of a Face

In Figure 3.6, the arbitrary polygonal cell from Figure 3.5 is illustrated with the centroid and the normal vector of the face f. As mentioned above, the geometric centre and the centroid of a shape coincide in some special cases. This is also true for faces in the 2D case because they resemble straight lines. Therefore, the geometric centre of a face $(\underline{X}_{\mathrm{CE}})_{\mathrm{f}}$ reads [35]

$$(\underline{X}_{\mathrm{CE}})_{\mathrm{f}} = (\underline{X}_{\mathrm{G}})_{\mathrm{f}} = \frac{1}{n} \sum_{i=1}^{n} \underline{X}_{\mathrm{N}_i} \,, \tag{3.7}$$

with the sum over the nodes n of the respective face and the subscript on the LHS, i.e. $(.)_{\mathrm{f}}$, referring to *face*. The area of the face, which is a length for the 2D case, can be calculated as the norm of the distance vector $\underline{d}$ between its nodes, which reads

$$A_{\mathrm{f}} = \|\underline{d}_{12}\| = \|\underline{X}_{\mathrm{N2}} - \underline{X}_{\mathrm{N1}}\| \,, \tag{3.8}$$

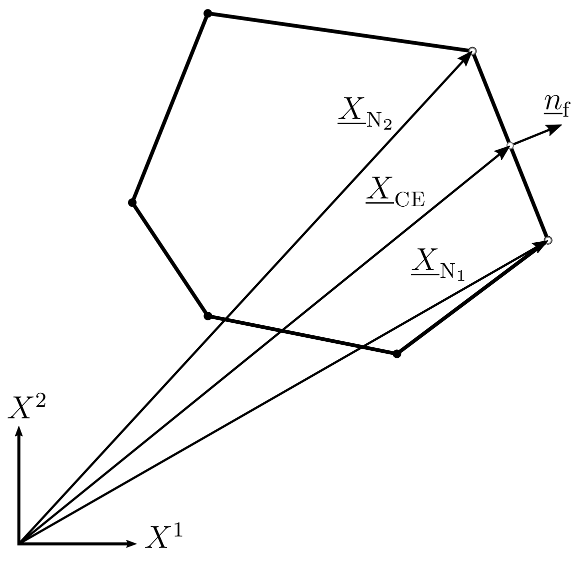

$\underline{X}_{\mathrm{N}_2}$
$\underline{n}_{\mathrm{f}}$
$\underline{X}_{\mathrm{CE}}$
$\underline{X}_{\mathrm{N}_1}$
X^2
X^1

Figure 3.6.: Centroid and Normal Vector of a Face of an Arbitrary Polygonal Cell (inspired by [35])

where $\underline{d}_{12}$ refers to distance vector between the nodes N_1 and N_2 (cf. Figure 3.6) with their position vectors $\underline{X}_{\mathrm{N}1}$ and $\underline{X}_{\mathrm{N}2}$, respectively.

With the distance vector, one can also calculate the normal vector of the face f by rotating the distance vector by 90 degrees and divide it by its norm. Note that for each face only *one*[7] normal vector is stored due to computational performance and memory efficiency [35].

3.2. Discretisation Process

Starting with the so-called transport equation (or conservation equation [16]) of the general scalar variable ϕ [35, 46], the discretisation is done only for the terms used in solid mechanics:

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{transient term}} + \underbrace{\underline{\nabla}\cdot(\rho\underline{v}\phi)}_{\text{convection term}} = \underbrace{\underline{\nabla}\cdot(\Gamma^{\phi}\underline{\nabla}\phi)}_{\text{diffusion term}} + \underbrace{Q^{\phi}}_{\text{source term}} \quad , \tag{3.9}$$

where the first term on the LHS is the time-dependent transient term[8] with the time variable t and the second one is the convection term with the density ρ and velocity vector $\underline{v}$. The first term on the RHS is the diffusion term, where Γ^{ϕ} refers to an arbitrary diffusion coefficient and the second term is the source term.

[7]Although it appears twice—in the topology of its owner *and* neighbour cell, which do not represent the same element.

[8]It is also called rate of change [46]

For a steady-state problem, one can neglect the transient term. Moreover, the convection term is not relevant for solids. Hence, Equation (3.9) is simplified to

$$\underline{\nabla} \cdot (\Gamma^{\phi} \underline{\nabla} \phi) + Q^{\phi} = 0 \,. \tag{3.10}$$

If Equation (3.10) is integrated over the volume V_{C} of the respective cell, one obtains [35]

$$\int\limits_{V_{\mathrm{C}}} (\underline{\nabla} \cdot (\Gamma^{\phi} \underline{\nabla} \phi) + Q^{\phi}) \,\mathrm{d}V = \int\limits_{V_{\mathrm{C}}} \underline{\nabla} \cdot (\Gamma^{\phi} \underline{\nabla} \phi) \,\mathrm{d}V + \int\limits_{V_{\mathrm{C}}} Q^{\phi} \,\mathrm{d}V = 0 \,. \tag{3.11}$$

Replacing the volume integral of the diffusion term with a surface integral applying *Gauss'* divergence theorem, Equation (3.11) becomes [46]

$$\int\limits_{\partial V_{\mathrm{C}}} (\Gamma^{\phi} \underline{\nabla} \phi) \cdot \mathrm{d}\underline{A} + \int\limits_{V_{\mathrm{C}}} Q^{\phi} \,\mathrm{d}V = 0 \,, \tag{3.12}$$

where ∂V_{C} refers to the boundary of the cell, and $\mathrm{d}\underline{A} = \underline{n} \,\mathrm{d}A$ is the differential surface vector of a face with normal vector $\underline{n}$ and differential area $\mathrm{d}A$.

3.2.1. Discretisation of the Diffusion Flux Term

The diffusion term in Equation (3.12) is denoted as *diffusion flux* [35] and reads

$$\underline{J}^{\phi,D} = \Gamma^{\phi} \underline{\nabla} \phi \,, \tag{3.13}$$

where the superscript D refers to *diffusion*. Due to the discretisation of the computational domain, the surface integral in Equation (3.12) can be replaced by a sum of integrals over the area of each face [16, 35]:

$$\int\limits_{\partial V_{\mathrm{C}}} \underline{J}^{\phi,D} \cdot \mathrm{d}\underline{A} = \sum_{\mathrm{f}} \int\limits_{A_{\mathrm{f}}} \underline{J}^{\phi,D} \cdot \mathrm{d}\underline{A} \,, \tag{3.14}$$

where f refers to the faces of the cell. For the final step of discretisation, the surface integral of each face must be replaced by a quadrature. Using a *Gaussian* quadrature, the integral part of Equation (3.14) becomes

$$\int\limits_{A_{\mathrm{f}}} \underline{J}^{\phi,D} \cdot \mathrm{d}\underline{A} = \int\limits_{A_{\mathrm{f}}} (\underline{J}^{\phi,D} \cdot \underline{n}) \,\mathrm{d}A = \sum_{i_p} (\underline{J}^{\phi,D} \cdot \underline{n})_{i_p} \, w_{i_p} \, A_{\mathrm{f}} \,, \tag{3.15}$$

with the sum over the integration points i_p. The term w_{i_p} refers to the weight at the respective integration point and A_{f} to the area of the face. It should be pointed out that the more integration points are used, the higher the order of accuracy. Although, the computational effort rises [35] as well. Therefore, in this thesis only one integration point is used for the quadrature. This refers to a mean value integration and is second-order accurate [35].

If Equation (3.15) is substituted in Equation (3.14), the discretised form of the diffusion flux finally reads

$$\sum_{\mathrm{f}}\Big(\sum_{i_p}(\underline{J}^{\phi,D}\cdot\underline{n})_{i_p}\ w_{i_p}\ A_{\mathrm{f}}\Big)_{\mathrm{f}}=\sum_{\mathrm{f}}(\underline{J}^{\phi,D}\cdot\underline{n})_f\ A_{\mathrm{f}}\,. \tag{3.16}$$

3.2.2. Discretisation of the Source Term

For the discretisation of the source term in Equation (3.12), also a *Gaussian* quadrature is applied [16, 35]:

$$\int\limits_{V_{\mathrm{C}}} Q^{\phi}\,\mathrm{d}V=\sum_{i_p}(Q^{\phi})_{i_p}\ w_{i_p}\ V_{\mathrm{C}}\,. \tag{3.17}$$

Again, only one integration point is used for the quadrature, which yields the final discretised form of the source term:

$$\sum_{i_p}(Q^{\phi})_{i_p}\ w_{i_p}\ V_{\mathrm{C}}=Q^{\phi}\ V_{\mathrm{C}}\,. \tag{3.18}$$

3.2.3. Spatial Discretisation and Gradient Computation

If Equation (3.15) and Equation (3.18) are combined, one obtains

$$\sum_{\mathrm{f}}(\Gamma^{\phi}\underline{\nabla}\phi\cdot\underline{n})_f\ A_{\mathrm{f}}+Q^{\phi}\ V_{\mathrm{C}}=0\,, \tag{3.19}$$

where all variables are either known or are a function, i.e. the gradient, of the general scalar variable ϕ.

For the variable ϕ, a linear variation in space is assumed, i.e. $\phi=\phi(\underline{X})$. Therefore, $\phi(\underline{X})$ can be calculated as [35]

$$\phi(\underline{X})=\phi(\underline{X}_{\mathrm{C}})+(\underline{X}-\underline{X}_{\mathrm{C}})\cdot(\underline{\nabla}\phi)_{\mathrm{C}}\,, \tag{3.20}$$

where $\underline{X}$ and $\underline{X}_{\mathrm{C}}$ are position vectors, and $\underline{X}_{\mathrm{C}}$ refers to the centroid of the respective cell. $\phi(\underline{X}_{\mathrm{C}})$ relates to the evaluated quantity in the centroid and $(\underline{\nabla}\phi)_{\mathrm{C}}$ to the evaluated gradient in this point.

The gradient terms $(\underline{\nabla}\phi)_{\mathrm{f}}$ in Equation (3.19) and $(\underline{\nabla}\phi)_{\mathrm{C}}$ in Equation (3.20) require a special treatment, which is developed below.

Least-Squares Method and Cell Gradient

In this thesis, each cell gradient $(\underline{\nabla}\phi)_\mathrm{C}$ is computed using the least-squares (LS) method. The calculation of the gradient with the LS method offers more flexibility with regard to the order of accuracy [1]. The advantage of flexibility can only be fully exploited by the use of appropriate weighting functions [35].

Starting with a cell C with its neighbours F_i in an unstructured grid (cf. Figure 3.7), the value of the general scalar variable ϕ_{F_i} in the centroid of a neighbour F_i can be calculated with Equation (3.20) as [4]

$$\phi_{\mathrm{F}_i} = \phi(\underline{X}_{\mathrm{F}_i}) = \phi_\mathrm{C} + (\underline{X}_{\mathrm{F}_i} - \underline{X}_\mathrm{C}) \cdot (\underline{\nabla}\phi)_\mathrm{C} \,. \tag{3.21}$$

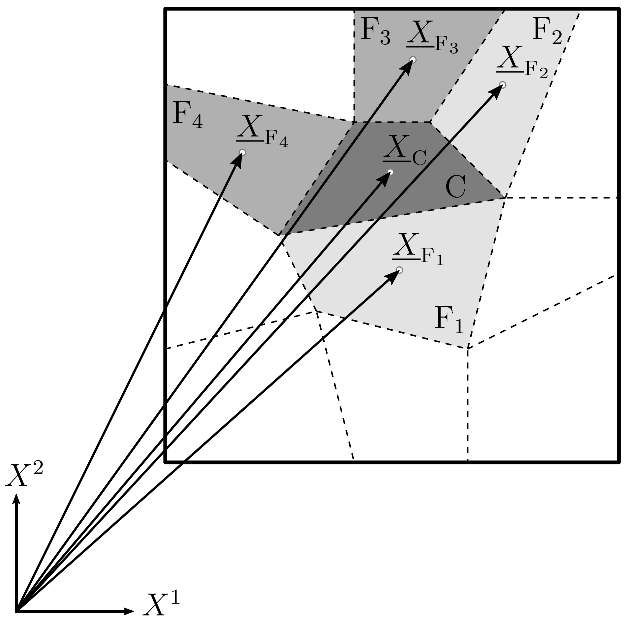

F$_3$
$\underline{X}_{\mathrm{F}_3}$
F$_2$
$\underline{X}_{\mathrm{F}_2}$
F$_4$
$\underline{X}_{\mathrm{F}_4}$
$\underline{X}_\mathrm{C}$
C
$\underline{X}_{\mathrm{F}_1}$
F$_1$
X^2
X^1

Figure 3.7.: Unstructured Grid with Position Vectors to Cell C and its Neighbours F_i

If this is done for each neighbour F_i of the cell C and one rearranges Equation (3.21), one obtains a matrix equation in the form of

$$\begin{pmatrix} X_{\mathrm{CF}_1} & Y_{\mathrm{CF}_1} & Z_{\mathrm{CF}_1} \\ X_{\mathrm{CF}_2} & Y_{\mathrm{CF}_2} & Z_{\mathrm{CF}_2} \\ \vdots & \vdots & \vdots \\ X_{\mathrm{CF}_n} & Y_{\mathrm{CF}_n} & Z_{\mathrm{CF}_n} \end{pmatrix} \begin{pmatrix} (\frac{\partial\phi}{\partial x})_\mathrm{C} \\ (\frac{\partial\phi}{\partial y})_\mathrm{C} \\ (\frac{\partial\phi}{\partial z})_\mathrm{C} \end{pmatrix} = \begin{pmatrix} \phi_{F_1} - \phi_\mathrm{C} \\ \phi_{F_2} - \phi_\mathrm{C} \\ \vdots \\ \phi_{F_n} - \phi_\mathrm{C} \end{pmatrix} , \tag{3.22}$$

where the subscript n relates to the number of neighbours. X_{CF_i}, Y_{CF_i} and Z_{CF_i} refer to the coordinates of the distance vector $\underline{d}_{\mathrm{CF}_i}$. The distance vector between an owner cell C and its neighbour F_i can be calculated with $\underline{d}_{\mathrm{CF}_i} = \underline{X}_{\mathrm{F}_i} - \underline{X}_\mathrm{C}$, $\underline{X}_\mathrm{C}$ and $\underline{X}_{\mathrm{F}_i}$

being the position vectors of owner and neighbour, respectively. In general, the matrix in Equation (3.22) is not square, therefore an exact solution cannot be obtained. There will always be an error for the solution of the system of equations. The LS method minimises the sum of the errors squared, whereas the solution of the cell gradient $(\underline{\nabla}\phi)_{\mathrm{C}}$ can be calculated with [35]

$$(\underline{\nabla}\phi)_{\mathrm{C}} = (\underline{\underline{d}}^T\ \underline{\underline{d}})^{-1}\ \underline{\underline{d}}^T\ \underline{\phi}_{\mathrm{CF}}\,, \tag{3.23}$$

where $\underline{\underline{d}}$ refers to the *matrix of distance vectors* between cell and neighbours, i.e.

$$\underline{\underline{d}} = \begin{pmatrix} X_{\mathrm{CF}_1} & Y_{\mathrm{CF}_1} & Z_{\mathrm{CF}_1} \\ X_{\mathrm{CF}_2} & Y_{\mathrm{CF}_2} & Z_{\mathrm{CF}_2} \\ \vdots & \vdots & \vdots \\ X_{\mathrm{CF}_n} & Y_{\mathrm{CF}_n} & Z_{\mathrm{CF}_n} \end{pmatrix}\,. \tag{3.24}$$

The vector $\underline{\phi}_{\mathrm{CF}}$ relates to the *vector of differences* of the general scalar variable ϕ between cell and neighbours calculated as

$$\underline{\phi}_{\mathrm{CF}} = \begin{pmatrix} \phi_{F_1} - \phi_{\mathrm{C}} \\ \phi_{F_2} - \phi_{\mathrm{C}} \\ \vdots \\ \phi_{F_n} - \phi_{\mathrm{C}} \end{pmatrix}\,. \tag{3.25}$$

As mentioned above, including a *weighting function* increases the flexibility of the LS method. The weighting $\underline{\underline{w}}$ is a diagonal matrix of form

$$\underline{\underline{w}} = \begin{pmatrix} \frac{1}{\|\underline{d}_1\|} & 0 & \cdots & 0 \\ 0 & \frac{1}{\|\underline{d}_2\|} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{\|\underline{d}_n\|} \end{pmatrix}\,, \tag{3.26}$$

with the nonzero elements $\frac{1}{\|\underline{d}_i\|}$, where i relates to the local index of the neighbour. Equation (3.23) then becomes [35]

$$(\underline{\nabla}\phi)_{\mathrm{C}} = (\underline{\underline{d}}^T\ \underline{\underline{w}}^T\ \underline{\underline{w}}\ \underline{\underline{d}})^{-1}\underline{\underline{d}}^T\ \underline{\underline{w}}^T\ \underline{\underline{w}}\ \underline{\phi}_{\mathrm{CF}}\,. \tag{3.27}$$

The weighting function $\underline{\underline{w}}$ considers the aspect ratio or size of the cell due to the reciprocal norm of the distance vector.

Equation (3.27) relates to the *explicit* form of the gradient. An *implicit* form, which is necessary for the implicit governing equations, is introduced in Section 4.2.1 and derived for a generic example in Section 5.2.2.

Geometric Weighting Factor and Face Gradient

In general, the gradient at the face can be interpolated linearly with [16, 35]

$$(\underline{\nabla}\phi)_\mathrm{f} = g_\mathrm{C}\ (\underline{\nabla}\phi)_\mathrm{C} + (1 - g_\mathrm{C})(\underline{\nabla}\phi)_\mathrm{F}\,, \tag{3.28}$$

where $(\underline{\nabla}\phi)_\mathrm{C}$ and $(\underline{\nabla}\phi)_\mathrm{F}$ are the cell gradients of the owner C and neighbour F calculated with Equation (3.27) and g_C is the *geometric weighting factor* [35]. In the literature there are different approaches to determine g_C, especially in combination with unstructured grids and the *Green-Gauss* gradient. As the proposed equations in this thesis are treated in a different manner, two options for the geometric weighting factor g_C were chosen.

Option 1: The influence of each cell gradient for the face gradient is related to the distance between the cell centroid and the face centroid. Moreover, this approach considers the aspect ratio or size of a cell. The geometric weighting factor g_C can then be calculated as

$$g_\mathrm{C} = \frac{\|\underline{X}_\mathrm{F} - \underline{X}_\mathrm{f}\|}{\|\underline{X}_\mathrm{F} - \underline{X}_\mathrm{f}\| + \|\underline{X}_\mathrm{f} - \underline{X}_\mathrm{C}\|}\,, \tag{3.29}$$

where $\underline{X}_i$ refers either to the position vector of the face centroid ($i = \mathrm{f}$), the centroid of the neighbour cell ($i = \mathrm{F}$) or the centroid of the owner cell (index $i = \mathrm{C}$). This approach is denoted as *distance* option.

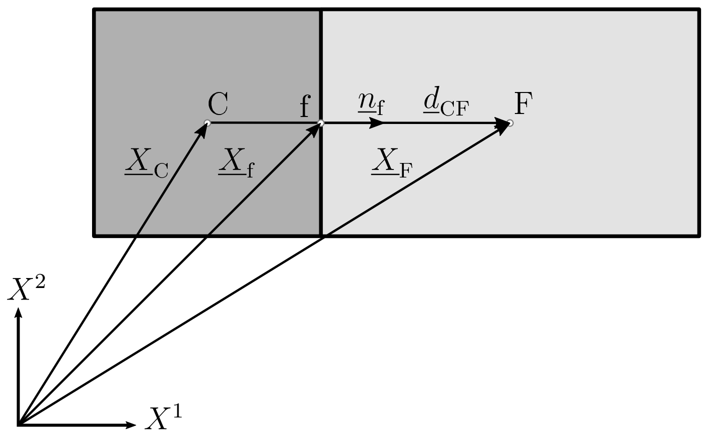

C
f
$\underline{n}_\mathrm{f}$
$\underline{d}_\mathrm{CF}$
F
$\underline{X}_\mathrm{C}$
$\underline{X}_\mathrm{f}$
$\underline{X}_\mathrm{F}$
X^2
X^1

Figure 3.8.: Conjunctional Face in a Two-Dimensional Grid (inspired by [35])

The reason why the distance vector $\underline{d}_\mathrm{CF}$ in Equation (3.29) is split into two separate parts is due to unstructured grids, cf. Figure 3.9. As can be observed, the face centroid and the intersection point of the segment $\overline{\mathrm{CF}}$ do not coincide. Hence, in order to take into account the aspect ratio or size of the cell, the geometric weighting factor is calculated as described in Equation (3.29). Moreover, the following option, which is proposed by different authors [4, 35], may be used as well.

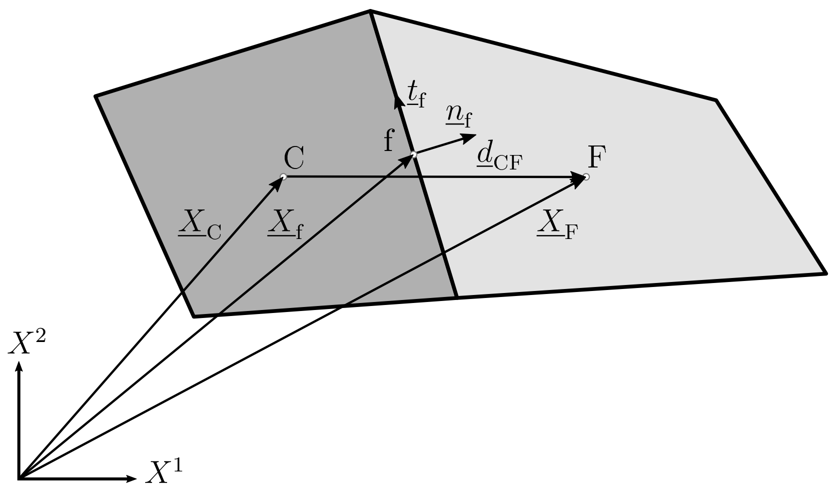

$\underline{t}_\mathrm{f}$
$\underline{n}_\mathrm{f}$
f
C
$\underline{d}_\mathrm{CF}$
F
$\underline{X}_\mathrm{C}$
$\underline{X}_\mathrm{f}$
$\underline{X}_\mathrm{F}$
X^2
X^1

Figure 3.9.: Nonconjunctional Face in a Two-Dimensional Grid (inspired by [35])

Option 2: The face gradient is the arithmetic average of the cell gradients of owner and neighbour. The geometric weighting factor then becomes

$$g_\mathrm{C} = \frac{1}{2}\,. \tag{3.30}$$

Equation (3.29) yields the same values as Equation (3.30) for orthogonal structured grids with uniform cell size. Moreover, this approach, which is denoted as *average* option, can be also linked to the nonorthogonal treatment of unstructured grids [35].

Treatment of Nonorthogonality

As illustrated in Figure 3.9, in an unstructured grid the distance vector $\underline{d}_\mathrm{CF}$ and the normal vector $\underline{n}_\mathrm{f}$ are not parallel. There is a nonorthogonal contribution due to the skewness of the mesh. This nonorthogonal contribution affects the diffusion term and subsequently the solution process.

Applying Equation (3.28) to calculate the face gradient is not sufficient. The treatment of nonorthogonality is well covered in the literature and three different approaches are presented below [33, 35]:

Approach 1: This approach was employed by *Muzaferija* [37] and developed by *Caughey and Jameson* [9]. The nonorthogonal correction is held as small as possible and therefore is called *minimum correction* approach. The corrected face gradient, denoted with a superscript asterisk, can be calculated as [33, 35]:

$$(\underline{\nabla}\phi)^*_\mathrm{f} = (\underline{\nabla}\phi)_\mathrm{f} - \frac{(\underline{\nabla}\phi)_\mathrm{f}\cdot\underline{d}_\mathrm{CF}}{\|\underline{d}_\mathrm{CF}\|^2}\underline{d}_\mathrm{CF} + \frac{\phi_\mathrm{F}-\phi_\mathrm{C}}{\|\underline{d}_\mathrm{CF}\|^2}\underline{d}_\mathrm{CF}\,, \tag{3.31}$$

where $(\underline{\nabla}\phi)_\mathrm{f}$ is the interpolated face gradient of Equation (3.28) and $\underline{d}_\mathrm{CF}$ refers to the distance vector between the cell and its neighbour.

Another treatment of the nonorthogonality was heavily discussed by *Jasak* [24]:

Approach 2: In the *over-relaxed* approach, the nonorthogonal correction is increased with the skewness of the mesh. The corrected face gradient reads [33]

$$(\underline{\nabla}\phi)^*_\mathrm{f} = (\underline{\nabla}\phi)_\mathrm{f} - \frac{(\underline{\nabla}\phi)_\mathrm{f} \cdot \underline{d}_\mathrm{CF}}{\|\underline{d}_\mathrm{CF} \cdot \underline{A}_\mathrm{f}\|}\underline{A}_\mathrm{f} + \frac{\phi_\mathrm{F} - \phi_\mathrm{C}}{\|\underline{d}_\mathrm{CF} \cdot \underline{A}_\mathrm{f}\|}\underline{A}_\mathrm{f}\,, \tag{3.32}$$

where $\underline{A}_\mathrm{f} = \underline{n}_\mathrm{f}\ A_\mathrm{f}$ refers to the surface vector of the face.

The last approach dealt with in this thesis is a modification of the correction approach above by *Demirdzic and Muzaferija* [13]:

Approach 3: The corrected face gradient of the *modified* approach can be calculated as follows [33]:

$$(\underline{\nabla}\phi)^*_\mathrm{f} = (\underline{\nabla}\phi)_\mathrm{f} - \frac{(\overline{\underline{\nabla}\phi})_\mathrm{f} \cdot \underline{d}_\mathrm{CF}}{\|\underline{d}_\mathrm{CF}\|\|\underline{A}_\mathrm{f}\|}\underline{A}_\mathrm{f} + \frac{\phi_\mathrm{F} - \phi_\mathrm{C}}{\|\underline{d}_\mathrm{CF}\|\|\underline{A}_\mathrm{f}\|}\underline{A}_\mathrm{f}\,, \tag{3.33}$$

where the face gradient with the overbar denotes an arithmetic average, i.e. $(\overline{\underline{\nabla}\phi})_\mathrm{f} = \frac{1}{2}\Big((\underline{\nabla}\phi)_\mathrm{C} + (\underline{\nabla}\phi)_\mathrm{F}\Big)$.

It should be pointed out that the nonorthogonal treatment for the proposed equations is also executed completely implicit. The derivation of the terms is explained for a generic example in Section 5.2.3. The general approach, including the distinction between orthogonal and nonorthogonal terms and thus the division into implicit and explicit terms, is discussed below.

3.3. Deferred Correction Approach and Solution

The discretised equations read

$$\sum_\mathrm{f} (\Gamma^\phi \underline{\nabla}\phi \cdot \underline{n})_\mathrm{f}\ A_\mathrm{f} + Q^\phi\ V_\mathrm{C} = 0\,. \tag{3.34}$$

In order to solve for the primary variable ϕ, Equation (3.34) must be transformed into a system of algebraic equations of the form

$$\underline{\underline{A}}\ \underline{\phi} = \underline{b}\,, \tag{3.35}$$

which is not discussed in detail here. In general, the source term in Equation (3.34) contributes to the RHS and the diffusion term to both sides, as described below. In a direct solution approach, ϕ can be calculated as $\phi = \underline{\underline{A}}^{-1}\ \underline{b}$, with $\underline{\underline{A}}^{-1}$ being the inverse of the matrix. In contrast, the standard approach for FV approaches is an iterative segregated solution approach[9], where the matrix $\underline{\underline{A}}$ is not in a block coupled form, but a vector of scalar equations that are solved in a segregated manner [35].

Another big difference to the proposed implicit approach is the *deferred correction approach*, which is common practice both for FV solvers for CFD and solid mechanics applications. The deferred correction approach is introduced briefly, but for further details the discerning reader may have a look in [16, 28, 35].

In order to transform Equation (3.34) into Equation (3.35) using the deferred correction approach, the face gradient is treated in a deferred manner—it is split into two parts, namely [35]

$$(\underline{\nabla}\phi)_\mathrm{f} \cdot \underline{A}_\mathrm{f} = \underbrace{(\underline{\nabla}\phi)_\mathrm{f} \cdot \underline{E}_\mathrm{f}}_{\text{orthogonal contribution}} + \underbrace{(\underline{\nabla}\phi)_\mathrm{f} \cdot \underline{T}_\mathrm{f}}_{\text{nonorthogonal contribution}}, \tag{3.36}$$

where $\underline{A}_\mathrm{f} = \underline{E}_\mathrm{f} + \underline{T}_\mathrm{f}$ refers to the surface vector of a face, which is the sum of an orthogonal and nonorthogonal contribution.

In general, the face gradient of the orthogonal contribution in Equation (3.36) is evaluated *implicitly*, and the face gradient of the nonorthogonal contribution is evaluated *explicitly*, using the latest available values of ϕ, i.e. values from the previous iteration step, in the segregated solution procedure. The implicit gradient term can be calculated as [35]

$$(\underline{\nabla}\phi)_\mathrm{f} \cdot \underline{E}_\mathrm{f} = E_\mathrm{f} \, \frac{\phi_\mathrm{F} - \phi_\mathrm{C}}{d_\mathrm{CF}}, \tag{3.37}$$

since it is the orthogonal contribution. E_f refers to the area of the face, d_CF is the distance between owner and neighbour, and the terms in the numerator in Equation (3.37) are the primary variables to be solved for owner C and neighbour F.

Moreover, the terms of the nonorthogonal treatment are then also split into implicit and explicit contributions which is not discussed further here. As a result, so-called *cross-diffusion* [35] terms comprising the nonorthogonal contributions are then added as source term to the RHS of the equation system.

In order to close the system of linear equations, one has to prescribe boundary conditions. The treatment of them in the deferred correction approach is discussed below.

Dirichlet Boundary Conditions

If the value of the general scalar variable ϕ is prescribed at the boundary, it is called a Dirichlet boundary condition (BC) (*value specified* BC, cf. Figure 3.10). As mentioned above, if the computational domain consists of a nonorthogonal mesh, the cross-diffusion

[9]e.g. for the pressure-velocity coupled SIMPLE approach

terms must be taken into account. This is also true for Dirichlet BCs. Therefore, if the values of ϕ are specified at the boundary, they read (cf. Equation (3.16))

$$(\underline{J}^{\phi,D} \cdot \underline{n})_{\mathrm{b}}\, A_{\mathrm{b}} = (\Gamma^{\phi}\underline{\nabla}\phi)_{\mathrm{b}} \cdot\ \underline{A}_{\mathrm{b}} = (\Gamma^{\phi}\underline{\nabla}\phi)_{\mathrm{b}} \cdot\ (\underline{E}_{\mathrm{b}} + \underline{T}_{\mathrm{b}})\,, \tag{3.38}$$

where the index b relates to the boundary face. The orthogonal contribution $(\underline{\nabla}\phi)_{\mathrm{b}} \cdot \underline{E}_{\mathrm{b}}$ is evaluated implicitly[10] and the nonorthogonal contribution $(\underline{\nabla}\phi)_{\mathrm{b}} \cdot\ \underline{T}_{\mathrm{b}}$ explicitly in a deferred correction manner as mentioned above.

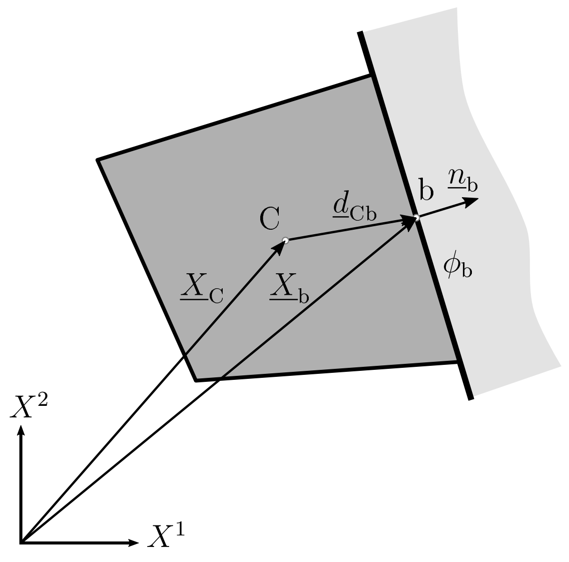

C
$\underline{d}_{\mathrm{Cb}}$
b
$\underline{n}_{\mathrm{b}}$
ϕ_{b}
$\underline{X}_{\mathrm{C}}$
$\underline{X}_{\mathrm{b}}$
X^2
X^1

Figure 3.10.: Dirichlet BC for a Nonorthogonal Cell (inspired by [35])

Neumann Boundary Conditions

The Neumann boundary condition refers to as *flux specified* BC, cf. Figure 3.11. If the flux is prescribed at the boundary, it can be written as [35]

$$-\,(\Gamma^{\phi}\underline{\nabla}\phi)_{\mathrm{b}} \cdot\ \underline{n}_{\mathrm{b}} = q_{\mathrm{b}}\,, \tag{3.39}$$

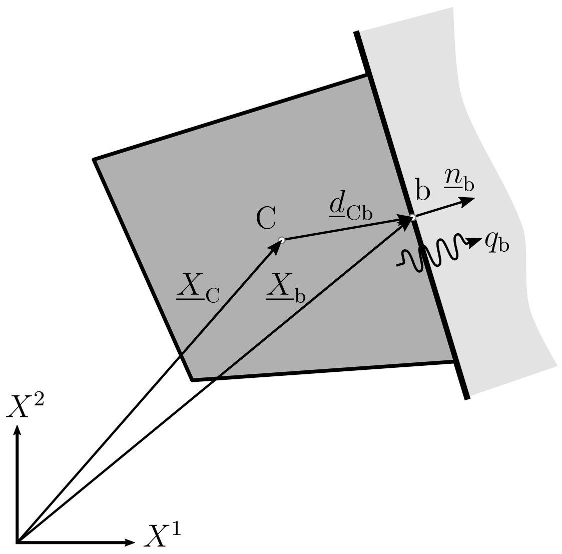
where q_{b} relates to the specified flux. Equation (3.39) resembles the diffusion flux term for this boundary face, since (cf. Equation (3.16))

$$(\underline{J}^{\phi,D} \cdot \underline{n})_{\mathrm{b}}\, A_{\mathrm{b}} = (\Gamma^{\phi}\underline{\nabla}\phi)_{\mathrm{b}} \cdot\ \underline{n}_{\mathrm{b}} A_{\mathrm{b}} = -q_{\mathrm{b}}\, A_{\mathrm{b}}\,. \tag{3.40}$$

Therefore, the term $-q_{\mathrm{b}}\, A_{\mathrm{b}}$, where A_{b} refers to the area of the boundary face, is treated as source term on the RHS of the equations system.

Thus the system of algebraic equations is assembled cell by cell and solved in an iterative segregated manner.

[10] Analogous to Equation (3.37), except that f and F are substituted with b.



b
$\underline{n}_\mathrm{b}$
C
$\underline{d}_\mathrm{Cb}$
q_b
$\underline{X}_\mathrm{C}$
$\underline{X}_\mathrm{b}$
X^2
X^1

Figure 3.11.: Neumann BC for a Nonorthogonal Cell (inspired by [35])

4. Implicit Finite Volume Method for Continuum Mechanics

In this chapter, the proposed implicit equations in total *Lagrangian* formulation are developed. At first (Section 4.1), the equations introduced in Section 2.3 are linearised using the directional derivative approach described in Section 2.5. After that (Section 4.2), the terms are discretised according to Section 3.2 and thus yield the discretised implicit governing equations in total *Lagrangian* formulation. The chapter concludes with the assembly to a set of linear algebraic equations and the solving process.

4.1. Linearised Governing Equations

The governing equations in integral form to calculate the deformation read, cf. Equation (2.29):

$$\int\limits_{\partial\mathcal{B}_0} \underline{\underline{F}}\,\underline{\underline{S}}\,\underline{N}\,\mathrm{d}A = \underline{0}\,,\ \ \underline{X} \in \mathcal{B}_0\,. \tag{4.1}$$

In order to close the equation system, BCs are prescribed:

$$\begin{aligned} \underline{\underline{\sigma}} \cdot \underline{n} &= \underline{t}\,, & \underline{x} &\in \partial_N\mathcal{B}_t \\ \chi_t(\underline{X}) &= \overline{\chi}_t(\underline{X})\,, & \underline{X} &\in \partial_D\mathcal{B}_0\,. \end{aligned} \tag{4.2}$$

In general, the equilibrium of internal forces[1] in Equation (4.1) is nonlinear for both the geometry and material law, hence the equations must be linearised. The equations in Equation (4.1) are usually first linearised and then discretised. This is the common approach in solid mechanics, although some authors prefer the contrary way [5].

The linearisation is carried out for a given deformation mapping χ. The increment $\Delta\chi_t$[2] is formed using the vector field of incremental displacements $\underline{\Delta u}(\underline{x})$[3] in $\mathcal{B}_t$, which are compatible with the Dirichlet BCs, i.e. $\underline{\Delta u}\,|_{\partial_D\mathcal{B}_t} \equiv \underline{0}$. A curve $\chi(s)$ similar to Equation (2.52) is used in order to determine the directional derivative [22]. It reads

$$\chi(s) = \chi + s\underline{\Delta u}(\underline{x}) \tag{4.3}$$

[1]It can also be identified as diffusion flux term with regard to the notation in the FVM.
[2]Δ relates to *increment* and is not a *Laplace operator*.
[3]see Footnote 2

and it holds true that

$$\chi(s,\underline{X}) = \chi(\underline{X}) + s\underline{\Delta u}\,(\chi(\underline{X})) \iff \underline{x}(s,\underline{X}) = \underline{x}(\underline{X}) + s\underline{\Delta u}(\underline{X})\,. \tag{4.4}$$

In addition, it holds

$$\chi(s=0,\underline{X}) = \chi(\underline{X})\,, \qquad D\chi(s=0,\underline{X}) = \underline{\Delta u}(\underline{X})\,. \tag{4.5}$$

The coherences in Equation (4.4) and Equation (4.5) are used for the linearisation executed in Section 4.1.1 and Section 4.1.2.

In the following, the term of internal forces in Equation (4.1) will be referred to as *diffusion flux* term. It is denoted in the style of *Moukalled et al.* [35] as

$$\underline{J}^{\underline{u},D} := \int\limits_{\partial\mathcal{B}_0} \underline{\underline{F}}\,\underline{\underline{S}}\,\underline{N}\,\mathrm{d}A\,, \tag{4.6}$$

where $\underline{J}^{\underline{u},D}$ stands for the diffusion flux ($\underline{J}$ and superscript D) due to the displacement vector (superscript $\underline{u}$).

The nonlinear equation system

$$\underline{J}^{\underline{u},D} = \underline{0} \tag{4.7}$$

is transformed into the linearised equation system

$$\underline{J}^{\underline{u},D}(\chi_t) \approx \underline{J}^{\underline{u},D}(\chi_t) + D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}] = \underline{0}\,, \tag{4.8}$$

where $D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]$ refers to the linearised diffusion flux due to the displacement of the mapping χ_t in the direction of the incremental displacement vector $\underline{\Delta u}$. The equation system in Equation (4.8) is solved for $\underline{\Delta u}$ in an iterative solution procedure.

4.1.1. Linearisation of Kinematics, Strain and Stress

In order to obtain the linearisation of the diffusion flux term, some correlations regarding kinematics, strain and stress are derived in advance. For the linearisation of the deformation gradient (Equation (2.5)), one obtains [5, 22]

$$\begin{aligned} D\underline{\underline{F}}(\chi_t)[\underline{\Delta u}] &= \left.\frac{d}{ds}\right|_{s=0} \left(\underline{\underline{F}}(\chi_t + s\underline{\Delta u})\right) = \\ &= \left.\frac{d}{ds}\right|_{s=0} \left(\frac{\partial\underline{x}}{\partial\underline{X}} + s\frac{\partial\underline{\Delta u}}{\partial\underline{X}}\right) = \frac{\partial\underline{\Delta u}}{\partial\underline{X}} =: \underline{\nabla}^0\underline{\Delta u}\,, \end{aligned} \tag{4.9}$$

where the directional derivative of the incremental displacement vector $\underline{\Delta u}$ with respect to $\underline{X}$ is defined as the *material gradient* $\underline{\nabla}^0\underline{\Delta u}$.

With the product rule of differentiation and the substitution of Equation (4.9), one can calculate the linearisation of the *Green-Lagrange* strain tensor (Equation (2.11)) [5, 22]:

$$
\begin{aligned}
D\underline{\underline{E}}(\chi_t)[\underline{\Delta u}] &= \frac{1}{2}\, D\left(\underline{\underline{F}}^T\ \underline{\underline{F}} - \underline{\underline{I}}\right)(\chi_t)[\underline{\Delta u}] = \\
&= \frac{1}{2}\left(D\underline{\underline{F}}^T(\chi_t)[\underline{\Delta u}]\ \underline{\underline{F}} + \underline{\underline{F}}^T\ D\underline{\underline{F}}(\chi_t)[\underline{\Delta u}]\right) = \\
&= \frac{1}{2}\left((\nabla^0 \underline{\Delta u})^T\ \underline{\underline{F}} + \underline{\underline{F}}^T\ \nabla^0 \underline{\Delta u}\right) .
\end{aligned}
\tag{4.10}
$$

Furthermore, the linearisation of the second *Piola-Kirchhoff* stress tensor (Equation (2.20)) can be evaluated applying the chain rule of differentiation as follows [5]:

$$
\begin{aligned}
D\underline{\underline{S}}(\chi_t)[\underline{\Delta u}] &= D\underline{\underline{S}}\left(\underline{\underline{E}}(\chi_t)\right)\left[D\underline{\underline{E}}(\chi_t)[\underline{\Delta u}]\right] = \\
&= \underline{\underline{\underline{\underline{\mathbb{C}}}}} : D\underline{\underline{E}}(\chi_t)[\underline{\Delta u}] .
\end{aligned}
\tag{4.11}
$$

4.1.2. Linearisation of the Diffusion Flux

The linearisation of the diffusion flux term is performed as explained above:

$$
\begin{aligned}
D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}] &= D\left(\int\limits_{\partial\mathcal{B}_0} \left(\underline{\underline{F}}\ \underline{\underline{S}}\ \underline{N}\,\mathrm{d}A\right)(\chi_t)\right)[\underline{\Delta u}] = \\
&= \int\limits_{\partial\mathcal{B}_0} \left(D\underline{\underline{F}}(\chi_t)[\underline{\Delta u}]\ \underline{\underline{S}}\ + \underline{\underline{F}}\ D\underline{\underline{S}}(\chi_t)[\underline{\Delta u}]\right)\ \underline{N}\,\mathrm{d}A ,
\end{aligned}
\tag{4.12}
$$

where in the second line of Equation (4.12) the product rule of differentiation was applied. Substituting Equation (4.9), Equation (4.10) and Equation (4.11) into Equation (4.12) yields

$$
\begin{aligned}
&\int\limits_{\partial\mathcal{B}_0} \left(D\underline{\underline{F}}(\chi_t)[\underline{\Delta u}]\ \underline{\underline{S}}\ + \underline{\underline{F}}\ D\underline{\underline{S}}(\chi_t)[\underline{\Delta u}]\right)\ \underline{N}\,\mathrm{d}A = \\
= &\int\limits_{\partial\mathcal{B}_0} \left(\nabla^0 \underline{\Delta u}\ \underline{\underline{S}}\ + \underline{\underline{F}}\ \underline{\underline{\underline{\underline{\mathbb{C}}}}} : D\underline{\underline{E}}(\chi_t)[\underline{\Delta u}]\right)\ \underline{N}\,\mathrm{d}A = \\
= &\int\limits_{\partial\mathcal{B}_0} \left(\nabla^0 \underline{\Delta u}\ \underline{\underline{S}}\ + \underline{\underline{F}}\ \underline{\underline{\underline{\underline{\mathbb{C}}}}} : \frac{1}{2}\left(D\underline{\underline{F}}^T(\chi_t)[\underline{\Delta u}]\ \underline{\underline{F}} + \underline{\underline{F}}^T\ D\underline{\underline{F}}(\chi_t)[\underline{\Delta u}]\right)\right)\ \underline{N}\,\mathrm{d}A .
\end{aligned}
\tag{4.13}
$$

Substituting again Equation (4.9) for the linearisation of the deformation gradient yields

$$\begin{aligned}&\int\limits_{\partial\mathcal{B}_0}\left(\underline{\nabla}^0\underline{\Delta u}\;\underline{\underline{S}}\;+\underline{\underline{F}}\;\underline{\underline{\underline{\underline{\mathbb{C}}}}}:\frac{1}{2}\left(D\underline{\underline{F}}^T(\chi_t)[\underline{\Delta u}]\;\underline{\underline{F}}+\underline{\underline{F}}^T\;D\underline{\underline{F}}(\chi_t)[\underline{\Delta u}]\right)\right)\;\underline{N}\,\mathrm{d}A=\\&=\int\limits_{\partial\mathcal{B}_0}\left(\underline{\nabla}^0\underline{\Delta u}\;\underline{\underline{S}}\;+\underline{\underline{F}}\;\underline{\underline{\underline{\underline{\mathbb{C}}}}}:\frac{1}{2}\left(\left(\underline{\nabla}^0\underline{\Delta u}\right)^T\;\underline{\underline{F}}+\underline{\underline{F}}^T\;\underline{\nabla}^0\underline{\Delta u}\right)\right)\;\underline{N}\,\mathrm{d}A\,.\end{aligned}\tag{4.14}$$

With the symmetry $\mathbb{C}^{IJKL}=\mathbb{C}^{IJLK}$ of the material elasticity tensor $\underline{\underline{\underline{\underline{\mathbb{C}}}}}$, one can determine the coherence

$$\underline{\underline{\underline{\underline{\mathbb{C}}}}}:\left(\underline{\nabla}^0\underline{\Delta u}\right)^T\;\underline{\underline{F}}=\underline{\underline{\underline{\underline{\mathbb{C}}}}}:\underline{\underline{F}}^T\;\underline{\nabla}^0\underline{\Delta u}\,.\tag{4.15}$$

Substituting Equation (4.15) in Equation (4.14) yields the final form of the linearised diffusion flux term:

$$\begin{aligned}&\int\limits_{\partial\mathcal{B}_0}\left(\underline{\nabla}^0\underline{\Delta u}\;\underline{\underline{S}}\;+\underline{\underline{F}}\;\underline{\underline{\underline{\underline{\mathbb{C}}}}}:\frac{1}{2}\left(\left(\underline{\nabla}^0\underline{\Delta u}\right)^T\;\underline{\underline{F}}+\underline{\underline{F}}^T\;\underline{\nabla}^0\underline{\Delta u}\right)\right)\;\underline{N}\,\mathrm{d}A=\\&=\int\limits_{\partial\mathcal{B}_0}\left(\underline{\nabla}^0\underline{\Delta u}\;\underline{\underline{S}}\;+\underline{\underline{F}}\;\underline{\underline{\underline{\underline{\mathbb{C}}}}}:\underline{\underline{F}}^T\;\underline{\nabla}^0\underline{\Delta u}\right)\;\underline{N}\,\mathrm{d}A=D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\,.\end{aligned}\tag{4.16}$$

As may be observed, Equation (4.16) is now a linear function of the material gradient of the incremental displacement vector $\underline{\nabla}^0\underline{\Delta u}$.

4.2. Discretised Governing Equations

4.2.1. Discretised Equations for Large Deformations

After the spatial discretisation, the computational domain consists of a finite number of cells. As a result, Equation (4.6) and Equation (4.16) must hold true for a single cell:

$$(\underline{J}^{\underline{u},D}(\chi_t))_{\mathrm{C}}=\int\limits_{\partial V_{\mathrm{C}}}\underline{\underline{F}}\;\underline{\underline{S}}\;\underline{N}\,\mathrm{d}\underline{A}\,,\tag{4.17}$$

$$(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}])_{\mathrm{C}}=\int\limits_{\partial V_{\mathrm{C}}}\left(\underline{\nabla}^0\underline{\Delta u}\;\underline{\underline{S}}\;+\underline{\underline{F}}\;\underline{\underline{\underline{\underline{\mathbb{C}}}}}:\underline{\underline{F}}^T\;\underline{\nabla}^0\underline{\Delta u}\right)\;\underline{N}\,\mathrm{d}\underline{A}\,,\tag{4.18}$$

where ∂V_{C} refers to the boundary of the cell. The discretisation is executed according to Section 3.2. The discretised formulation of Equation (4.17) then reads

$$\int\limits_{\partial V_{\mathrm{C}}}\underline{\underline{F}}\;\underline{\underline{S}}\;\underline{N}\,\mathrm{d}\underline{A}\approx\sum_{\mathrm{f}}\left(\underline{\underline{F}}\;\underline{\underline{S}}\;\underline{N}\right)_{\mathrm{f}}A_{\mathrm{f}}\,,\tag{4.19}$$

and for Equation (4.18), one obtains

$$\begin{aligned}
&\int\limits_{\partial V_\mathrm{C}} \left(\underline{\nabla}^0 \underline{\Delta u}\ \underline{\underline{S}}\ + \underline{\underline{F}}\ \underline{\underline{\underline{\underline{\mathbb{C}}}}} : \underline{\underline{F}}^T\ \underline{\nabla}^0 \underline{\Delta u} \right)\ \underline{N}\, \mathrm{d}\underline{A} \approx \\
&\approx \sum_\mathrm{f} \left(\left(\underline{\nabla}^0 \underline{\Delta u}\ \underline{\underline{S}}\ + \underline{\underline{F}}\ \underline{\underline{\underline{\underline{\mathbb{C}}}}} : \underline{\underline{F}}^T\ \underline{\nabla}^0 \underline{\Delta u} \right)\ \underline{N} \right)_\mathrm{f} A_\mathrm{f}\,,
\end{aligned} \tag{4.20}$$

with the sum over the faces f in the reference configuration of a cell C and the subscript, i.e. $(.)_\mathrm{f}$, also referring to it. A_f refers to the area of the respective face.

Equation (4.19) reads in index notation

$$\sum_\mathrm{f} \left(F^i_J\ S^{JK}\ N_K \right)_\mathrm{f} A_\mathrm{f}\,. \tag{4.21}$$

The deformation gradient F^i_J is calculated using the *explicit* gradient of displacements and reads

$$(F^i_J)_\mathrm{f} = \delta^i_J + ({}^\mathrm{ex}u^i_{,J})_\mathrm{f} \tag{4.22}$$

where the superscript ${}^\mathrm{ex}(.)$ refers to $\underline{\mathrm{ex}}$plicit. The displacement gradient for a *cell* is calculated with the LS method:

$$({}^\mathrm{ex}u^i_{,J})_\mathrm{C} = (G^K_J)_\mathrm{C}\ (\hat{u}^i_K)_\mathrm{CF}\,, \tag{4.23}$$

with the *geometry matrix* G^K_J ($J = 1 : dim$[4]; $K = 1 :$ `non`[5]) and the *matrix of displacement differences* $(\hat{u}^i_K)_\mathrm{CF}$ ($i = 1 : dim$) between owner C and neighbours F. As $(\hat{u}^i_K)_\mathrm{CF}$[6] only has a meaning in connection with *cells*, the subscript $(.)_\mathrm{C}$ is not used for it. G^K_J reads, cf. Equation (3.27):

$$G^K_J = H^M_J\ W^K_M\,, \tag{4.24}$$

where $H^M_J = \check{B}^L_J\ d^M_L$ ($L = 1 : dim$; $M = 1 :$ `non`) and $\check{B}^L_J$ relates to the components of $\underline{\underline{B}}^{-1}$, which is the inverse of $\underline{\underline{B}}$. The components of $\underline{\underline{B}}$ read $B^L_J = d^O_J\ W^P_O\ g^{LR}\ g_{PS}\ d^S_R$[7] ($R = 1 : dim$; $O, P, S = 1 :$ `non`) and $W^K_M = w^Q_M\ w^K_Q$ ($Q = 1 :$ `non`). d^M_L refers to the components of the matrix of distance vectors between cell and neighbours (cf. Equation (3.24)) and w^Q_M relates to the components of the weighting function (cf. Equation (3.26)) as introduced in Section 3.2.3. The explicit face gradient in Equation (4.22) is then calculated applying Equation (3.28):

$$({}^\mathrm{ex}u^i_{,J})_\mathrm{f} = g_\mathrm{C}\ ({}^\mathrm{ex}u^i_{,J})_\mathrm{C} + (1 - g_\mathrm{C})({}^\mathrm{ex}u^i_{,J})_\mathrm{F}\,. \tag{4.25}$$

[4] dim = dimension; i.e. $dim = 2$ or $dim = 3$

[5] `non` = number of neighbours; it includes the number of neighbours F_i of the cell C

[6] The subscript $(.)_\mathrm{CF}$ refers to cell C and neighbours F.

[7] g^{LR} and g_{PS} refer to contra- and covariant metric tensors, respectively.

The face gradient $({}^{\text{ex}}u^i_{,J})_\text{f}$ in Equation (4.25) is *corrected* with one of the nonorthogonal treatment approaches as introduced in Section 3.2.3. The derivation of the terms above is discussed for a generic example in Section 5.2.2 and Section 5.2.3.

Equation (4.20) reads in index notation

$$\sum_\text{f} \Big(\big({}^{\text{im}}\Delta u^i_{,J} \; S^{JK} + F^i_J \; \mathbb{C}^{JKLM} \; g_{no} \; F^n_L \; {}^{\text{im}}\Delta u^o_{,M} \big) \; N_K \Big)_\text{f} A_\text{f} \,, \tag{4.26}$$

where the superscript ${}^{\text{im}}(.)$ refers to <u>im</u>plicit. The implicit form of the gradient is evaluated via *coefficient matrices* which are derived from the geometry matrices. It then reads

$$\big({}^{\text{im}}\Delta u^i_{,J} \big)_\text{C} = (T^A_J)_\text{C} \; \Delta\hat{u}^i_A \,, \tag{4.27}$$

where T^A_J ($J = 1 : dim$; $A = 1 :$ `nov`[8]) refers to the (global) coefficient matrix. The term $\Delta\hat{u}^i_A$ ($i = 1 : dim$) refers to the incremental displacement values of the cell centroids of the computational domain. As $\Delta\hat{u}^i_A$ only has meaning in connection with *cells*, the subscript $(.)_\text{C}$ is not used for it. The implicit face gradient in Equation (4.26) is then calculated applying Equation (3.28):

$$\big({}^{\text{im}}\Delta u^i_{,J} \big)_\text{f} = \underbrace{\big(g_\text{C} \; (T^A_J)_\text{C} + (1 - g_\text{C})(T^A_J)_\text{F} \big)}_{:=(T^A_J)_\text{f}} \Delta\hat{u}^i_A \,. \tag{4.28}$$

The face gradient coefficient matrix $(T^A_J)_\text{f}$ in Equation (4.28) is *corrected* with one of the nonorthogonal treatment approaches as introduced in Section 3.2.3. The derivation of the terms above is discussed in detail in Section 5.2.2 and Section 5.2.3.

If Equation (4.28) is substituted in Equation (4.26), one obtains

$$\begin{aligned} &\sum_\text{f} \Big(\big(T^A_J \; \Delta\hat{u}^i_A \; S^{JK} + F^i_J \; \mathbb{C}^{JKLM} \; g_{no} \; F^n_L \; T^A_M \; \Delta\hat{u}^o_A \big) \; N_K \Big)_\text{f} A_\text{f} = \\ &\sum_\text{f} \Big(\big(T^A_J \; S^{JK} \; \delta^i_p + F^i_J \; \mathbb{C}^{JKLM} \; g_{no} \; \delta^o_p \; F^n_L \; T^A_M \big) \; N_K \Big)_\text{f} A_\text{f} \; \Delta\hat{u}^p_A \,. \end{aligned} \tag{4.29}$$

In the style of *Bonet and Wood* [5], an *initial stress component* [5] can be detected in the first term in Equation (4.29):

$$\underbrace{\sum_\text{f} \big(T^A_J \; S^{JK} \; \delta^i_p \; N_K \big)_\text{f} A_\text{f} \, \Delta\hat{u}^p_A}_{:=(K^{iA}_p)^{\text{ISC}}} \,, \tag{4.30}$$

[8] `nov` = <u>n</u>umber <u>o</u>f <u>v</u>ariables; it includes the number of cells and boundary faces with prescribed Dirichlet BCs value $\neq 0$

where $(K_p^{iA})^{\mathrm{ISC}}$ refers to the components of the tensor, and the superscript $(.)^{\mathrm{ISC}}$ refers to *initial stress component*, since it contains the second *Piola-Kirchhoff* stress tensor $\underline{\underline{S}}$. The second term in Equation (4.29) reveals a *constitutive component* [5]:

$$\underbrace{\sum_{\mathrm{f}} \left(F_J^i \, \mathbb{C}^{JKLM} \, g_{no} \, \delta_p^o \, F_L^n \, T_M^A \, N_K\right)_{\mathrm{f}} A_{\mathrm{f}}}_{:=(K_p^{iA})^{\mathrm{CC}}} \Delta \hat{u}_A^p \,, \tag{4.31}$$

where $(K_p^{iA})^{\mathrm{CC}}$ refers to the components of the tensor, and the superscript $(.)^{\mathrm{CC}}$ refers to *constitutive component*, since it contains the constitutive law.

If Equation (4.21) and Equation (4.29) are substituted in Equation (4.8), one obtains the discretised governing equations for large deformations in total *Lagrangian* formulation:

$$\begin{aligned} &\sum_{\mathrm{f}} \left(F_J^i \, S^{JK} \, N_K\right)_{\mathrm{f}} A_{\mathrm{f}} + \\ +&\sum_{\mathrm{f}} \left(\left(T_J^A \, S^{JK} \, \delta_p^i + F_J^i \, \mathbb{C}^{JKLM} \, g_{no} \, \delta_p^o \, F_L^n \, T_M^A\right) \, N_K\right)_{\mathrm{f}} A_{\mathrm{f}} \, \Delta \hat{u}_A^p = 0 \,. \end{aligned} \tag{4.32}$$

4.2.2. Discretised Equations for Small Deformations

For the small strain theory, the governing equations read, cf. Equation (2.27):

$$\int_{\mathcal{B}_t} \underline{\nabla} \cdot \underline{\underline{\sigma}} \, \mathrm{d}v = \int_{\mathcal{B}_0} \underline{\nabla} \cdot \underline{\underline{\sigma}} \, \mathrm{d}V = \underline{0} \,. \tag{4.33}$$

Due to the small deformations there is no need for a pull-back operation. A linearisation due to the linear coherences (linear kinematics and material law) is also not necessary. The discretisation then can be executed straightforward, applying the divergence theorem of *Gauss* and the findings of Section 3.2. The discretised equations read

$$\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{C}} = \int_{V_{\mathrm{C}}} \underline{\nabla} \cdot \underline{\underline{\sigma}} \, \mathrm{d}V = \int_{\partial V_{\mathrm{C}}} \underline{\underline{\sigma}} \, \underline{N} \, \mathrm{d}\underline{A} \approx \sum_{\mathrm{f}} \left(\underline{\underline{\sigma}} \, \underline{N}\right)_{\mathrm{f}} A_{\mathrm{f}} = \underline{0} \,. \tag{4.34}$$

With *Hooke's* law for small deformations, one obtains:

$$\sum_{\mathrm{f}} \left(\underline{\underline{\sigma}} \, \underline{N}\right)_{\mathrm{f}} A_{\mathrm{f}} = \sum_{\mathrm{f}} \left(\underline{\underline{\underline{\underline{\mathbb{C}}}}} \, : \underline{\underline{\varepsilon}} \, \underline{N}\right)_{\mathrm{f}} A_{\mathrm{f}} = \underline{0} \,, \tag{4.35}$$

which is equivalent to its form in index notation[9]:

$$\sum_{\mathrm{f}} \left(\mathbb{C}^{ijkl} \; {}^{\mathrm{im}}\varepsilon_{kl} \, N_j\right)_{\mathrm{f}} A_{\mathrm{f}} = 0 \,, \tag{4.36}$$

[9]for the indices holds true: $i, j, k, l = 1 : dim$

where the superscript ${}^{\mathrm{im}}(.)$ indicates that the infinitesimal strain tensor $\underline{\underline{\varepsilon}}$ is calculated with the implicit form of the displacement gradient. The infinitesimal strain can then be described as, cf. Equation (2.16):

$$
{}^{\mathrm{im}}\varepsilon_{kl} = \frac{1}{2}\left({}^{\mathrm{im}}u_{k,l} + {}^{\mathrm{im}}u_{l,k}\right) = \frac{1}{2}\left(T_k^A\ \delta_{lp} + T_l^A\ \delta_{kp}\right)\hat{u}_A^p\,. \tag{4.37}
$$

The term $\hat{u}_A^p$ ($p = 1 : dim$; $A = 1 :$ `nov`) refers to the displacement values of the cell centroids of the computational domain. If Equation (4.37) is substituted in Equation (4.36), one obtains the final discretised equations for the small deformation theory:

$$
\sum_{\mathrm{f}} \left(\mathbb{C}^{ijkl}\ \frac{1}{2}\left(T_k^A\ \delta_{lp} + T_l^A\ \delta_{kp}\right)\ N_j\right)_{\mathrm{f}} A_{\mathrm{f}}\ \hat{u}_A^p = 0\,. \tag{4.38}
$$

4.3. Treatment of BCs and Solution Process

The solution process is only explained for finite deformations, but is executed for linear elasticity analogous to the description below. The only difference is the omission of the NR procedure. In order to close the equation system, BCs are prescribed:

Neumann Boundary Conditions

For some test cases in Chapter 6, Neumann BCs are prescribed. For this thesis, only the prescription of *dead loads*[10] was investigated, cf. Figure 4.1. A dead load can be prescribed as follows:

$$
(\underline{\underline{P}}\ \underline{N})_{\mathrm{b}} = \underline{T}_{\mathrm{b}}\,, \tag{4.39}
$$

where $\underline{T}_{\mathrm{b}}$ refers to the specified dead load traction vector as force per unit area. Equation (4.39) resembles the diffusion flux term (cf. Equation (3.16)) for this boundary face, since

$$
\int_{\partial\mathcal{B}_0} \underline{\underline{F}}\ \underline{\underline{S}}\ \underline{N}\,\mathrm{d}A = \int_{\partial\mathcal{B}_0} \underline{\underline{P}}\ \underline{N}\,\mathrm{d}A \approx (\underline{\underline{P}}\ \underline{N})_{\mathrm{b}} A_{\mathrm{b}} = \underline{T}_{\mathrm{b}} A_{\mathrm{b}}\,. \tag{4.40}
$$

Therefore, the term $\underline{T}_{\mathrm{b}} A_{\mathrm{b}}$, where A_{b} refers to the area of the boundary face, is treated as source term on the RHS of the system of equations.

Dirichlet Boundary Conditions

If the value of the displacement vector $\underline{u}$ is prescribed at the boundary, it is called a Dirichlet BC, cf. Figure 4.2. The integration of Dirichlet BCs into the resulting system of linear equations is described below. How the terms are calculated is discussed in detail for a generic example in Section 5.2.5.

[10] i.e. a constant force (per unit area), which neither changes magnitude nor direction

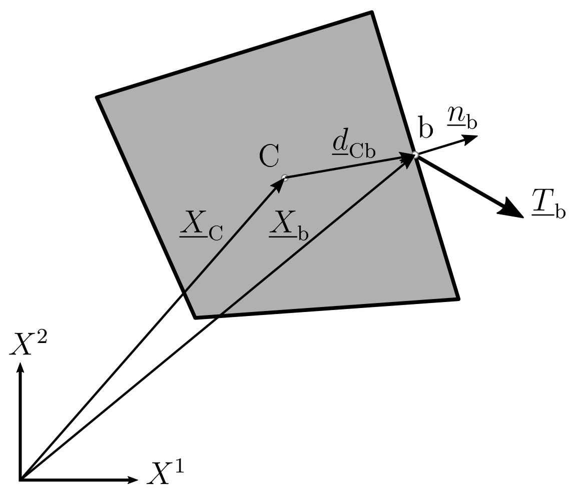

$\underline{n}_\mathrm{b}$
b
$\underline{d}_\mathrm{Cb}$
C
$\underline{T}_\mathrm{b}$
$\underline{X}_\mathrm{C}$
$\underline{X}_\mathrm{b}$
X^2
X^1

Figure 4.1.: Neumann BC (Dead Load) for a Nonorthogonal Cell

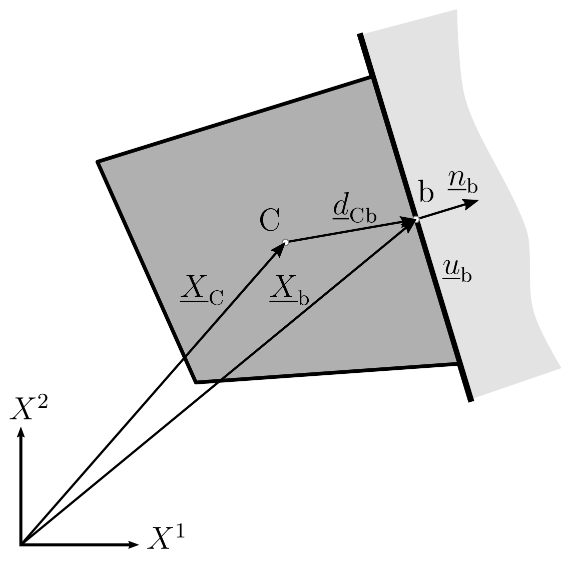

$\underline{n}_\mathrm{b}$
b
$\underline{d}_\mathrm{Cb}$
C
$\underline{u}_\mathrm{b}$
$\underline{X}_\mathrm{C}$
$\underline{X}_\mathrm{b}$
X^2
X^1

Figure 4.2.: Dirichlet BC for a Nonorthogonal Cell

Equation (4.32) is set up for *each* cell of the computational domain and is transformed into a set of algebraic equations of the form $A^j_k \Delta\hat{\tilde{u}}^k = b^j$. The terms in Equation (4.32) have following contribution to the equation system:

$$\begin{aligned} &\overbrace{\sum_{\mathrm{f}} \left(F^i_J\ S^{JK}\ N_K\right)_{\mathrm{f}} A_{\mathrm{f}}}^{b^i_{\mathrm{n}}} + \\ &\underbrace{+\sum_{\mathrm{f}} \left(\left(T^A_J\ S^{JK}\ \delta^i_p + F^i_J\ \mathbb{C}^{JKLM}\ g_{no}\ \delta^o_p\ F^n_L\ T^A_M\right)\ N_K\right)_{\mathrm{f}} A_{\mathrm{f}}\ \Delta\hat{u}^p_A}_{A^i_k\ ,\ b^i_{\mathrm{bf}}} = 0\,. \end{aligned} \tag{4.41}$$

The contribution of the first term in Equation (4.41) is denoted as *nonlinear* contribution b^i_{n} ($i = 1 : dim$) as it is calculated using the nonlinear diffusion flux term, i.e. the residuum. The subscript $(.)_{\mathrm{n}}$ refers to *nonlinear*. Due to the consideration of boundary faces as temporary variables, the second term $K^{iA}_p = (K^{iA}_p)^{\mathrm{ISC}} + (K^{iA}_p)^{\mathrm{CC}}$ reshaped into matrix form is of size $[dim \times (\mathtt{nov} \cdot dim)]$ and contributes to both A^i_k ($k = 1 : \mathtt{nod}$[11]) and b^i_{bf}. The subscript $(.)_{\mathrm{bf}}$ refers to *boundary face* as it is calculated with the values of the prescribed boundary faces. In the assemble process (for a 2D case), the columns $1 : \mathtt{noc}$ and $(\mathtt{nov} + 1) : (\mathtt{nov} + \mathtt{noc})$[12] of K^{iA}_p contribute to A^i_k resulting in the *Newton tangent* A^j_k ($j = 1 : \mathtt{nod}$) with size $[\mathtt{nod} \times \mathtt{nod}]$. The coefficients of the remaining columns[13] are multiplied with the respective prescribed value of the BC and contribute to b^i_{bf}. In the assemble process, the term $b^i = b^i_{\mathrm{n}} + b^i_{\mathrm{bf}}$ results in the *residuum* b^j. The derivation of these terms is discussed for a generic example in Section 5.2.5.

The equations are established for each iteration step t in the NR iterative solution procedure and read

$$(A^j_k)_t\ (\Delta\hat{\tilde{u}}^k)_{t+1} = (b^j)_t\,, \tag{4.42}$$

whereas in analogy to the NRM introduced in Section 2.5, $(A^j_k)_t$ is denoted as *Newton* tangent and $(b^j)_t$ as residuum. The equation is solved for the incremental displacement of the cell centroids $(\Delta\hat{\tilde{u}}^k)_{t+1}$. The solution vector $(\Delta\hat{\tilde{u}}^k)_{t+1}$ contains the incremental cell centroid displacement values in all directions, i.e. $\Delta\hat{u}$, $\Delta\hat{v}$ and $\Delta\hat{w}$ for a 3D case. This is denoted with the tilde, i.e. $\Delta\hat{\tilde{u}}$. The displacement vector of cell centroids $(\hat{\tilde{u}}^k)_{t+1} = (\hat{\tilde{u}}^k)_t + (\Delta\hat{\tilde{u}}^k)_{t+1}$ is updated in each iteration step until a desired convergence criteria for both the increment and residuum is reached.

[11] nod = number of degrees of freedom; nod is equal to number of cells times the dimension

[12] In a 3D case also the columns $(2 \cdot \mathtt{nov} + 1) : (2 \cdot \mathtt{nov} + \mathtt{noc})$.

[13] These are the columns that relate to the boundary faces with prescribed Dirichlet BCs.

5. Implementation and Discussion

In this chapter, the equations developed in Chapter 4 are discussed and explained. The chapter starts with a brief introduction of the implemented code in Section 5.1. In order to clarify the form of the coefficient matrices as well as the treatment of Dirichlet BCs, their derivation is discussed in detail for a generic example in Section 5.2. In parallel, excerpts of pseudocode of the MATLAB© implementation are embedded and explained.

Most of the terms in this chapter are written in symbolic notation and in matrix form. The indices of the components of vectors and tensors must not be confused with the indices of the index notation. They are for explanatory purposes only and relate to the cells and faces as discussed below.

5.1. SOOFVAM

The governing equations were implemented in MATLAB©. The resulting implicit solver was called SOOFVAM[1], based on the in-house MATLAB© code SOOFEAM[2] of the Institute of Strength of Materials at TU Graz. Therefore, SOOFVAM shares some basic `classes` and `functions` with SOOFEAM. Indeed, the author implemented some of the `classes` and `functions` of SOOFEAM as part of a lecture before starting with this thesis. This has been very helpful for the implementation of SOOFVAM.

5.1.1. Structure of SOOFVAM

The proposed code was implemented using object-oriented programming, and its source directory structure is illustrated in Figure 5.1. Unified Modeling Language (UML) diagrams are attached in Appendix A.

Script `soofvam.m`

The script `soofvam.m` (cf. Figure 5.1) is the centerpiece of the implemented code. It defines which `example` is to be solved, whether the mesh is to be created or loaded and starts the `analysis`.

[1]Software for Object-Oriented Finite Volume Analysis in Matlab
[2]Software for Object-Oriented Finite Element Analysis in Matlab

Directory `examples`

The directory `examples` contains all files needed for the setup of a problem. These files are the `.geo` and the `.msh` files, which are necessary for mesh generation. The preprocessing is done with `Gmsh`© as described below. The BCs of the `example` are prescribed within the `MyBCHandler.m` file. Information about `material` and type of `analysis` are prescribed in the `example.m` file. Moreover, there are the directories `meshes` and `results`, in which `mesh` data and `.vtk` files for postprocessing in `Paraview`© are stored, cf. Figure 5.1.

Directory `src`

The directory `src` is the most extensive directory of SOOFVAM, cf. Figure 5.1. Most files are stored in the `namespace nsModel`, which includes all files for the topology of the computational domain. In addition, there are the `namespaces nsDOF`, handling the degrees of freedom, and `nsMaterial`, in which the respective material law is defined. Furthermore, there is the `namespace nsIO`, which contains the classes for the input (`.msh` file) and output (`.vtk` file). The classes for the calculation of the geometrical coherences for the mesh are defined in `namespace nsGeometry`. The most important `namespace` is `nsAnalyser`. Within it, there are the `namespaces nsAnalysis`, in which the linear and nonlinear analysis are implemented, and `namespace nsImplementation` with the `class DiffusionTerm.m`, in which the calculation of the governing equations is executed.

5.1.2. Workflow of SOOFVAM

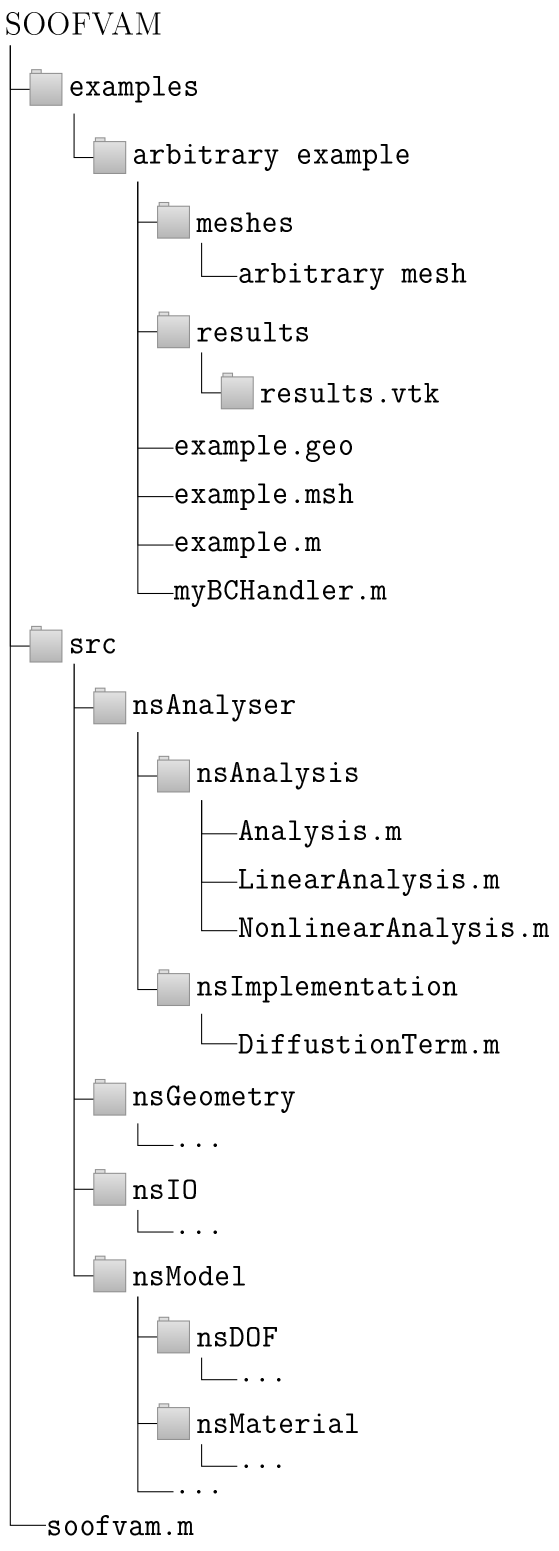
Preprocessing

As mentioned above, the geometry of the computational domain is created using the software `Gmsh`©. `Gmsh`© is an open-source 3D finite element mesh generator. The specification of any input is done either interactively using the graphical user interface or in ASCII text files using `Gmsh`© 's own scripting language [17].

The result of the mesh generation is a `.msh` file containing information about all nodes, elements, i.e. cells, and the boundary faces, due to being generally a 3D finite element mesh generator. Therefore, an additional routine, which is illustrated in Algorithm 5.1, was implemented in SOOFVAM for the creation the topology, i.e. connectivities, especially the generation of interior faces.

Solution Process

The calculation of the deformed configuration in SOOFVAM is explained for a generic example in Section 5.2.



```
SOOFVAM
├── examples
│   └── arbitrary example
│       ├── meshes
│       │   └── arbitrary mesh
│       ├── results
│       │   └── results.vtk
│       ├── example.geo
│       ├── example.msh
│       ├── example.m
│       └── myBCHandler.m
├── src
│   ├── nsAnalyser
│   │   ├── nsAnalysis
│   │   │   ├── Analysis.m
│   │   │   ├── LinearAnalysis.m
│   │   │   └── NonlinearAnalysis.m
│   │   └── nsImplementation
│   │       └── DiffustionTerm.m
│   ├── nsGeometry
│   │   └── ...
│   ├── nsIO
│   │   └── ...
│   └── nsModel
│       ├── nsDOF
│       │   └── ...
│       ├── nsMaterial
│       │   └── ...
│       └── ...
└── soofvam.m
```

Figure 5.1.: Source Directory Structure of SOOFVAM

Algorithm 5.1 Creation of Mesh Topology

```
 1: function createMeshTopology(model)
 2:     call setBoundaryFaceTopology(model)

 3:     for each cell in model do
 4:         determine surrounding_cells
 5:         for each surrounding_cell of cell do
 6:             determine shared_face in node_list of cell and surrounding_cell
 7:             if shared_face is new then
 8:                 create face
 9:                 set owner and neighbour of face
10:                 set topology for owner and neighbour
11:             end if
12:         end for
13:     end for

14:     call calcGeometricQuantities(model)
15: end function

16: function setBoundaryFaceTopology(model)
17:     for each face in boundary do
18:         set owner of face
19:         set topology for owner
20:     end for
21: end function

22: function calcGeometricQuantities(model)
23:     for each face in model do
24:         calculate area, centroid and normal_vector
25:     end for

26:     for each cell in model do
27:         calculate volume and centroid
28:     end for
29: end function
```

Postprocessing

Besides the numerical solution and using the data for plots, the results of SOOFVAM can be visualised with the software `Paraview`©. `Paraview`© is an open-source data analysis and visualisation application. The data exploration can be done interactively using the graphical user interface or programmatically [30]. Some visualised results are illustrated in Chapter 6.

5.2. Solving an Example with SOOFVAM

With the workflow of SOOFVAM in mind, we want to examine the solution process for a generic example. We assume a quadratic computational domain in two dimensions as shown in Figure 5.2. The mesh was generated with e.g. `Gmsh`© comprising an unstructured grid with arbitrary quadrilateral cells.

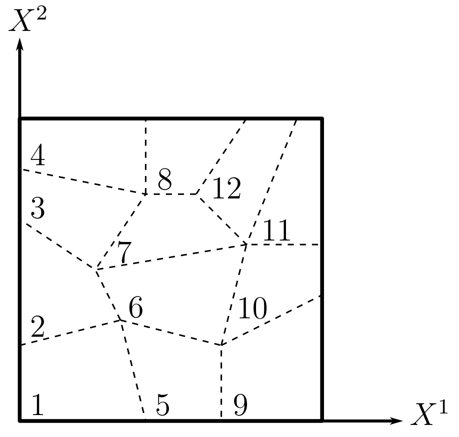


Figure 5.2.: Square Computational Domain with Arbitrary Quadrilateral Cells

We assume a given hyperelastic material behaviour and are interested in solving the governing equations which read

$$\begin{aligned} &\sum_{\mathrm{f}} \left(F^i_J\, S^{JK}\, N_K\right)_{\mathrm{f}} A_{\mathrm{f}} + \\ +&\sum_{\mathrm{f}} \left(\left(T^A_J\, S^{JK}\, \delta^i_p + F^i_J\, \mathbb{C}^{JKLM}\, g_{no}\, \delta^o_p\, F^n_L\, T^A_M\right)\, N_K\right)_{\mathrm{f}} A_{\mathrm{f}}\, \Delta\hat{u}^p_A = 0\,, \end{aligned} \tag{5.1}$$

and we want to transform Equation (5.1) into the form

$$(A^j_k)_t\ (\Delta\hat{\hat{u}}^k)_{t+1} = (b^j)_t \tag{5.2}$$

and solve this set of equations for each iteration step t. In each iteration step, the displacement vector is updated with $(\hat{\hat{u}}^k)_{t+1} = (\hat{\hat{u}}^k)_t + (\Delta\hat{\hat{u}}^k)_{t+1}$. This algorithm is executed until a certain prescribed convergence criterion is reached, which we also assume to be given.

In order to close the system of equations above, we have to prescribe BCs. For our example we assume that the square is fixed at the bottom in X^1- and X^2-direction and we prescribe an arbitrary finite vertical displacement $v = \overline{v}$ at the top as shown in Figure 5.3.

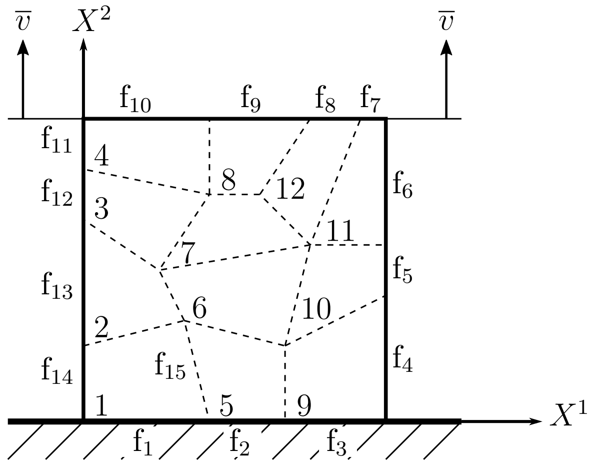


Figure 5.3.: Computational Domain with Prescribed BCs

When the displacement field $\hat{\hat{u}}^K$ for each cell centroid is calculated, we can plot the results or visualise the solution using e.g. `Paraview`©.

In the subsequent sections, the following topics of the solution process of SOOFVAM are discussed:

- ▷ workflow of the solution process (Section 5.2.1),
- ▷ calculation of the cell gradient coefficient matrix (Section 5.2.2),
- ▷ calculation of the face gradient and face gradient coefficient matrix of interior faces (Section 5.2.3),
- ▷ calculation of the face gradient and face gradient coefficient matrix of boundary faces, i.e. treatment of Dirichlet BCs, (Section 5.2.4) and
- ▷ treatment of BCs in the NR method and solution process (Section 5.2.5).

5.2.1. Workflow of the Solution Process

The workflow of the proposed implicit FV solver for large deformations is illustrated in Algorithm 5.2 and Algorithm 5.3. The `function run` calls two other `functions`: on the one hand `solveEquationSystem`, in which the NRM is embedded, shown in Algorithm 5.3, and on the other hand `solveFVSystem`, in which the linear set of equations is assembled and solved, illustrated in Algorithm 5.10.

One of the first steps in `solveEquationSystem` is the calculation of the coefficient matrices in global indices of each cell of the computational domain. As the governing equations are in total *Lagrangian* formulation, this step is done only once at the beginning of the analysis. After this, the NR loop starts. In the first iteration step t, the *reference* values, i.e. *increment* and *residuum*, required for the *normed* values are determined. The normed values are compared with the preset convergence criteria. The analysis ends, if the criteria are achieved. If a certain prescribed maximum iteration step t_{max} is reached (and the convergence criteria are not), the analysis is aborted, cf. Algorithm 5.3.

Algorithm 5.2 Run Analysis

```
1: function run(analysis)
2:     calculate noc ← number of cells
3:     calculate nod ← number of DOFs
4:     write .vtk file                ▷ displacements of reference configuration, $\underline{u} = \underline{0}$
5:     call solveEquationSystem(noc, nod)
6: end function
```

5.2.2. Explicit and Implicit Cell Gradient

In Line 5 of Algorithm 5.3 the `function calcGlobalCM` to calculate the coefficient matrix of the cell gradient is called. The internal workflow of this function is described below by means of the introduced example.

As mentioned in Section 4.2.1, in order to calculate the coefficients of the coefficient matrix T_J^A, the geometry matrix $\underline{\underline{G}}$ is needed. It can be calculated as, cf. Equation (3.27)

$$\underline{\underline{G}} = (\underline{\underline{d}}^T\ \underline{\underline{w}}^T\ \underline{\underline{w}}\ \underline{\underline{d}})^{-1}\underline{\underline{d}}^T\ \underline{\underline{w}}^T\ \underline{\underline{w}}\,. \tag{5.3}$$

The size of the geometry matrix $\underline{\underline{G}}$ is [$dim \times$ `non`], where *dim* refers to *dimension* and `non` to *number of neighbours*. The geometry matrix for e.g. cell with *global* index 1 may look as follows:

$$\underline{\underline{G}} = (\underline{\underline{d}}^T\ \underline{\underline{w}}^T\ \underline{\underline{w}}\ \underline{\underline{d}})^{-1}\underline{\underline{d}}^T\ \underline{\underline{w}}^T\ \underline{\underline{w}} = \begin{pmatrix} G_{12} & G_{13} \\ G_{22} & G_{23} \end{pmatrix}\,, \tag{5.4}$$

since cell 1 has two neighbours (cell 2 and cell 5). The components of $\underline{\underline{G}}$ read G_{ij}, where i refers to the row and j to the *local* index of the neighbour. With the geometry matrix, one can calculate the *explicit* form of the cell gradient. It reads in symbolic notation, cf. Equation (4.23):

$$({}^{\text{ex}}\underline{\nabla}^0\underline{u})_{\text{C}} = (\underline{\underline{G}})_{\text{C}}\ (\underline{\underline{\hat{u}}})_{\text{CF}}\,, \tag{5.5}$$

with the matrix of cell displacement differences between owner and neighbours $(\underline{\underline{\hat{u}}})_{\text{CF}}$. In matrix form, Equation (5.5) reads for cell 1:

$$\left({}^{\text{ex}}\underline{\nabla}^0\underline{u}\right)_1 = \begin{pmatrix} G_{12} & G_{13} \\ G_{22} & G_{23} \end{pmatrix}\begin{pmatrix} \hat{u}_2 - \hat{u}_1 & \hat{v}_2 - \hat{v}_1 \\ \hat{u}_3 - \hat{u}_1 & \hat{v}_3 - \hat{v}_1 \end{pmatrix}\,, \tag{5.6}$$

Algorithm 5.3 Workflow of Total Lagrangian Solution Procedure

```
1: function solveEquationSystem(analysis, noc, nod)
2:     call incorporateBC(BCHandler)                     ▷ load boundary conditions
3:     calculate nov ← number of variables

4:     for each cell do
5:         call calcGlobalCM(cell)          ▷ coefficient matrix of cell, Section 5.2.2
6:     end for

7:     for iteration step $t = 1$ to $t_{max}$ do              ▷ loop for NR iteration procedure
8:         inc, res ← call solveFVSystem(noc, nod, nov)

9:         if $t = 1$ then
10:            set reference increment ref_inc ← inc
11:            set reference residuum ref_res ← res
12:        end if

13:        update displacement vector $(\hat{\hat{u}}^k)_{t+1} = (\hat{\hat{u}}^k)_t + (\Delta\hat{\hat{u}}^k)_{t+1}$
14:        calculate normed increment normed_inc ← $\frac{\texttt{inc}}{\texttt{ref\_inc}}$
15:        calculate normed residuum normed_res ← $\frac{\texttt{res}}{\texttt{ref\_res}}$

16:        if normed_inc and normed_res < convergence criteria then
17:            break
18:        else if $t = t_{max}$ then
19:            error: "convergence not reached" → abort analysis
20:        end if
21:    end for

22:    update nodal values                     ▷ via interpolation and extrapolation
23:    calculate stresses
24:    write .vtk file                        ▷ displacements of current configuration
25: end function
```

where index 1 is the local index for the owner cell 1 and the indices 2 and 3 are the local indices of the neighbours 2 and 5, respectively. For the derivation of the coefficient matrix $\underline{\underline{T}}$, we start with Equation (5.5), but assume that we want to calculate the *implicit* form of the cell gradient:

$$({}^{\text{im}}\underline{\nabla}^0 \underline{\Delta u})_{\text{C}} = (\underline{\underline{G}})_{\text{C}} \; (\underline{\underline{\Delta \hat{\hat{u}}}})_{\text{CF}} \,, \tag{5.7}$$

with the matrix of incremental cell displacement differences between owner and neighbours $(\underline{\underline{\Delta \hat{\hat{u}}}})_{\text{CF}}$. The matrix form of Equation (5.7) reads for cell 1:

$$\left({}^{\text{im}}\underline{\nabla}^0 \underline{\Delta u}\right)_1 = \begin{pmatrix} G_{12} & G_{13} \\ G_{22} & G_{23} \end{pmatrix} \begin{pmatrix} \Delta\hat{u}_2 - \Delta\hat{u}_1 & \Delta\hat{v}_2 - \Delta\hat{v}_1 \\ \Delta\hat{u}_3 - \Delta\hat{u}_1 & \Delta\hat{v}_3 - \Delta\hat{v}_1 \end{pmatrix} . \tag{5.8}$$

Applying the matrix multiplication yields

$$\begin{aligned} &\left({}^{\text{im}}\underline{\nabla}^0 \underline{\Delta u}\right)_1 = \\ &= \begin{pmatrix} G_{11}\Delta\hat{u}_1 + G_{12}\Delta\hat{u}_2 + G_{13}\Delta\hat{u}_3 & G_{11}\Delta\hat{v}_1 + G_{12}\Delta\hat{v}_2 + G_{13}\Delta\hat{v}_3 \\ G_{21}\Delta\hat{u}_1 + G_{22}\Delta\hat{u}_2 + G_{23}\Delta\hat{u}_3 & G_{21}\Delta\hat{v}_1 + G_{22}\Delta\hat{v}_2 + G_{23}\Delta\hat{v}_3 \end{pmatrix} , \end{aligned} \tag{5.9}$$

with $G_{i1} = -\left(G_{i2} + G_{i3}\right)$ and $i = 1, 2$. If we take a closer look at the components in Equation (5.9), one may observe that the G_{ij} terms are identical for each column due to the matrix multiplication. Therefore, the resulting coefficient matrices for the scalar displacements Δu and Δv have to be identical. As a result, for each cell in the computational domain, only *one* cell gradient coefficient matrix $\underline{\underline{T}}$ needs to be calculated, and we derive the coefficient matrix $\underline{\underline{T}}$ for the incremental displacement $\Delta\tilde{u}$[3]. The cell gradient for $\Delta\tilde{u}$ can be set up from e.g. the first column of Equation (5.9). It reads for cell 1:

$$\left({}^{\text{im}}\underline{\nabla}^0 \Delta\tilde{u}\right)_1 = \begin{pmatrix} G_{11}\Delta\hat{\hat{u}}_1 + G_{12}\Delta\hat{\hat{u}}_2 + G_{13}\Delta\hat{\hat{u}}_3 \\ G_{21}\Delta\hat{\hat{u}}_1 + G_{22}\Delta\hat{\hat{u}}_2 + G_{23}\Delta\hat{\hat{u}}_3 \end{pmatrix} = \begin{pmatrix} \frac{\partial \Delta\tilde{u}}{\partial X^1} \\ \frac{\partial \Delta\tilde{u}}{\partial X^2} \end{pmatrix}_1 , \tag{5.10}$$

where $\Delta\hat{\hat{u}}_i$ are the unknown incremental displacements of the owner cell 1 (index 1) and its two neighbours 2 and 5 (index 2 and 3, respectively). As illustrated in Equation (5.10), each row represents the partial derivative of the displacement $\Delta\tilde{u}$ with respect to the coordinate directions for cell 1. With a mathematical manipulation, one obtains for Equation (5.10)

$$\left({}^{\text{im}}\underline{\nabla}^0 \Delta\tilde{u}\right)_1 = \begin{pmatrix} G_{11} & G_{12} & G_{13} \\ G_{21} & G_{22} & G_{23} \end{pmatrix} \begin{pmatrix} \Delta\hat{\hat{u}}_1 \\ \Delta\hat{\hat{u}}_2 \\ \Delta\hat{\hat{u}}_3 \end{pmatrix} , \tag{5.11}$$

and we define the matrix on the RHS as the *local* coefficient matrix ${}^{\text{loc}}\underline{\underline{T}}$:

$$\left({}^{\text{loc}}\underline{\underline{T}}\right)_1 := \begin{pmatrix} G_{11} & G_{12} & G_{13} \\ G_{21} & G_{22} & G_{23} \end{pmatrix} , \tag{5.12}$$

[3]The tilde indicates that $\Delta\tilde{u}$ can either refer to Δu or Δv.

with the superscript ${}^{\text{loc}}(.)$ referring to *local.* In order to obtain the *global* coefficient matrix, Equation (5.11) needs to be rewritten. For the introduced example, the implicit form of the gradient, in global indices, reads

$$\left({}^{\text{im}}\underline{\nabla}^0 \Delta\tilde{u}\right)_1 = \begin{pmatrix} G_{11} & G_{12} & 0 & 0 & G_{13} & 0 & \cdots & 0 \\ G_{21} & G_{22} & 0 & 0 & G_{23} & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} \Delta\hat{\tilde{u}}_1 \\ \Delta\hat{\tilde{u}}_2 \\ \Delta\hat{\tilde{u}}_3 \\ \Delta\hat{\tilde{u}}_4 \\ \Delta\hat{\tilde{u}}_5 \\ \vdots \\ \Delta\hat{\tilde{u}}_{12} \\ \Delta\hat{\tilde{u}}_{\mathrm{f}_7} \\ \Delta\hat{\tilde{u}}_{\mathrm{f}_8} \\ \Delta\hat{\tilde{u}}_{\mathrm{f}_9} \\ \Delta\hat{\tilde{u}}_{\mathrm{f}_{10}} \end{pmatrix}. \tag{5.13}$$

We define the *global* coefficient matrix ${}^{\text{glo}}\underline{\underline{T}}$ as

$$\left({}^{\text{glo}}\underline{\underline{T}}\right)_1 := \begin{pmatrix} G_{11} & G_{12} & 0 & 0 & G_{13} & 0 & \cdots & 0 \\ G_{21} & G_{22} & 0 & 0 & G_{23} & 0 & \cdots & 0 \end{pmatrix} \tag{5.14}$$

with the superscript ${}^{\text{glo}}(.)$ referring to *global.* As can be observed, the coefficients in the global coefficient matrix are arranged according to the global index of the respective cell. Moreover, the vector in Equation (5.13) does not only contain the displacement value of the cell centroids but also the displacements of the faces f_7 - f_{10} as (temporary) additional variables. This is due to the prescription of the top boundary with value $v = \overline{v}$.

The explained procedure is performed for *each* cell of the computational domain and the coherences described above are summarized in Algorithm 5.4 and Algorithm 5.5.

Algorithm 5.4 Calculation of Global Coefficient Matrix

```
1: function calcGlobalCM(cell)                                          ▷ Equation (5.14)
2:     get local coefficient matrix $\left({}^{\text{loc}}\underline{\underline{T}}\right)_{\mathrm{C}}$ ← assembleLocalCM(cell)
3:     initialise global coefficient matrix $\left({}^{\text{glo}}\underline{\underline{T}}\right)_{\mathrm{C}}$
4:     assemble global coefficient matrix $\left({}^{\text{glo}}\underline{\underline{T}}\right)_{\mathrm{C}}$ according to global indices of cells
5:     save coefficient matrix as property of cell
6: end function
```

Algorithm 5.5 Calculation of Local Coefficient Matrix

```
1: function assembleLocalCM(cell)                                   ▷ Equation (5.12)
2:     calculate geometry matrix $\underline{\underline{G}}$        ▷ Equation (5.4)
3:     initialise local coefficient matrix $\left({}^{\mathrm{loc}}\underline{\underline{T}}\right)_{\mathrm{C}}$
4:     calculate first column of $\left({}^{\mathrm{loc}}\underline{\underline{T}}\right)_{\mathrm{C}} \leftarrow T_{i1} = -(G_{i2} + G_{i3})$
5:     calculate remaining columns of $\left({}^{\mathrm{loc}}\underline{\underline{T}}\right)_{\mathrm{C}}$
6:     return $\left({}^{\mathrm{loc}}\underline{\underline{T}}\right)_{\mathrm{C}}$
7: end function
```

5.2.3. Explicit and Implicit Face Gradient

The explicit face gradient is needed for the calculation of Equation (5.1), since

$$(\underline{\underline{F}})_{\mathrm{f}} = \left(\underline{\underline{I}} + {}^{\mathrm{ex}}\underline{\nabla}^0 \underline{u}\right)_{\mathrm{f}} , \tag{5.15}$$

with the identity matrix $\underline{\underline{I}}$ and the explicit face gradient $\left({}^{\mathrm{ex}}\underline{\nabla}^0 \underline{u}\right)_{\mathrm{f}}$. For explanatory purposes we want to calculate the face gradient of face f_{15}, cf. Figure 5.3. Applying the interpolation of the face gradient (cf. Equation (3.28)), one obtains

$$\left({}^{\mathrm{ex}}\underline{\nabla}^0 \underline{u}\right)_{\mathrm{f}_{15}} = g_{\mathrm{C}} \; \left({}^{\mathrm{ex}}\underline{\nabla}^0 \underline{u}\right)_1 + (1 - g_{\mathrm{C}}) \left({}^{\mathrm{ex}}\underline{\nabla}^0 \underline{u}\right)_5 . \tag{5.16}$$

The explicit cell gradients are calculated with Equation (5.6), whereas the matrix containing the differences of displacements is determined using the *latest available* displacement values, i.e. the values of the previous iteration step in the NRM.

Moreover, Equation (5.1) requires the coefficient matrix of the *face gradient*. For our example we want to calculate the coefficient matrix for the interior face with global index 15, cf. Figure 5.3. Applying the interpolation of the face gradient (cf. Equation (3.28)), one obtains

$$\left({}^{\mathrm{im}}\underline{\nabla}^0 \underline{u}\right)_{\mathrm{f}_{15}} = \underbrace{\left(g_{\mathrm{C}} \; (\underline{\underline{T}})_1 + (1 - g_{\mathrm{C}})(\underline{\underline{T}})_5\right)}_{:=(\underline{\underline{T}})_{\mathrm{f}_{15}}} (\Delta\hat{\underline{\underline{u}}})_{\mathrm{CF}} , \tag{5.17}$$

with the coefficient matrices for cell 1 and 5 calculated as described above (cf. Equation (5.14)), the geometric weighting factor g_{C}, and $(\Delta\hat{\underline{\underline{u}}})_{\mathrm{CF}}$ relates to the matrix of cell displacement differences between owner and neighbours. We decided to calculate g_{C} with the option *distance* as introduced in Equation (3.29). As illustrated in Equation (5.17), for further calculations we define the interpolation of the cell gradient coefficient matrices as the face gradient coefficient matrix $(\underline{\underline{T}})_{\mathrm{f}_{15}}$.

As one may observe, the example illustrated in Figure 5.3 consists of an unstructured mesh. Hence, we decided to use the *minimum correction* approach as described in Section 3.2.3[4].

The nonorthogonal treatment of the explicit face gradient of face f_{15} reads, cf. Equation (3.31):

$$\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{f}_{15}} = \left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{f}_{15}} - \frac{1}{\|\underline{d}_{15}\|^2}\,\underline{d}_{15}\,\underline{d}_{15}^T\,\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{f}_{15}} + \frac{1}{\|\underline{d}_{15}\|^2}\,\underline{d}_{15}\,\left(\underline{\hat{u}}_5 - \underline{\hat{u}}_1\right)^T\,, \quad (5.18)$$

where the superscript asterisk refers to the *corrected* face gradient. For the difference of cell centroid displacement vectors in the last term of Equation (5.18), values of the latest available iteration step are used. One has to be aware that $\underline{d}_{15}$ relates to the distance vector between the cells 1 and 5 and does not refer to any quantity of face f_{15}[5]. The gradient in Equation (5.18) is the transposition of the Jacobian of the vector field $\underline{u}$, hence Equation (5.18) is transposed for further calculations in SOOFVAM.

The nonorthogonal treatment for the coefficient matrix $(\underline{\underline{T}})_{\mathrm{f}_{15}}$ can be calculated analogously and reads

$$(\underline{\underline{T}})^*_{\mathrm{f}_{15}} = (\underline{\underline{T}})_{\mathrm{f}_{15}} - \frac{1}{\|\underline{d}_{15}\|^2}\,\underline{d}_{15}\,\underline{d}_{15}^T\,(\underline{\underline{T}})_{\mathrm{f}_{15}} + \frac{1}{\|\underline{d}_{15}\|^2}\,\underline{d}_{15}\,(\Delta\hat{\hat{u}}_5 - \Delta\hat{\hat{u}}_1)\,. \quad (5.19)$$

$\Delta\hat{\hat{u}}$ can either refer to the incremental displacements $\Delta\hat{u}$ or $\Delta\hat{v}$. For interior faces, the coefficient matrices remain identical for each displacement. Therefore, $(\underline{\underline{T}})^*_{\mathrm{f}_{15}}$ is calculated only once. As Equation (5.19) deals with coefficient matrices, also the last term refers to a matrix of the form

$$\frac{1}{\|\underline{d}_{15}\|^2}\,\underline{d}_{15}\,(\Delta\hat{\hat{u}}_5 - \Delta\hat{\hat{u}}_1) := \frac{1}{\|\underline{d}_{15}\|}\begin{pmatrix} -e_{15,1} & 0 & 0 & 0 & e_{15,1} & 0 & \cdots & 0 \\ -e_{15,2} & 0 & 0 & 0 & e_{15,2} & 0 & \cdots & 0 \end{pmatrix}\,, \quad (5.20)$$

whereas the matrix has the same size as $(\underline{\underline{T}})_{\mathrm{f}_{15}}$. This means, the matrix in Equation (5.20) is of size $[2 \times 16]$, since the computational domain consists of 12 cells and 4 additional (temporary) variables due to the prescribed boundary faces. The columns of the nonzero elements in the matrix in Equation (5.20) are 1 and 5, referring to the cell's global index. Therefore, the incremental displacement values $\Delta\hat{\hat{u}}_1$ and $\Delta\hat{\hat{u}}_5$ do not disappear, but are only representatives for the positions of the nonzero elements of the matrix in Equation (5.20). The unit vector $\underline{e}_{15}$ is calculated as

$$\underline{e}_{15} = \frac{\underline{d}_{15}}{\|\underline{d}_{15}\|} = \begin{pmatrix} e_{15,1} \\ e_{15,2} \end{pmatrix}\,. \quad (5.21)$$

Any other *interior* face of the computational domain is treated analogously.

[4]Note that also for orthogonal cells, the derivation in SOOFVAM is executed analogously.

[5]In this context it is a coincidence that the face between cell 1 and 5 has global index 15.

5.2.4. Treatment of Dirichlet BCs

The face gradient and its coefficient matrix for a *boundary* face with prescribed Dirichlet BCs is calculated similar to the interior faces. Again, we want to use the *minimum correction* approach for the nonorthogonal treatment. Note that the derivations below are executed for *each*[6] boundary face with prescribed Dirichlet BCs. We calculate the terms for face f_9, cf. Figure 5.3.

For the face gradient coefficient matrix of a boundary face we assume that the coefficient matrix is *equal*[7] to the gradient of its owner cell, i.e. the coefficient matrix of boundary face f_9 reads:

$$(\underline{\underline{T}})_{f_9} = (\underline{\underline{T}})_8 \,. \tag{5.22}$$

Applying Equation (3.31) for Equation (5.22) yields

$$(\underline{\underline{T}})^*_{f_9} = (\underline{\underline{T}})_{f_9} - \frac{1}{\|\underline{d}_{8f_9}\|^2}\, \underline{d}_{8f_9}\, \underline{d}^T_{8f_9}\, (\underline{\underline{T}})_{f_{f_9}} + \frac{1}{\|\underline{d}_{8f_9}\|^2}\, \underline{d}_{8f_9}\, (\Delta\hat{\hat{u}}_{f_9} - \Delta\hat{\hat{u}}_8)\,. \tag{5.23}$$

The last term in Equation (5.23) reads

$$\frac{1}{\|\underline{d}_{8f_9}\|^2}\, \underline{d}_{8f_9}\, (\Delta\hat{\hat{u}}_{f_9} - \Delta\hat{\hat{u}}_8) := \frac{1}{\|\underline{d}_{8f_9}\|} \begin{pmatrix} 0 & \cdots & 0 & -e_{8f_9,1} & 0 & \cdots & 0 & e_{8f_9,1} & 0 \\ 0 & \cdots & 0 & -e_{8f_9,2} & 0 & \cdots & 0 & e_{8f_9,2} & 0 \end{pmatrix}, \tag{5.24}$$

whereas the matrix in Equation (5.24) is of same size as $(\underline{\underline{T}})_{f_9}$, which means $[2 \times 16]$ for our example. The columns of the nonzero elements in the matrix are 8, referring to the global index of cell 8 and the corresponding position of boundary face f_9. For the introduced example this position is 15, since f_9 is the third boundary face with a prescribed boundary face value not equal to zero and $12 + 3 = 15$. The unit vector $\underline{e}_{8f_9}$ is calculated analogously to Equation (5.21).

One ought to be aware, that there is a significant difference to the coefficient matrices of interior faces: In contrast to interior faces, where the coefficient matrices are identical for *all*[8] displacements, for the boundary faces the coefficient matrices are calculated separately for *each* displacement. This is due to the fact that for each boundary face the boundary value can be prescribed independent of each other. In face f_9 only the value in X^2- direction is constrained. Hence, just the coefficient matrix for displacement v is determined according to Equation (5.23). For the coefficient matrix for displacement u the last term in Equation (5.23), i.e. the term in Equation (5.24), is a zero matrix. Therefore, if the value of a boundary face is not prescribed, the coefficient matrix is calculated with Equation (5.23), but omitting the last term (Equation (5.24)).

Moreover, for boundary faces with a prescribed value of zero (faces f_1 - f_3, cf. Figure 5.3) Equation (5.24) is also treated in a special way. Since these faces do

[6]i.e. not only for the faces of the top boundary (prescribed value $\neq 0$) but also for the faces of the bottom boundary (prescribed value $= 0$)

[7]Since there is no neighbour cell, the owner cell fully affects the boundary face.

[8]i.e. u and v for our example

not appear as (temporary) variables in the equation system, Equation (5.24) for e.g. boundary face f_2 reads

$$\frac{1}{\|\underline{d}_{5\mathrm{f}_2}\|^2}\,\underline{d}_{5\mathrm{f}_2}\,(0-\Delta\hat{\underline{u}}_2) := \frac{1}{\|\underline{d}_{5\mathrm{f}_2}\|}\begin{pmatrix}0 & -e_{5\mathrm{f}_2,1} & 0 & \cdots & 0\\ 0 & -e_{5\mathrm{f}_2,2} & 0 & \cdots & 0\end{pmatrix}. \tag{5.25}$$

That means these faces have only a *cell* contribution and there is no contribution for the boundary face itself.

The distinction of boundary faces with prescribed value not equal to zero and equal to zero was introduced for a specific reason. In general, one could add the boundary faces of the bottom boundary to the *number of variables* (`nov`) as well. Since in the assemble process the coefficients of $\underline{\underline{T}}$ corresponding to the boundary faces are multiplied with the prescribed boundary face value (cf. Section 5.2.5), which is zero for the faces of the bottom boundary, the apparent influence of boundary faces with prescribed value of zero can already be neglected in advance. Therefore, only the faces of the top boundary are added as additional, temporary variables, i.e. added to `nov`. Furthermore, this also positively affects the computational efficiency of SOOFVAM, since less variables are involved in the calculation.

For the nonorthogonal treatment of the explicit face gradient, we also assume that the face gradient is equal to the gradient of the owner cell, i.e. for boundary face f_9:

$$\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{f}_9} = \left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_8 . \tag{5.26}$$

The nonorthogonal treatment yields

$$\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{f}_9} = \left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{f}_9} - \frac{1}{\|\underline{d}_{8\mathrm{f}_9}\|^2}\,\underline{d}_{8\mathrm{f}_9}\,\underline{d}^T_{8\mathrm{f}_9}\,\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{f}_9} + \frac{1}{\|\underline{d}_{8\mathrm{f}_9}\|^2}\,\underline{d}_{8\mathrm{f}_9}\,(\hat{\underline{u}}_{\mathrm{f}_9}-\hat{\underline{u}}_8)^T . \tag{5.27}$$

For the last term in Equation (5.27) latest available displacement values are used. The gradient in Equation (5.27) is the transposition of the Jacobian of the vector field $\underline{u}$, hence Equation (5.27) is transposed for further calculations in SOOFVAM.

Any other *boundary* face of the computational domain is treated analogously as described above.

The described coherences regarding the coefficient matrices are illustrated in Algorithm 5.6. The pseudocode for `function correctBfaceGradientCM`[9] is shown in Algorithm 5.7 and for `function correctFaceGradientCM`[10] in Algorithm 5.8. The algorithms for the explicit face gradient are shown in Algorithm 5.9.

[9] coefficient matrix of boundary faces

[10] coefficient matrix of interior faces

Algorithm 5.6 Calculation of Face Gradient Coefficient Matrix

```
 1: function getGradientCoefficientMatrix(cell, face, noc, nov)
 2:     if neighbour of face is empty then                ▷ the face is a boundary face
 3:         $(\underline{\underline{T}})^*_{\mathrm{b},i}$ ← call correctBfaceGradientCM(cell, face, noc, nov)
 4:     else                                              ▷ the face is an inner face
 5:         $(\underline{\underline{T}})^*_{\mathrm{f}}$ ← call correctFaceGradientCM(cell, face, noc, nov)
 6:     end if

 7:     return $(\underline{\underline{T}})^*_{\mathrm{b},i}$ or $(\underline{\underline{T}})^*_{\mathrm{f}}$      ▷ $i$ either refers to $u$, $v$ or $w$
 8: end function
```

Algorithm 5.7 Nonorthogonal Treatment of Boundary Face Coefficient Matrix

```
 1: function correctBfaceGradientCM(cell, face, noc, nov)
 2:     set $(\underline{\underline{T}})_{\mathrm{b}}$ ← $(\underline{\underline{T}})_{\mathrm{C}}$                          ▷ Equation (5.22)
 3:     calculate corrected face gradient $(\underline{\underline{T}})^*_{\mathrm{b}}$     ▷ Equation (5.23)
 4:     if displacement $u$ is prescribed then
 5:         adapt $(\underline{\underline{T}})^*_{\mathrm{b},u}$ ← $(\underline{\underline{T}})^*_{\mathrm{b}}$
 6:     else if displacement $v$ is prescribed then
 7:         adapt $(\underline{\underline{T}})^*_{\mathrm{b},v}$ ← $(\underline{\underline{T}})^*_{\mathrm{b}}$
 8:     else if displaement $w$ is prescribed then
 9:         adapt $(\underline{\underline{T}})^*_{\mathrm{b},w}$ ← $(\underline{\underline{T}})^*_{\mathrm{b}}$
10:     end if

11:     return $(\underline{\underline{T}})^*_{\mathrm{b},i}$
12: end function
```

Algorithm 5.8 Nonorthogonal Treatment of Interior Face Coefficient Matrix

```
 1: function correctFaceGradientCM(cell, face, noc, nov)
 2:     get $(\underline{\underline{T}})_{\mathrm{C}}$, $(\underline{\underline{T}})_{F}$ ← coefficient matrices of owner and neighbour
 3:     calculate interpolated face gradient $(\underline{\underline{T}})_{\mathrm{f}}$        ▷ Equation (5.17)
 4:     calculate corrected face gradient $(\underline{\underline{T}})^*_{\mathrm{f}}$         ▷ Equation (5.19)

 5:     return $(\underline{\underline{T}})^*_{\mathrm{f}}$
 6: end function
```

Algorithm 5.9 Calculation and Correction of Gradient of Interior and Boundary Face

```
1: function calcGradUFace(cell, face)
2:     if neighbour of face is empty then                    ▷ the face is a boundary face
3:         $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{b}}$ ← call correctBfaceGradient(cell, face)
4:     else                                                   ▷ the face is an inner face
5:         $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{f}}$ ← call correctFaceGradient(cell, face)
6:     end if

7:     return $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{b}}$ or $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{f}}$
8: end function

9: function correctBfaceGradient(cell, face)
10:    set $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{b}} \leftarrow \left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{C}}$                 ▷ Equation (5.26)
11:    calculate corrected face gradient $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{b}}$     ▷ Equation (5.27)

12:    return $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{b}}$
13: end function

14: function correctFaceGradient(cell, face)
15:    get $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{C}}$, $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{F}$ ← cell gradients of owner and neighbour
16:    calculate interpolated face gradient $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)_{\mathrm{f}}$  ▷ Equation (5.16)
17:    calculate corrected face gradient $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{f}}$     ▷ Equation (5.18)

18:    return $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{f}}$
19: end function
```

5.2.5. Solution of the Equation System

Since we are able to calculate each face gradient and face gradient coefficient matrix, we are now interested in the assemble process and the solution of the resulting system of algebraic equations. The system of equations to be solved in index notation reads

$$\begin{aligned}&\overbrace{\sum_{\mathrm{f}} \left(F^i_J\ S^{JK}\ N_K\right)_{\mathrm{f}} A_{\mathrm{f}}}^{b^i_{\mathrm{n}}} + \\ &\underbrace{+\sum_{\mathrm{f}} \left(\left(T^A_J\ S^{JK}\ \delta^i_p + F^i_J\ \mathbb{C}^{JKLM}\ g_{no}\ \delta^o_p\ F^n_L\ T^A_M\right)\ N_K\right)_{\mathrm{f}} A_{\mathrm{f}}\ \Delta\hat{u}^p_A}_{A^i_k\ ,\ b^i_{\mathrm{bf}}} = 0\,.\end{aligned} \tag{5.28}$$

Now we want to transform this equation into the general form

$$(A^j_k)_t\ (\Delta\hat{\hat{u}}^k)_{t+1} = (b^j)_t\,, \tag{5.29}$$

as we are then able to solve for the incremental displacement values of the cell centroids $(\Delta\hat{\hat{u}}^k)_{t+1}$ and can update the displacement vector in each iteration step t with $(\hat{\hat{u}}^k)_{t+1} = (\hat{\hat{u}}^k)_t + (\Delta\hat{\hat{u}}^k)_{t+1}$. As can be observed in Equation (5.28), the terms have different contributions to the equation system which is explained below for the introduced example.

The set of linear algebraic equations in matrix form reads

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & & \cdots & A_{1\mathrm{d}} \\ A_{21} & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & & & & \\ & & & & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & A_{\mathrm{d-1d}} \\ A_{\mathrm{d1}} & \cdots & & \cdots & A_{\mathrm{dd-1}} & A_{\mathrm{dd}} \end{pmatrix}_t \begin{pmatrix} \Delta u_1 \\ \vdots \\ \Delta u_{\mathrm{c}} \\ \Delta v_1 \\ \vdots \\ \Delta v_{\mathrm{c}} \end{pmatrix}_{t+1} = \begin{pmatrix} b_1 \\ \vdots \\ \\ \\ \vdots \\ b_{\mathrm{d}} \end{pmatrix}_t\,, \tag{5.30}$$

where the subscript $(.)_{\mathrm{d}}$ refers to `nod` and the subscript $(.)_{\mathrm{c}}$ to `noc`. The size of matrix $\left(\underline{\underline{A}}\right)_t$ is [`nod` $\times$ `nod`] and the size of the vectors $\left(\underline{b}\right)_t$ and $(\Delta\underline{\hat{\hat{u}}})_{t+1}$ equals [`nod` $\times$ 1]. For the example illustrated in Figure 5.3 this means that the size of $\underline{\underline{A}}$ equals $[24 \times 24]$, and the size of the two vectors is $[24 \times 1]$. As mentioned in Section 4.3 and Section 5.2.2, the coefficient matrices are initialised with size [dim $\times$ `nov`] which results in a size of [dim $\times$ (`nov` $\cdot$ dim)] for the second term[11] in Equation (5.28). For the example this means that the coefficient matrices are of size $[2 \times 16]$ and the linearised diffusion flux in Equation (5.28) is of size $[2 \times 32]$ and has therefore more columns than $\underline{\underline{A}}$. In the assemble process this issue is fixed.

[11]It refers to the linearised diffusion flux term regarding the denotion of Section 4.1.2.

For explanatory purposes, we want to assemble the terms of the cell with global index 2. The linearised diffusion flux of cell 2 may look like as follows:

$$\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_2 = \begin{pmatrix} \underbrace{\begin{matrix} DJ_{11} & \cdots & DJ_{112} \\ DJ_{21} & \cdots & DJ_{212} \end{matrix}}_{\Delta\hat{u}_1\text{-}\Delta\hat{u}_{12}} & \underbrace{\begin{matrix} DJ_{113} & \cdots & DJ_{116} \\ DJ_{213} & \cdots & DJ_{216} \end{matrix}}_{\Delta\hat{u}_{\mathrm{f}_7}\text{-}\Delta\hat{u}_{\mathrm{f}_{10}}} & \underbrace{\begin{matrix} DJ_{117} & \cdots & DJ_{132} \\ DJ_{217} & \cdots & DJ_{232} \end{matrix}}_{\Delta\hat{v}_1\text{-}\Delta\hat{v}_{\mathrm{f}_{10}}} \end{pmatrix}, \tag{5.31}$$

where the indices of the elements DJ_{ij} refer to the row i and the global index j. As can be seen in Equation (5.31), only the columns 1 - 12 relate to the incremental cell displacements $\Delta\hat{u}_{\mathrm{C}}$ and the columns 17 - 28 relate to $\Delta\hat{v}_{\mathrm{C}}$. In the assemble process only the columns corresponding to the cell displacements are included in $\underline{\underline{A}}$:

$$\underline{\underline{A}} = \begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \\ \underbrace{\begin{matrix} DJ_{11} & \cdots & DJ_{112} \\ DJ_{21} & \cdots & DJ_{212} \end{matrix}}_{\Delta\hat{u}_1\text{-}\Delta\hat{u}_{12}} & \underbrace{\begin{matrix} DJ_{117} & \cdots & DJ_{128} \\ DJ_{217} & \cdots & DJ_{228} \end{matrix}}_{\Delta\hat{v}_1\text{-}\Delta\hat{v}_{12}} \\ \vdots & \vdots \end{pmatrix}, \tag{5.32}$$

which results in the appropriate *Newton* tangent if this is done for each cell of the computational domain. The remaining columns contribute to the residuum. In the assemble process, the coefficients of the remaining columns in Equation (5.31) are multiplied with their prescribed values of the Dirichlet BC:

$$(\underline{b}_{\mathrm{bf}})_2 = -\begin{pmatrix} \underbrace{\begin{matrix} DJ_{113} & \cdots & DJ_{116} \\ DJ_{213} & \cdots & DJ_{216} \end{matrix}}_{\Delta\hat{u}_{\mathrm{f}_7}\text{-}\Delta\hat{u}_{\mathrm{f}_{10}}} & \underbrace{\begin{matrix} DJ_{129} & \cdots & DJ_{132} \\ DJ_{229} & \cdots & DJ_{232} \end{matrix}}_{\Delta\hat{v}_{\mathrm{f}_7}\text{-}\Delta\hat{v}_{\mathrm{f}_{10}}} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \overline{v}_{\mathrm{f}_7} \\ \overline{v}_{\mathrm{f}_8} \\ \overline{v}_{\mathrm{f}_9} \\ \overline{v}_{\mathrm{f}_{10}} \end{pmatrix} = \begin{pmatrix} b_{\mathrm{bf},1} \\ b_{\mathrm{bf},2} \end{pmatrix}, \tag{5.33}$$

and are assembled in the residuum:

$$\underline{b} = \begin{pmatrix} \cdots \\ \cdots \\ b_{\mathrm{bf},1} \\ b_{\mathrm{bf},2} \\ \vdots \end{pmatrix}. \tag{5.34}$$

The subscript $(.)_{\mathrm{bf}}$ refers to *boundary face*. The nonlinear diffusion flux term, also called residuum in the denotion of the NR procedure, in Equation (5.28) may look for cell 2 as follows:

$$\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_2 = \begin{pmatrix} J_1 \\ J_2 \end{pmatrix} = -\begin{pmatrix} b_{\mathrm{n},1} \\ b_{\mathrm{n},2} \end{pmatrix} = -(\underline{b}_{\mathrm{n}})_2\,, \tag{5.35}$$

and, as can be seen, is denoted as *nonlinear* contribution $\underline{b}_{\mathrm{n}}$. The term in Equation (5.35) is assembled as before:

$$\underline{b} = \begin{pmatrix} \cdots \\ \cdots \\ b_{\mathrm{bf},1} + b_{\mathrm{n},1} \\ b_{\mathrm{bf},2} + b_{\mathrm{n},2} \\ \vdots \end{pmatrix}. \tag{5.36}$$

Therefore, the boundary face and nonlinear contribution result in the residuum $\underline{b}$.

Treatment of Dirichlet BCs in the NRM

Similar to the FEM, where in the NR solution procedure the prescribed nodal values are set in the first iteration step and set to zero for subsequent iteration steps, also in the proposed solution process, the displacement values of prescribed boundary faces are treated in a similar way.

One has to be aware, that in SOOFVAM there are two different displacement `properties` for both cells and faces: on the one hand the *incremental* displacement and on the other hand the displacement *value*. For *cells*, the incremental displacement is the calculated primary variable, while the displacement value is updated for each iteration step as described above. For *faces*, they have a different meaning. The incremental displacement of the face is set in the first iteration step due to the prescribed BC. When the Dirichlet BCs are assembled into $\underline{b}$ (cf. Equation (5.33) and Algorithm 5.13), the displacement value of the faces are set to the prescribed value and the incremental displacement is set to zero. This is because after the first solution the faces have reached the prescribed displacement value.

The final form of the set of linear equations then reads

$$(A^j_k)_t\;(\Delta\hat{\tilde{u}}^k)_{t+1} = (b^j_{\mathrm{n}})_t + (b^j_{\mathrm{bf}})_t = (b^j)_t \tag{5.37}$$

and can now be solved for the incremental displacements $(\Delta\hat{\tilde{u}}^k)_{t+1}$ in each iteration step t.

The coherences described above are summarised in Algorithm 5.10, Algorithm 5.11, Algorithm 5.12 and Algorithm 5.13.

Algorithm 5.10 Workflow of FV Solution Procedure

```
1: function solveFVSystem(noc, nod, nov)
2:     initialise $\underline{\underline{A}}_t$ [nod × nod]
3:     initialise $\underline{b}_t$ [nod × 1]

4:     for each cell in model do
5:         initialise $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{C}}$ [$dim \times (\mathtt{nov} \cdot dim)$]
6:         initialise $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{C}}$ [$dim \times 1$]

7:         for each face of cell do
8:             if neighbour of face is not empty or
                 neighbour of face is empty and BC is prescribed then
9:                 $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{f}}$, $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{f}}$ ← call
                   calcDiffusionTermNonlinear(cell, face, noc, nov)
10:                $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{C}} += \left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{C}} + \left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{f}}$
11:                $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{C}} += \left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{C}} + \left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{f}}$
12:            end if
13:        end for

14:        assemble $\underline{\underline{A}}_t$ ← call
           assembleMatrixA($\underline{\underline{A}}$, cell, noc, nov, $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{C}}$)
                                                                  ▷ Algorithm 5.12
15:        assemble $\underline{b}_t$ ← call
           assembleVectorb($\underline{b}$, cell, nov, $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{C}}$, $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{C}}$)
                                                                  ▷ Algorithm 5.13
16:    end for

17:    assemble $\underline{b}_t$ ← call integrateNeumannBCandSourceTerm($\underline{b}$)   ▷ where required
18:    calculate increment $\underline{\Delta u}_{t+1} = \underline{\underline{A}}_t^{-1}\underline{b}_t$

19:    return increment $\underline{\Delta u}_{t+1}$, residuum $\underline{b}_t$
20: end function
```

Algorithm 5.11 Calculation of Diffusion Flux Terms

```
1: function calcDiffusionTermNonlinear(noc, nov)
2:     $(\underline{\underline{T}})^*_{\mathrm{b},i}$ or $(\underline{\underline{T}})^*_{\mathrm{f}}$ ← call getGradientCoefficientMatrix(cell, face, noc, nov)
3:     $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{b}}$ or $\left({}^{\mathrm{ex}}\underline{\nabla}^0\underline{u}\right)^*_{\mathrm{f}}$ ← call calcGradientOfFace(cell, face)  ▷ Section 5.2.4
4:     initialise $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{f}}$ $[dim \times (\mathtt{nov} \cdot dim)]$
5:     initialise $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{f}}$ $[dim \times 1]$
6:     calculate $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{f}}$, $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{f}}$    ▷ Equation (4.21), Equation (4.26)
7:     if cell is not owner of face then
8:         $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{f}} = \left(-D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{f}}$, $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{f}} = -\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{f}}$
9:     end if

10:    return $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{f}}$, $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{f}}$
11: end function
```

Algorithm 5.12 Assembly of $\underline{\underline{A}}$

```
1: function assembleMatrixA($\underline{\underline{A}}$, cell, noc, nov, $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{C}}$)
2:     determine rows corresponding to the cell's global index
3:     determine columns corresponding to the cell's displacement values
4:     assemble $\underline{\underline{A}}$

5:     return $\underline{\underline{A}}$
6: end function
```

Algorithm 5.13 Assembly of $\underline{b}$ and Treatment of Dirichlet BCs

```
1:  function assembleVectorb($\underline{b}$, cell, nov, $\left(D\underline{J}^{\underline{u},D}(\chi_t)[\underline{\Delta u}]\right)_{\mathrm{C}}$, $\left(\underline{J}^{\underline{u},D}(\chi_t)\right)_{\mathrm{C}}$)
2:      determine rows corresponding to the cell's global index

3:      for each face of cell do
4:          if DOF in face is prescribed and value $\neq 0$ then
5:              get boundary face incremental displacement $\underline{\Delta u}_{\mathrm{b}}$
6:              if iteration step $t = 1$ then                              ▷ first step in NRM
7:                  set boundary face displacement $\underline{u}_{\mathrm{b}} \leftarrow \underline{\Delta u}_{\mathrm{b}}$
8:                  set $\underline{\Delta u}_{\mathrm{b}} \leftarrow \underline{0}$
9:              end if
10:             determine columns corresponding to the boundary face's global index
11:             calculate boundary face contribution $\underline{b}_{\mathrm{bf}}$              ▷ Equation (5.33)
12:             assemble $(\underline{b})_{\mathrm{bf}}$ in $\underline{b}$                               ▷ Equation (5.34)
13:         end if
14:         assemble $(\underline{b})_{\mathrm{n}}$ in $\underline{b}$                                    ▷ Equation (5.36)
15:     end for

16:     return $\underline{b}$
17: end function
```

6. Results of the Simulation

In this chapter the proposed algorithm, and thus the implemented code SOOFVAM, is applied to different cases both for small and large deformations. In order to cover as many facets as possible, the following test cases were developed:

- ▷ performance test of mesh topology generation (Section 6.1),
- ▷ comparison of performance with different software for small deformation theory (Section 6.2),
- ▷ behaviour of convergence of the iterative solution procedure (Section 6.3),
- ▷ comparison of performance with different software for large deformation theory (Section 6.4),
- ▷ influence of parameters and type of mesh (Section 6.5), and
- ▷ performance of large deformations in a 3D case (Section 6.6).

The test cases were executed using the "R2020a" release of MATLAB© in its Student's version on the author's personal computer, an *acer Aspire E* 15 laptop (OS: MS Windows 10). The hardware consists of

- ▷ an Intel© Core™ i7-4510U processor with a minimal clock frequency of 2.0 GHz
- ▷ 8 GB DDR3L RAM.

For the solution of the equation system $\underline{\underline{A}}\ \Delta\underline{\hat{\hat{u}}} = \underline{b}$, the *backslash* operator (or `mldivide`) was used. Besides the distinction between *full/dense* or *sparse* input arrays, the in-built MATLAB© algorithm determines whether the matrix is square, diagonal, triangular or Hermetian. It searches the matrix for symmetries and dispatches it to an appropriate solver [34]. The equation systems of the test cases in the subsequent sections are solved via UMFPACK V5.4.0[1] [34], which is a set of routines for solving unsymmetric sparse linear systems implemented in MATLAB© [12].

6.1. Performance of Mesh Generation

In the first test case (TC) the performance of the mesh generation algorithm was investigated.

As mentioned in Section 5.1.2, an additional routine (cf. Algorithm 5.1) is required to perform the analysis. In Line 4 of Algorithm 5.1 the surrounding neighbour cells for each cell of the computational domain are determined, before the topology is created. This

[1] Unsymmetric MultiFrontal package

information is not provided by the `.msh` file created with `Gmsh`© because this information is not needed for FE calculations. Algorithm 5.1 is designed specifically for unstructured grids and covers meshes for both 2D and 3D computational domains.

For the tests, a simple geometry as illustrated in Figure 6.1 was chosen. The mesh generation time over cells per side was compared to the in-house FE solver SOOFEAM and the simulation results are shown in Figure 6.2. As might be expected, the mesh generation time multiplies as the number of cells per side rises.

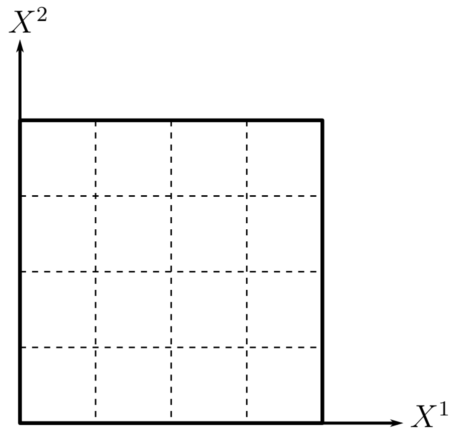


Figure 6.1.: TC1 - Computational Domain

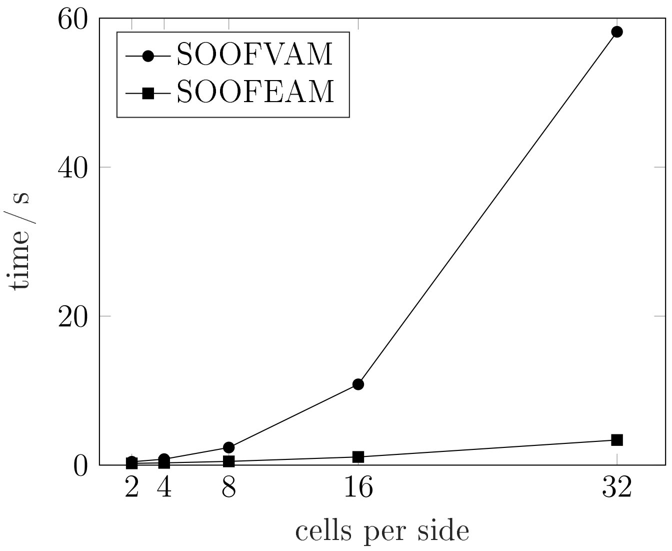


Figure 6.2.: TC1 - Comparison of Mesh Generation Time (Create Topology)

In case of problems in a two-dimensional domain, the proposed algorithm may be avoided because it is not mandatory to determine the surrounding cells. One can create the topology just with the nodal information provided by `Gmsh`©. Although, for the 3D case this is not possible as the nodes can be arbitrarily ordered in the cell. Therefore, it is necessary to determine which node is located where and with which nodes it forms a face. From the author's point of view, a simple solution is the proposed routine in Algorithm 5.1. Hence, the author decided to maintain the proposed approach and implement the routine in SOOFVAM.

The FE solver SOOFEAM generates the mesh anew for each simulation, since only the created `.msh` file is read in. Since this approach is not promising for SOOFVAM (cf. Figure 6.2), once a topology is created, it can be saved. As long as the topology remains the same, the saved mesh can be loaded for the analysis. In another test, the loading time was compared again to SOOFEAM. Since in SOOFVAM only one variable has to be loaded, normed time values as shown in Figure 6.3 were compared. As one can observe, they show a similar behaviour.

All the time values illustrated in Figure 6.2 and Figure 6.3 are the average of seven simulation runs, neglecting the highest and lowest value[2].

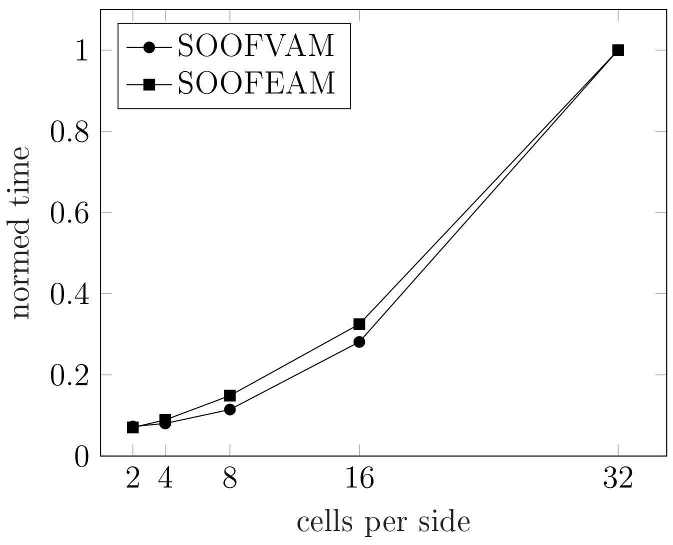


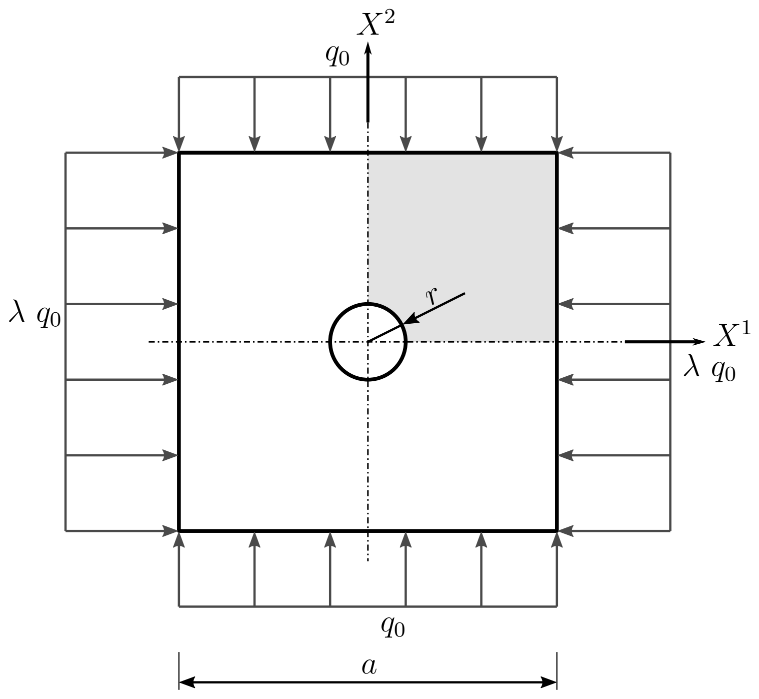
Figure 6.3.: TC1 - Comparison of Mesh Generation Time (Read Topology)

6.2. Comparison of Solvers for Small Deformations

The second test case is meant to verify the results of the proposed implicit algorithm for small deformations by comparing it with different types of software. The setup

[2]This was done in order to exclude outliners. Therefore, the times are an arithmetic average out of five values.

of the test case with details about geometry and loading is shown in Figure 6.4. The analysis comprises a finite plate with a circular hole. *Kirsch* [29] formulated an analytical solution for an *infinte* plate, therefore this setup is also called *Kirsch's problem*. For the simulations, plane stress with a *Young* modulus $E = 2.1\text{e}5\,\text{MPa}$ and a *Poisson* ratio $\nu = 0.3$ were assumed. The *distance* option (cf. Equation (3.29)) was chosen for the geometric weighting factor and the *over-relaxed* approach (cf. Equation (3.32)) for the nonorthogonal treatment.



q_0	$100\,\frac{\text{N}}{\text{mm}}$
λ	1.5
a	1600 mm
r	20 mm

Figure 6.4.: TC2 - Kirsch's Problem

The original proposed analytical equations [6, 29], after a transformation into Cartesian coordinates[3], for the displacement along the circular boundary read

$$\begin{aligned} u &= -\frac{q_0 r}{4E}(1+\nu)(\alpha+1)(3\lambda-1)\cos(\varphi)\,, \\ v &= -\frac{q_0 r}{4E}(1+\nu)(\alpha+1)(3-\lambda)\sin(\varphi)\,, \end{aligned} \tag{6.1}$$

where φ can be calculated as $\varphi = \text{atan2}\left(\frac{y}{x}\right)$, with x and y being the coordinates of the nodes along the circular boundary (cf. Figure 6.5 right), and the factor α, for the plane stress case, reads

$$\alpha = \frac{3-\nu}{1+\nu}\,. \tag{6.2}$$

[3]The original equations describe the coherences in Polar coordinates.

As one may observe, the computational domain consists of two axes of symmetry, therefore the calculations are only carried out for the shaded quarter, cf. Figure 6.4. This is shown in Figure 6.5 with the prescribed BCs and the form of the mesh used for the simulations. The faces on the bottom boundary are constrained in X^2-direction and the faces on the left boundary in X^1-direction.

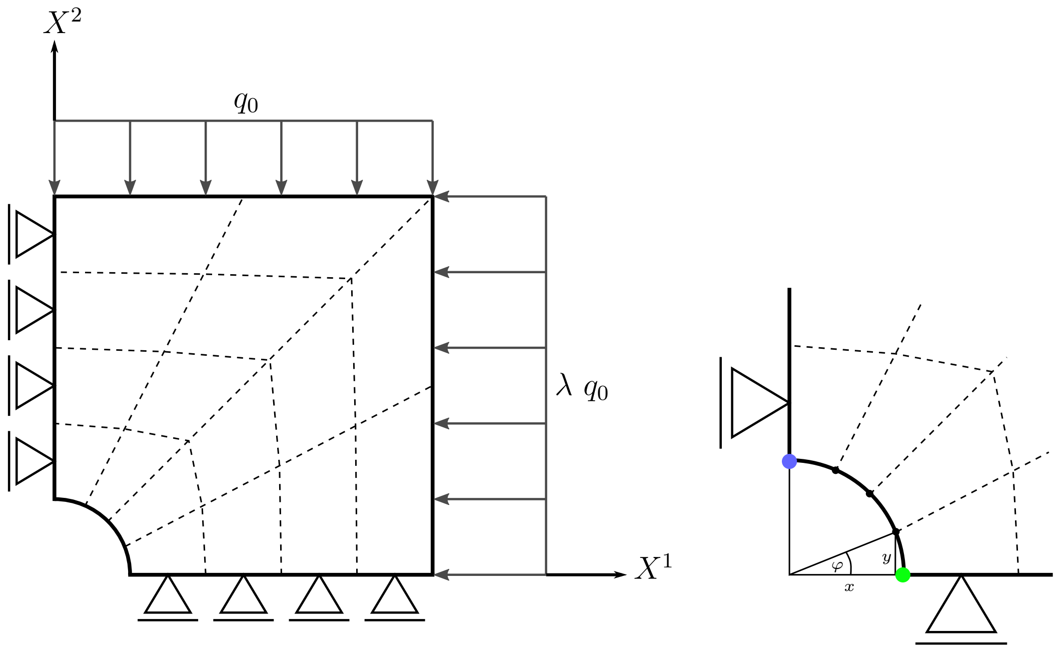


Figure 6.5.: TC2 - Computational Domain and BCs

In a first test, the simulation results were compared to the analytical solution in Equation (6.1). Therefore, the relative error of displacements $\varepsilon_{\underline{u}}^{rel}$ in the nodes along the circular boundary was calculated as

$$\varepsilon_{\underline{u}}^{rel} = \frac{\|\underline{u}_a - \underline{u}_n\|}{\|\underline{u}_a\|}, \tag{6.3}$$

where $\underline{u}_a$ refers to the vector with the analytical displacement values and $\underline{u}_n$ relates to the numerical solution. Different meshes analogous to the one shown in Figure 6.5 were used for the simulations. The coarsest mesh consists of 64 cells and the finest one of 1600. The relative error over an increasing number of cells both for SOOFVAM and SOOFEAM is displayed in Figure 6.6.

As can be observed, the relative error for SOOFEAM is smaller than for SOOFVAM. Note that for the introduced setup, the number of degrees of freedom for the FV solver is always smaller than for the calculations with finite elements. For example, for the mesh with 1600 cells, the computational domain consists of 1681 nodes. In addition, the nodal displacement values are extrapolated for the FV solver. As the analytical solution holds for $a \to \infty$, additional tests may observe the performance for rising length a of the plate.

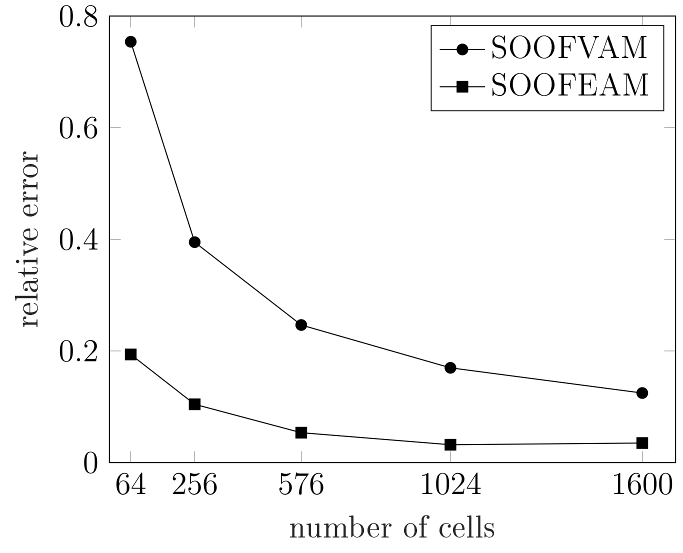


Figure 6.6.: TC2 - Comparison of Relative Error

In another test, the simulation results were compared to the open-source CFD toolbox OpenFOAM© [25, 27, 47]. For the introduced problem, the OpenFOAM© case called `plateHole`, that incorporates a plate of finite length with a circular hole, of the `solidDisplacementFoam` tutorials was used. The displacement values in the *circle bottom*[4] and *circle top*[5] position for a mesh consisting of 1600 cells were compared. The results are shown in Table 6.1 and Table 6.2. As can be observed, the FE solver shows results that come closest to the analytical solution, while SOOFVAM and OpenFOAM© provide similar but less precise results. Since the results only show a specific setup, more tests with different parameters should substantiate the validity of the results.

Table 6.1.: TC2 - Comparison of Displacement Values: Circle Bottom

source	analytical	SOOFEAM	SOOFVAM	OpenFOAM©
u / mm	−0.03333	−0.03091	−0.04173	−0.04009
v / mm	0.00000	0.00000	0.00005	0.00000

The comparison of the analysis time is shown in Figure 6.7. As can be observed, the computation time for all solvers increases with increasing number of cells, whereas OpenFOAM© works significantly faster for a high number of cells. The time values

[4]marked with a green circle in Figure 6.5 right
[5]marked with a blue cirlce in Figure 6.5 right

Table 6.2.: TC2 - Comparison of Displacement Values: Circle Top

source	analytical	SOOFEAM	SOOFVAM	OpenFOAM©
u / mm	0.00000	0.00000	0.00018	0.00000
v / mm	−0.01429	−0.01348	−0.01535	−0.01331

illustrated in Figure 6.7 are the average of seven simulation runs, neglecting the highest and lowest value.

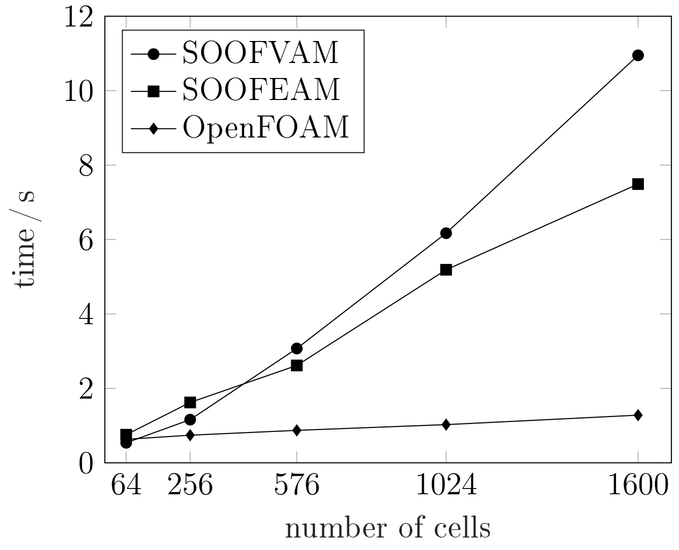


Figure 6.7.: TC2 - Comparison of Analysis Time

6.3. Performance of NR Iteration Convergence

Test case number three is the first case that deals with the theory of large deformations. Details about geometry and the prescribed Dirichlet BCs are shown in Figure 6.8, and the relevant computational domain of the problem is illustrated in Figure 6.9. A *Neo-Hookean* material model and plane stress with a *Young* modulus $E = 20\,\text{MPa}$ and a *Poisson* ratio $\nu = 0.45$ were assumed. As this test case deals with large deformations, an imaginary material with *rubberlike* material parameters was assumed. Due to the structured mesh with orthogonal cells, the choice of the geometric weighting factor[6] and nonorthogonal treatment do not affect the simulation.

[6] $g_C = \frac{1}{2}$ holds true for the whole domain for both the *average* and *distance* option.

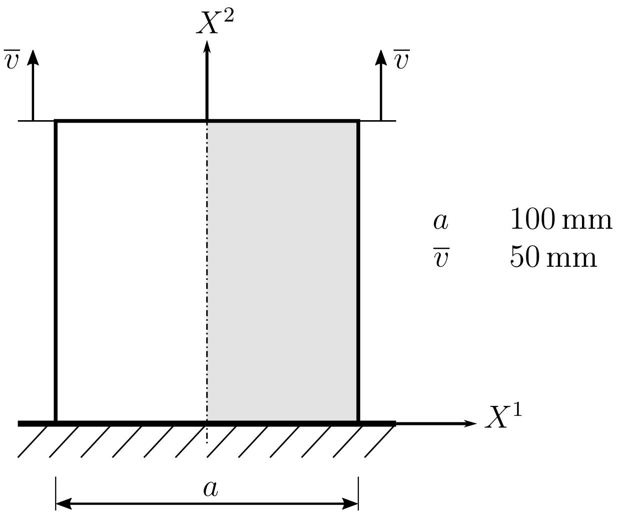


Figure 6.8.: TC3 - Quadratic Plate

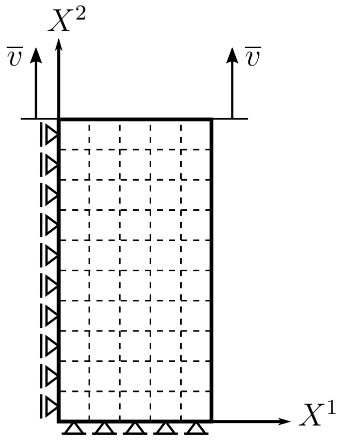


Figure 6.9.: TC3 - Computational Domain and BCs

At first, the convergence behaviour was investigated and compared to the FE solver SOOFEAM. The test was carried out for 20 cells per side[7] and the results of the convergence behaviour are displayed in Figure 6.10 and Figure 6.11. The iterative solution process was stopped when the convergence criterion of 1e-15 for both the normed increment and residuum was reached. The normed values were calculated as

$$value_{normed} = \frac{value_t}{value_{ref}}\,, \tag{6.4}$$

where *value* either relates to the increment or residuum. The subscript $(.)_t$ refers to the iteration step and $(.)_{ref}$ relates to the reference value, which is the increment/residuum

[7]This means 10×20 cells for the computational domain due to symmetry, cf. Figure 6.9.

of the first iteration step. Note that the increment refers to the calculated incremental displacement and the residuum is $\underline{b}$, cf. Equation (5.37). The diagrams in Figure 6.10 and Figure 6.11 depict a similar behaviour with quadratic convergence both for SOOFVAM and SOOFEAM showing that the proposed algorithm may be an equivalent alternative.

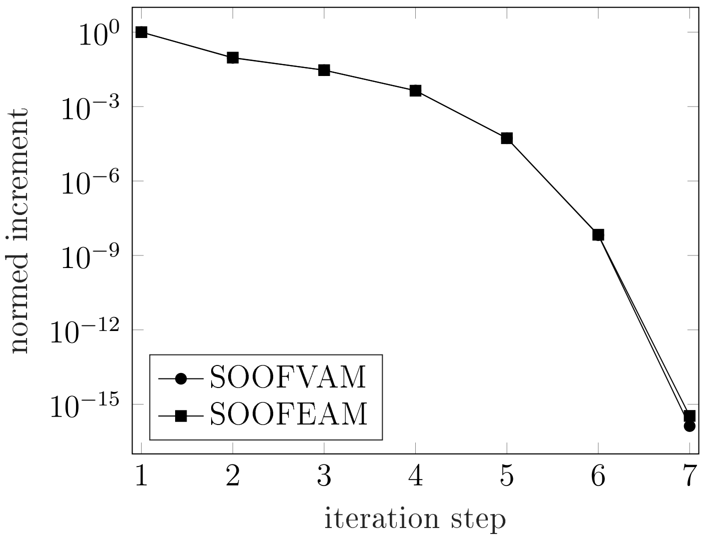


Figure 6.10.: TC3 - Behaviour of Normed Increment

In a second test, the nodal displacement values of the whole computational domain were compared. The relative deviation of the displacements $\phi_{\underline{u}}^{rel}$ to the results of the FE solver was evaluated in the form of

$$\phi_{\underline{u}}^{rel} = \frac{\|\underline{u}_{FEM} - \underline{u}_{FVM}\|}{\|\underline{u}_{FEM}\|}\,, \tag{6.5}$$

where $\underline{u}_{FEM}$ and $\underline{u}_{FVM}$ relate to the vectors of the nodal displacements calculated with SOOFEAM and SOOFVAM, respectively. The results for the relative deviation with rising number of cells per side is illustrated in Table 6.3. As one observes, the relative deviation is very small and declines with rising number of cells/elements, which confirms that the implemented code provides comparable results also for finite deformations.

Table 6.3.: TC3 - Relative Deviation of Displacement

cells per side	10	20	30
$\phi_{\underline{u}}^{rel}$	0.01041	0.00313	0.00157

A last test compared the *von Mises* stress σ_{vM} of the two solvers. σ_{vM} is defined as

$$\sigma_{vM} = \sqrt{\|\underline{\underline{\sigma}}\|^2 - 3 \cdot p^2}\,, \tag{6.6}$$

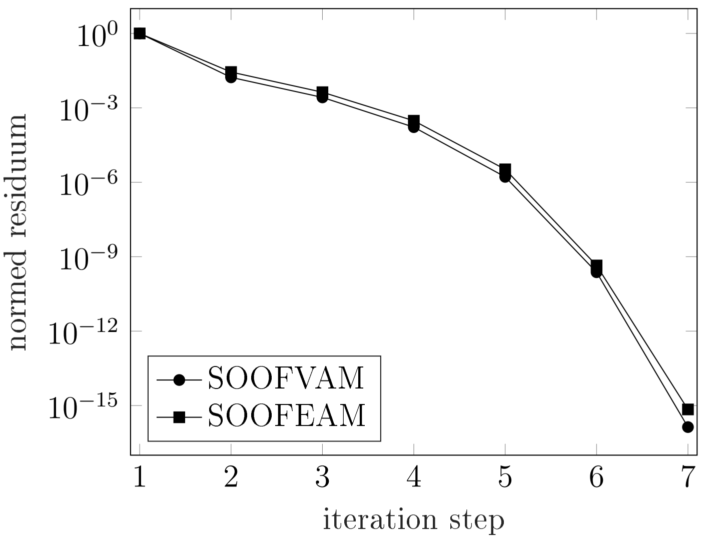


Figure 6.11.: TC3 - Behaviour of Normed Residuum

with the *Cauchy* stress tensor $\underline{\underline{\sigma}}$ and the hydrostatic pressure p calculated as $p = \frac{1}{3}\,\mathrm{tr}(\underline{\underline{\sigma}})$. The difference to the FE solver was determined as relative deviation $\phi^{rel}_{\sigma_{vM}}$, which reads

$$\phi^{rel}_{\sigma_{vM}} = \frac{\|(\underline{\sigma}_{vM})_{FEM} - (\underline{\sigma}_{vM})_{FVM}\|}{\|(\underline{\sigma}_{vM})_{FEM}\|}\,, \tag{6.7}$$

where $\underline{\sigma}_{vM}$ refers to a vector with the stresses for each cell/element of the computational domain. The results of the relative deviation for the *von Mises* stress are displayed in Table 6.4.

Table 6.4.: TC3 - Relative Deviation of von Mises Stress

cells per side	10	20	30
$\phi^{rel}_{\sigma_{vM}}$	0.02947	0.01073	0.00784

The comparison of the analysis time is shown in Figure 6.12. For the introduced setup, the proposed algorithm is nearly twice as fast as the FE code. Both solvers show an increasing computation time for an increasing number of cells. The time values illustrated in Figure 6.12 are the average of seven simulation runs, neglecting the highest and lowest value.

A visualisation of the deformed configuration of the problem for a mesh with 900 cells is shown in Figure 6.13.

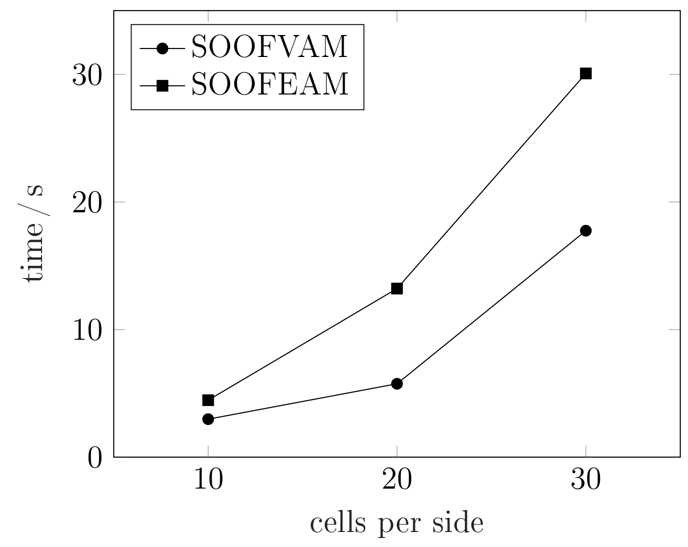


Figure 6.12.: TC3 - Comparison of Analysis Time

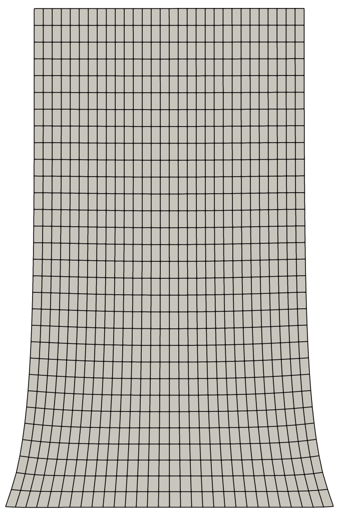
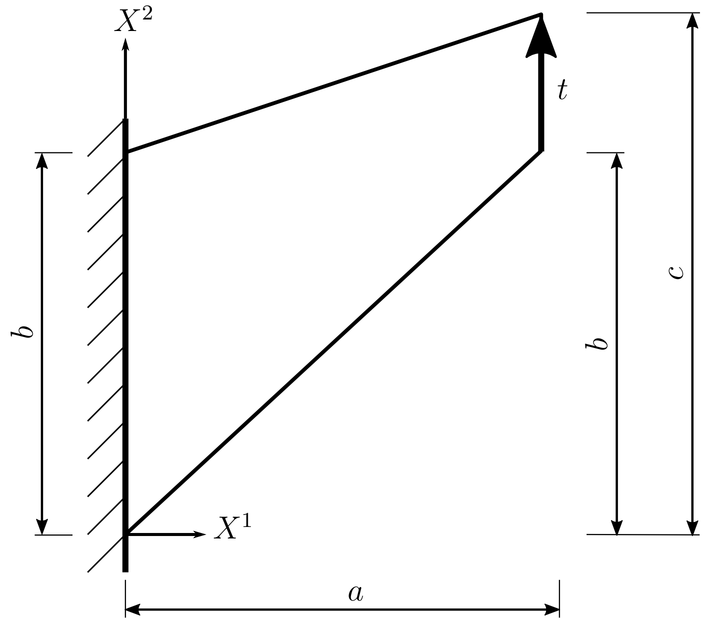


Figure 6.13.: TC3 - Visualisation of Deformed Configuration

6.4. Comparison of Solvers for Large Deformations

The fourth test case compares the proposed large deformation algorithm to the FV solver foam-extend. The open-source software foam-extend [11] is the community edition of the OpenFOAM© library, and it integrates contributions from users and developers, who also helped to find bugs and extended the code base [48]. The solver called `elasticNonLinTLSolidFoam` was used for the test case. It is a solver for large deformations in total *Lagrangian* formulation using the *St. Venant-Kirchhoff material* model. Plane stress with a *Young* modulus $E = 2.1\text{e}5\,\text{MPa}$ and a *Poisson* ratio $\nu = 0.3$ were assumed for the simulations. Details about geometry and the loading are shown in Figure 6.14. The convergence criterion for the NRM was set to 1e-10 for this test. The *distance* option (cf. Equation (3.29)) was chosen for the geometric weighting factor and the *modified* approach (cf. Equation (3.33)) for the nonorthogonal treatment. The computational domain with the prescribed BCs and the form of the used mesh is illustrated in Figure 6.15.



t	$200\,\frac{\text{N}}{\text{mm}}$
a	$480\,\text{mm}$
b	$440\,\text{mm}$
c	$600\,\text{mm}$

Figure 6.14.: TC4 - Cook's Membrane

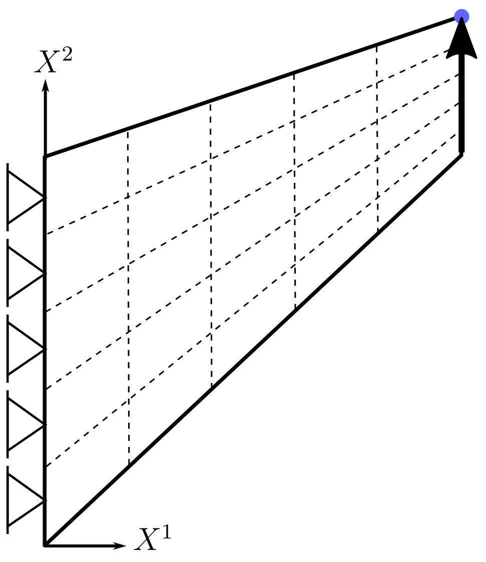


Figure 6.15.: TC4 and TC5 - Computational Domain

The investigated test case comprises the so-called *Cook's membrane* (cf. Figure 6.14) problem, for which no analytical solution exists. Therefore, the relative deviation of the cell displacement values of SOOFVAM to the numerical results obtained with foam-extend was determined. The relative deviation of displacements $\phi_{\underline{u}}^{rel}$ reads

$$\phi_{\underline{u}}^{rel} = \frac{\|\underline{u}_{sv} - \underline{u}_{fme}\|}{\|\underline{u}_{fme}\|}, \tag{6.8}$$

where $\underline{u}_{sv}$ refers to the cell displacement values of SOOFVAM, while $\underline{u}_{fme}$ relates to the ones obtained with foam-extend.

The results for the relative deviation are shown Table 6.5. Tests were carried out for five different meshes, starting with five cells per side and ending with 25 cells per side. As may be observed, the relative deviation is declining with rising mesh size and becomes very small for the mesh with 625 cells. Hence, test case four confirms that the proposed implicit approach provides equivalent results compared to the *regular* approach of FV solvers for nonlinear elasticity. More tests with different parameters should substantiate the validity of the results.

Table 6.5.: TC4 - Relative Deviation of Displacement

cells per side	5	10	15	20	25
$\phi_{\underline{u}}^{rel}$	0.24933	0.08253	0.03267	0.01231	0.00356

The comparison of the analysis time is shown in Figure 6.16. The time values illustrated in Figure 6.16 are the average of seven simulation runs, neglecting the highest and lowest value. As can be seen, both solvers show an increasing behaviour with increasing number of cells, but the computing time of foam-extend is superior to the proposed algorithm.

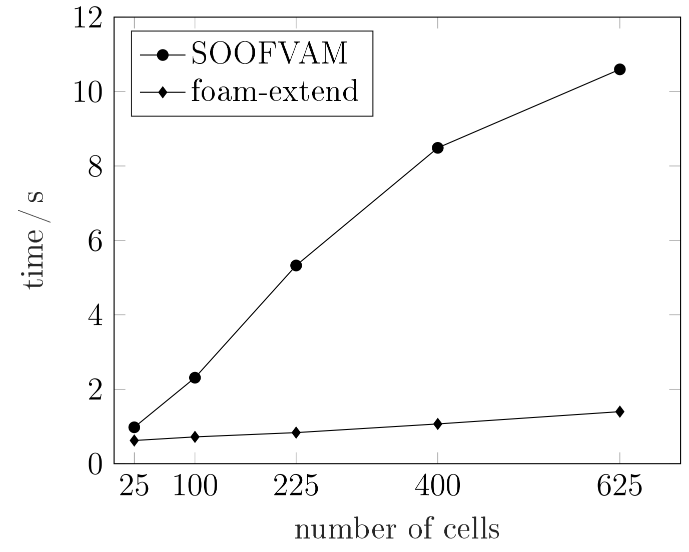


Figure 6.16.: TC4 - Comparison of Analysis Time

6.5. Influence of Parameters and Mesh

Test case number five also comprises the *Cook's membrane* problem with the same BCs as shown in Figure 6.15, but with different material parameters and loading. The loading was set to $t = 0.6\,\mathrm{N/mm}$ and the convergence criterion was prescribed with 1e-10. A *Neo-Hookean* material and plane stress with a *Young* modulus $E = 20\,\mathrm{MPa}$ and a *Poisson* ratio $\nu = 0.45$ were assumed. This should model a compressible *rubberlike* material. The results of the nodal displacement in the top right corner, marked with a blue circle in Figure 6.15, were compared for four different meshes, cf. Figure 6.17. Furthermore, the influence of the adjustable parameters, geometric weighting factor and nonorthogonal treatment, on different types of meshes is discussed below.

One can observe the results for the nonorthogonal structured mesh illustrated in Figure 6.17 (a) in Table 6.6. The computational domain consists of 20 cells per side resulting in 400 cells. As can be seen, within the *modified* approach there are the smallest differences between the options *distance* and *average*. Moreover, the *over-relaxed* and *modified* approach provide similar results, as both approaches share similarities in the consideration of the skewness of the mesh. In contrast, the *minimum correction* approach minimises the influence of the nonorthogonality.

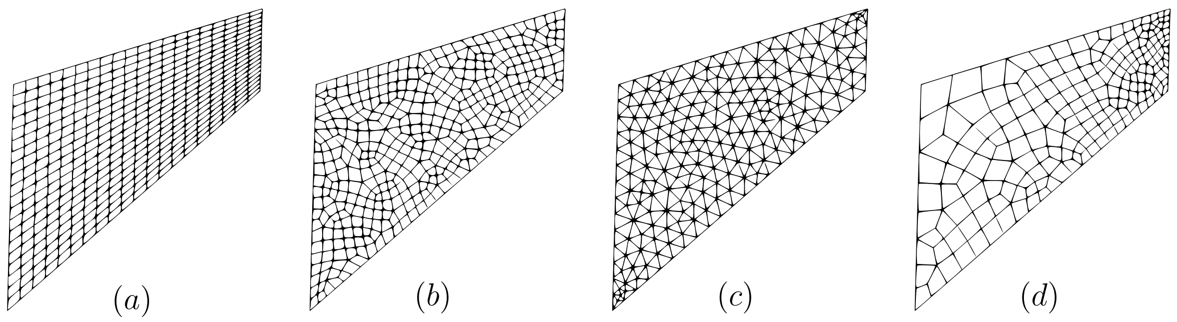


Figure 6.17.: TC5 - Nonorthogonal Grids: (*a*) Structured, (*b*) Unstructured Quadrilateral, (*c*) Unstructured Triangular, (*d*) Unstructured Quadrilateral Refined

Table 6.6.: TC5 - Displacement Values for Nonorthogonal Structured Grid

nonorthogonal treatment	geometric weighting factor	
minimum correction	*distance*	*average*
u / mm	−81.6803	−81.4446
v / mm	91.1311	90.9662
over-relaxed		
u / mm	−79.9770	−79.4603
v / mm	90.4033	90.0307
modified		
u / mm	−79.9402	−79.9112
v / mm	90.2595	90.2489

In Table 6.7 the numerical results for the quadrilateral unstructured grid (Figure 6.17 (*b*)) are displayed. The mesh consists of 397 cells to ensure comparability to the greatest extent possible. The differences between the different nonorthogonal treatment approaches are even smaller compared to Table 6.6.

Table 6.7.: TC5 - Displacement Values for Nonorthogonal Unstructured Quadrilateral Grid

nonorthogonal treatment	geometric weighting factor	
minimum correction	*distance*	*average*
u / mm	−81.6917	−81.7240
v / mm	91.5815	91.6033
over-relaxed		
u / mm	−81.2973	−81.3143
v / mm	91.3347	91.3457
modified		
u / mm	−81.4387	−81.3759
v / mm	91.4026	91.3790

The simulation results for the triangular unstructured grid (Figure 6.17 (*c*)) consisting of 396 cells are illustrated in Table 6.8. The calculated displacement values show a similar behaviour within a nonorthogonal treatment approach as described above. As can be observed, the simulation results in larger displacement values compared to the quadrilateral meshes. It is assumed that this is due to the higher skewness between cells in triangular meshes. More tests should investigate this behaviour.

In a last test the results for the mesh displayed in Figure 6.17 (*d*) were investigated. It consists of a coarse mesh which is only refined at the boundary of the prescribed Neumann BC and consists of 203 cells. The results for this mesh can be observed in Table 6.9. Although the mesh consists of only about half of the cells, the displacement values do not differ significantly from the results above. As a result, the option to refine the mesh at critical areas and keep it as coarse as possible[8] in the remaining computational domain should be kept in mind.

[8]for reasons of computational performance

Table 6.8.: TC5 - Displacement Values for Nonorthogonal Unstructured Triangular Grid

nonorthogonal treatment	geometric weighting factor	
minimum correction	*distance*	*average*
u / mm	−88.7912	−87.9126
v / mm	94.2502	93.9579
over-relaxed		
u / mm	−88.0140	−87.2257
v / mm	94.0950	93.7954
modified		
u / mm	−87.8447	−87.3325
v / mm	94.0143	93.8300

Table 6.9.: TC5 - Displacement Values for Nonorthogonal Unstructured Refined Grid

nonorthogonal treatment	geometric weighting factor	
minimum correction	*distance*	*average*
u / mm	−82.1730	−81.6790
v / mm	91.3665	90.8216
over-relaxed		
u / mm	−81.8847	−81.2910
v / mm	91.3072	90.6822
modified		
u / mm	−81.3065	−81.3592
v / mm	90.6281	90.6949

A visualised result of the deformed configuration of the problem for the mesh in Figure 6.17 (*d*) is shown in Figure 6.18.

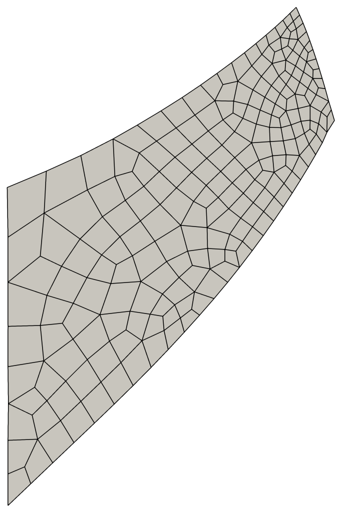


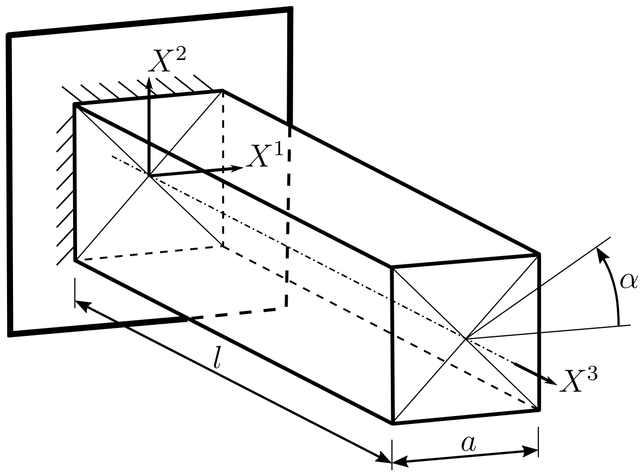
Figure 6.18.: TC5 - Visualisation of Deformed Configuration

6.6. Three-Dimensional Twist

The sixth and last test case covered in this thesis deals with a 3D case. The problem consists of a fixed cantilever that is twisted on the free end. Details about the geometry and the BCs are shown in Figure 6.19. The twist is realised by incrementally prescribing an angle $d\alpha$ until the *full* angle α is reached. In order to obtain this, the previous approach of the total *Lagrangian* description was changed to an updated *Lagrangian* formulation with a pseudotime loop for the incremental BCs, which is not discussed here. The implemented routine is illustrated in Algorithm 6.1. For the simulations, a *Young* modulus $E = 20\,\mathrm{MPa}$ and a *Poisson* ratio $\nu = 0.4$ were assumed. The constitutive law comprises the *Neo-Hookean* material model. The *distance* option (cf. Equation (3.29)) was chosen for the geometric weighting factor and the *over-relaxed* approach (cf. Equation (3.32)) for the nonorthogonal treatment. For the convergence criteria, 1e-8 was chosen.

In order to verify the results, the relative deviation of nodal displacements $\phi_{\underline{u}}^{rel}$ to the results of the FE solver was evaluated in the form of

$$\phi_{\underline{u}}^{rel} = \frac{\|\underline{u}_{FEM} - \underline{u}_{FVM}\|}{\|\underline{u}_{FEM}\|}\,. \tag{6.9}$$



a	120 mm
l	300 mm
α	π rad

Figure 6.19.: TC6 - Three-Dimensional Twist of a Cantilever

The results of the simulation for three different types of meshes and seven time steps are illustrated in Table 6.10.

Table 6.10.: TC6 - Relative Deviation of Displacement

mesh	$8 \times 8 \times 20$	$10 \times 10 \times 25$	$12 \times 12 \times 30$
$\phi_{\underline{u}}^{rel}$	0.03489	0.03155	0.02921

Furthermore, the relative deviation of the *von Mises* stress $\phi_{\sigma_{vM}}^{rel}$ for the cells/elements was compared. It reads

$$\phi_{\sigma_{vM}}^{rel} = \frac{\|(\underline{\sigma}_{vM})_{FEM} - (\underline{\sigma}_{vM})_{FVM}\|}{\|(\underline{\sigma}_{vM})_{FEM}\|} . \tag{6.10}$$

The results are displayed in Table 6.11.

Table 6.11.: TC6 - Relative Deviation of von Mises Stress

mesh	$8 \times 8 \times 20$	$10 \times 10 \times 25$	$12 \times 12 \times 30$
$\phi_{\sigma_{vM}}^{rel}$	0.12313	0.11155	0.10181

The higher deviation in comparison with the 2D cases is due to the updated *Lagrangian* approach, since the updating of the geometry depends on updated nodal coordinates, which are also extrapolated for three-dimensional problems. It also has a greater impact on the *von Mises* stress, since for the displacements a higher number of values[9] is

[9]a number equal to the number of degrees of freedom, which is calculated as three times number of nodes

compared than for the *von Mises* stress[10]. More tests with three-dimensional problems should investigate the performance of the proposed algorithm.

The deformed configuration of the problem for a mesh consisting of 4320 cells visualised with `Paraview`© is illustrated in Figure 6.20.

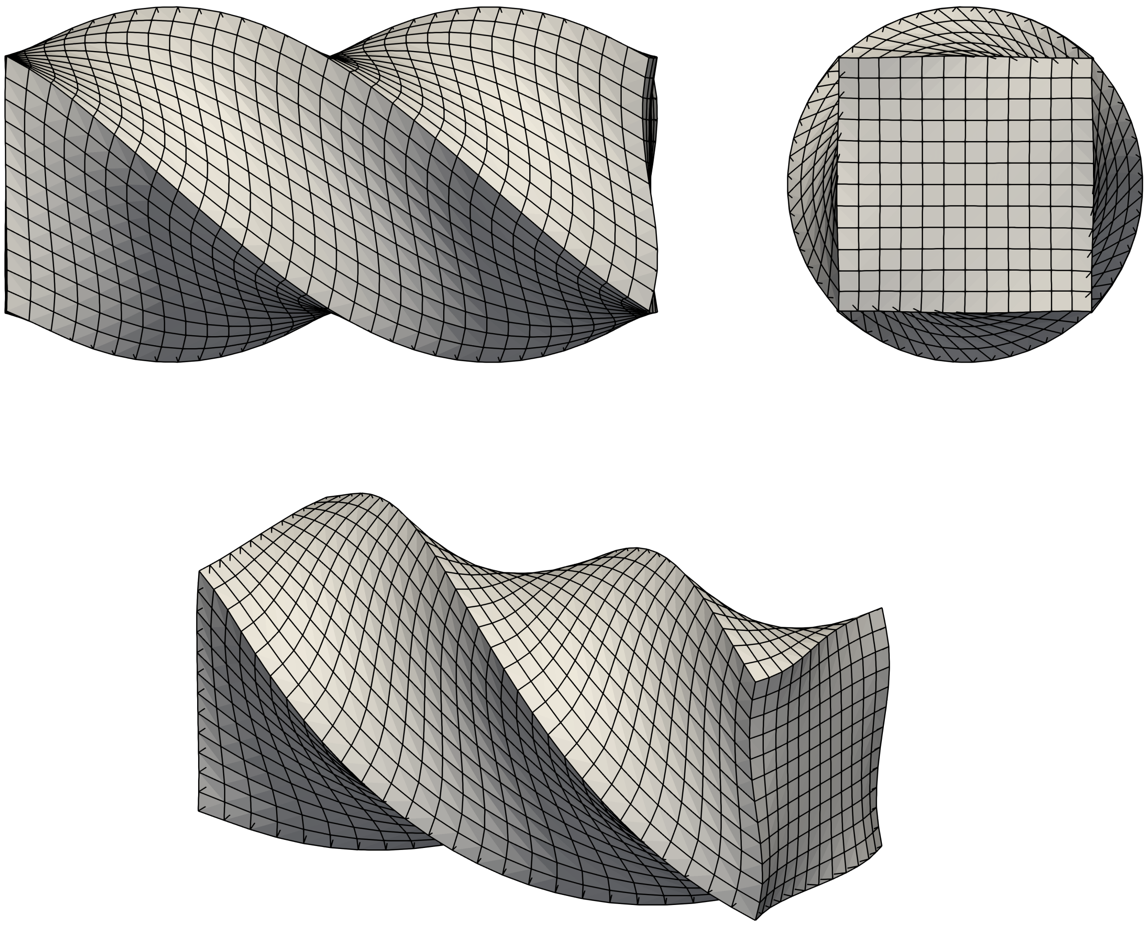


Figure 6.20.: TC6 - Visualisation of Deformed Configuration

[10]a number equal to the number of cells

Algorithm 6.1 Workflow of Updated Lagrangian Solution Procedure

```
function solveEquationSystem(analysis, noc, nod)
    for time step $\tau = 0$ to $\tau_{max}$ do                  ▷ time loop for quasistatic approach
        reset displacement field of cells $\underline{u} = \underline{0}$   ▷ reset displacements of last time step
        call incorporateBC(BCHandler)            ▷ prescription of incremental angle $d\alpha$
        if $\tau = 0$ then
            calculate nov ← number of variables
        end if

        for each cell do
            call calcGlobalCM(cell)
        end for

        for iteration step $t = 1$ to $t_{max}$ do                ▷ loop for NR iteration procedure
            inc, res ← call solveFVSystem(noc, nod, nov)
            if $t = 1$ then
                set reference increment ref_inc ← inc
                set reference residuum ref_res ← res
            end if
            update displacement vector $(\hat{\tilde{u}}^k)_{t+1} = (\hat{\tilde{u}}^k)_t + (\Delta\hat{\tilde{u}}^k)_{t+1}$
            calculate normed increment normed_inc ← $\frac{\texttt{inc}}{\texttt{ref\_inc}}$
            calculate normed residuum normed_res ← $\frac{\texttt{res}}{\texttt{ref\_res}}$
            if normed_inc and normed_res < convergence criteria then
                break
            else if $t = t_{max}$ then
                error: “convergence not reached”
            end if
        end for

        update nodal values                      ▷ via interpolation and extrapolation
        calculate stresses
        update Geometry                          ▷ Updated Lagrangian approach
        write .vtk file                          ▷ displacements of current configuration
    end for
end function
```

7. Conclusion and Outlook

The task of this thesis was to develop an implicit finite volume (FV) solver for nonlinear solid mechanics. MATLAB$^{©}$ served as programming environment since the in-house finite element (FE) code SOOFEAM, which is also used in teaching, was referred to as benchmark for early implementations. The biggest task besides the implementation was the development of the implicit governing equations, since the regular approach in the FVM for solids is a deferred correction approach.

The proposed algorithm, including the calculation of the gradient coefficient matrices with the least-squares method, represents one possible approach. Other approaches may be conceivable, e.g. the gradient computation with the *Green-Gauss* gradient scheme.

The implementation was divided into three steps: At first, an implicit solver for the 2D heat equation, which is not discussed in this thesis, was implemented. The scalar equations of this problem were a good start, as the results were easy to interpret. The main difficulties were the derivation of the gradient coefficient matrices as well as the topology routine for unstructured meshes. In order to *translate* the code into an application for solid mechanics, the next step was an implicit code for the small deformation theory. Numerical results of this solver were discussed in Chapter 6. Major challenges at this point were the correct treatment of Dirichlet boundary conditions, which led to the approach discussed in Section 5.2.4. Finally, the governing equations in total *Lagrangian* formulation as introduced in Section 4.2.1 were developed and implemented. The correct treatment of the BCs in the iterative *Newton-Raphson* (NR) solution procedure (cf. Section 5.2.5) and the development of a neat index notation for the governing equations were the biggest issues to cope with.

In order to verify the proposed algorithm, it was compared to different in-house as well as open-source software:

The first test case (Section 6.1) discussed the mesh generation algorithm. This step is necessary for simulations as the meshes are generated with `Gmsh`$^{©}$, which is primarily a FE mesh generator. The author implemented a routine to save created mesh topology because the generation of computational domains with a large number of cells requires a considerable amount of time. The development of an advanced mesh generation algorithm or the usage of other mesh generation software may be helpful for future applications.

The results of test case two in Section 6.2 already showed that the proposed implicit algorithm provides simulation results equivalent to other solvers including FEM and FVM approaches. The less precise results compared to the FEM are due to the fact that in a quadrilateral mesh of the same size the number of primary variables involved in the equation systems is always smaller for the FV solver than for the FE approach. In addition, a comparison of nodal values can be criticised, since in the proposed FV algorithm these values are linearly interpolated via the values of surrounding cells for interior nodes and extrapolated with the value and gradient of the nearest neighbour cells for boundary nodes. The introduced approach with the relative error for the nodal displacement values may be a good compromise because not a single value is compared. Furthermore, the numerical results were compared to the open-source software OpenFOAM©. The `solidDisplacementFoam` solver of the `stressAnalysis` algorithms showed equivalent results for the introduced problem. Since the code implemented in OpenFOAM© works with the deferred correction approach, the proposed code seems to be an equivalent alternative for the common FV solvers used in solid mechanics.

The convergence performance of the *Newton-Raphson* (NR) iteration procedure for the proposed solving algorithm was discussed in test case three. In this matter, the calculated increments and residua of each iteration step were compared to the in-house FEM code and showed a similar behaviour of quadratic convergence. For the introduced setup in Section 6.3, the proposed algorithm seems to work faster than calculations with SOOFEAM, but it is less precise.

The comparison to the open-source code foam-extend was conducted in Section 6.4. The provided solver called `elasticNonLinTLSolidFoam` was used for test case four. The code for the large deformation theory implemented in foam-extend applies the *St. Venant-Kirchhoff* material model. For the introduced setup both solvers showed equivalent results, which allows the conclusion that also the proposed implicit iterative solution process seems to be a possible alternative.

Test case five (Section 6.5) showed that the choice of the implemented parameters regarding the geometric weighting factor and the nonorthogonal treatment approach has no major influence on the numerical results. Indeed, the mesh has the greatest impact. Not only the size of the cells is decisive but also the choice of the type of the cell. Triangular cells seem to provide larger displacement values than quadrilateral cells for the same loading. More tests should validate this behaviour. As the test case has shown, it may be sufficient to work with a very coarse mesh and refine it in *critical* areas, i.e. an adaptive mesh refinement routine should be considered.

The last test case in Section 6.6 showed that it is also possible to handle three-dimensional problems with the proposed algorithm. If the Dirichlet BCs must be incrementally increased, as in the case of the prescription of a twist, the code must be adjusted to work in the updated *Lagrangian* formulation. Since the deviation to FEM results is higher for three-dimensional problems compared to two-dimensional ones, further tests should verify the performance of the proposed algorithm for three-dimensional problems.

Although the results of the selected test cases show comparable results, there is a certain drawback to call. While in a FE solver the boundary node values are accessed and set *directly* if a Dirichlet BC is prescribed, in the proposed approach one cannot do this due to the cell-centred scheme. Dirichlet BCs are set with the prescription of boundary face values which are not part of the degrees of freedom. Hence, they are added as temporary variables in the equation system. Furthermore, the prescription of the boundary face value only has impact on the cell displacement via the gradient of the boundary face. Another issue is the handling of boundary nodes. While these values are the degrees of freedom for a FE solver, the nodal values are interpolated respectively extrapolated for the proposed approach. Especially the extrapolation process should be approached with caution. The updated *Lagrangian* formulation, which is briefly introduced in Section 6.6 depends on the nodal values, since the geometry of cells and faces is updated via the nodal coordinates of the computational domain. Therefore, it is recommended to keep the ratio of boundary cells to interior cells as low as possible.

This thesis has shown that the methodology of the FVM can be used for the implicit approach with the NR iteration procedure, which is common practice for FE solvers.

From the author's point of view there are different aspects for further steps that can be taken into account. The proposed algorithm only treats the equilibrium state with neglected body forces. Besides the implementation of body forces, the dynamic response of a loading may also be of interest, which makes it necessary to take the neglected transient term into account. Moreover, SOOFVAM currently includes two different hyperelastic materials laws, the *St. Venant-Kirchhoff* and the *Neo-Hookean* material model. Further material models may be added[1] as well as the code may be extended to elastoplastic material behaviour. As it was the first implementation of this scale for the author, there may be improvements in coding possible. Furthermore, a different programming environment might be conceivable as MATLAB© is only one of many alternatives.

Finally, to verify that the proposed algorithm is applicable to various geometries and mesh types, additional test cases must be examined.

[1]e.g. incompressible *Neo-Hookean* or *Mooney-Rivlin* model

Appendix A. UML Diagrams of SOOFVAM

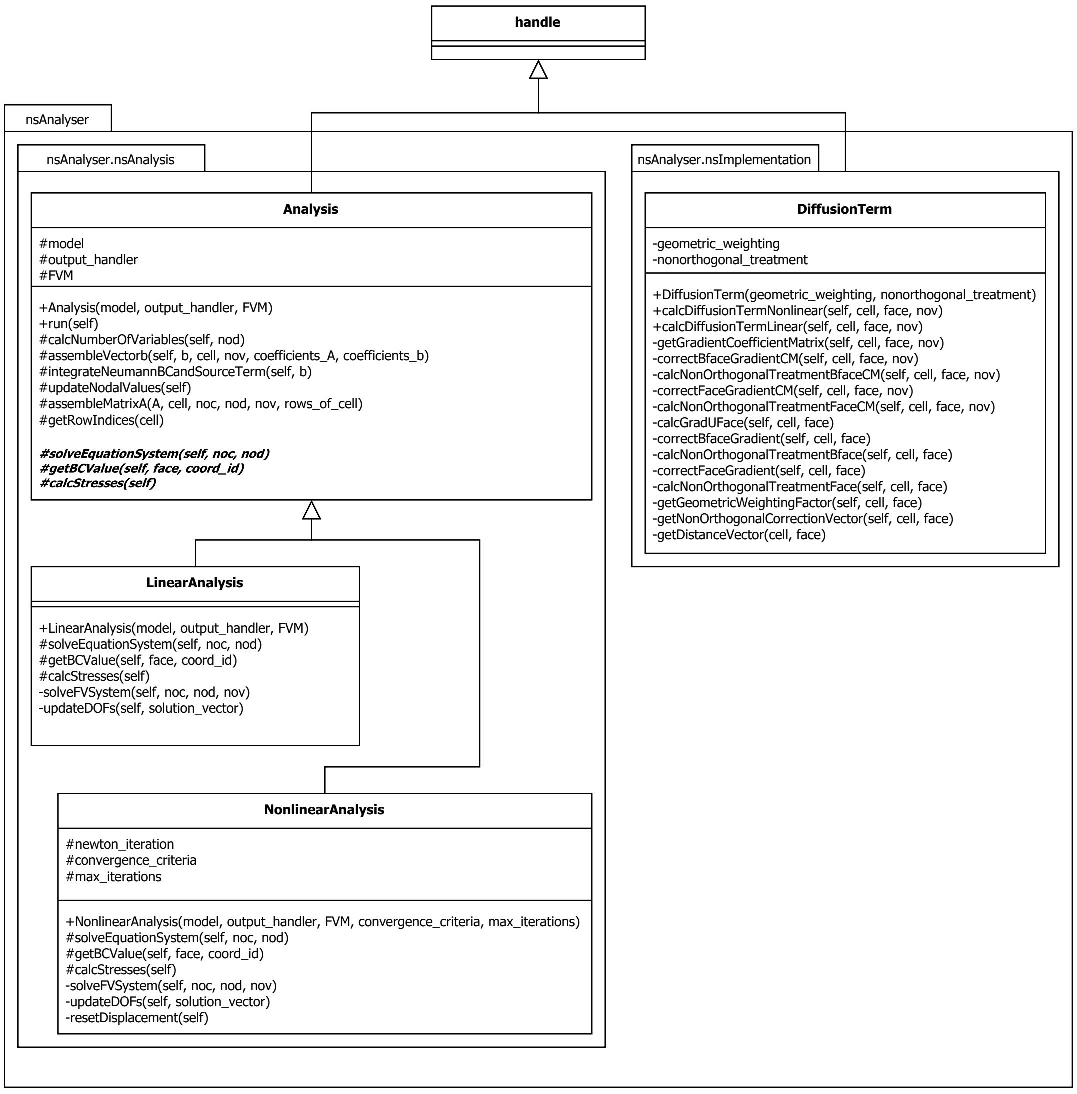

handle
nsAnalyser
nsAnalyser.nsAnalysis
nsAnalyser.nsImplementation
Analysis
#model
#output_handler
#FVM
+Analysis(model, output_handler, FVM)
+run(self)
#calcNumberOfVariables(self, nod)
#assembleVectorb(self, b, cell, nov, coefficients_A, coefficients_b)
#integrateNeumannBCandSourceTerm(self, b)
#updateNodalValues(self)
#assembleMatrixA(A, cell, noc, nod, nov, rows_of_cell)
#getRowIndices(cell)
#solveEquationSystem(self, noc, nod)
#getBCValue(self, face, coord_id)
#calcStresses(self)
LinearAnalysis
+LinearAnalysis(model, output_handler, FVM)
#solveEquationSystem(self, noc, nod)
#getBCValue(self, face, coord_id)
#calcStresses(self)
-solveFVSystem(self, noc, nod, nov)
-updateDOFs(self, solution_vector)
NonlinearAnalysis
#newton_iteration
#convergence_criteria
#max_iterations
+NonlinearAnalysis(model, output_handler, FVM, convergence_criteria, max_iterations)
#solveEquationSystem(self, noc, nod)
#getBCValue(self, face, coord_id)
#calcStresses(self)
-solveFVSystem(self, noc, nod, nov)
-updateDOFs(self, solution_vector)
-resetDisplacement(self)
DiffusionTerm
-geometric_weighting
-nonorthogonal_treatment
+DiffusionTerm(geometric_weighting, nonorthogonal_treatment)
+calcDiffusionTermNonlinear(self, cell, face, nov)
+calcDiffusionTermLinear(self, cell, face, nov)
-getGradientCoefficientMatrix(self, cell, face, nov)
-correctBfaceGradientCM(self, cell, face, nov)
-calcNonOrthogonalTreatmentBfaceCM(self, cell, face, nov)
-correctFaceGradientCM(self, cell, face, nov)
-calcNonOrthogonalTreatmentFaceCM(self, cell, face, nov)
-calcGradUFace(self, cell, face)
-correctBfaceGradient(self, cell, face)
-calcNonOrthogonalTreatmentBface(self, cell, face)
-correctFaceGradient(self, cell, face)
-calcNonOrthogonalTreatmentFace(self, cell, face)
-getGeometricWeightingFactor(self, cell, face)
-getNonOrthogonalCorrectionVector(self, cell, face)
-getDistanceVector(cell, face)

Figure A.1.: nsAnalyser

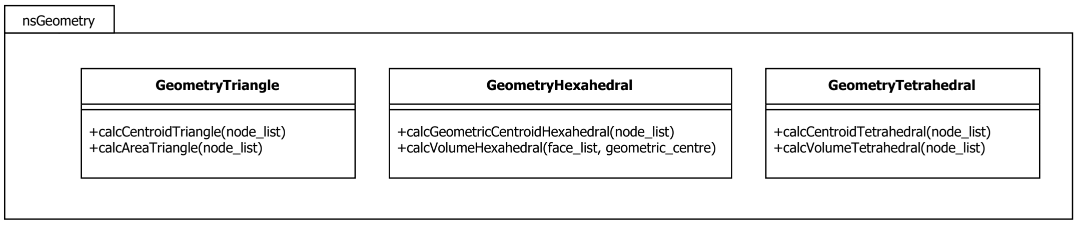

nsGeometry
GeometryTriangle
+calcCentroidTriangle(node_list)
+calcAreaTriangle(node_list)
GeometryHexahedral
+calcGeometricCentroidHexahedral(node_list)
+calcVolumeHexahedral(face_list, geometric_centre)
GeometryTetrahedral
+calcCentroidTetrahedral(node_list)
+calcVolumeTetrahedral(node_list)

Figure A.2.: nsGeometry

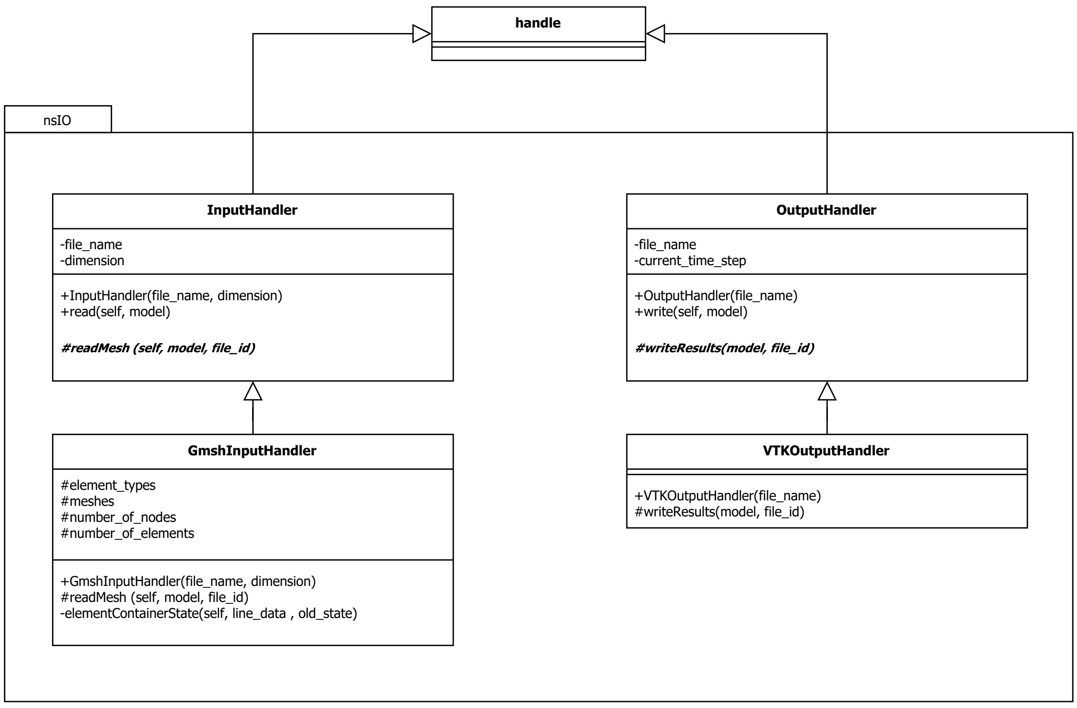

handle
nsIO
InputHandler
-file_name
-dimension
+InputHandler(file_name, dimension)
+read(self, model)
#readMesh (self, model, file_id)
OutputHandler
-file_name
-current_time_step
+OutputHandler(file_name)
+write(self, model)
#writeResults(model, file_id)
GmshInputHandler
#element_types
#meshes
#number_of_nodes
#number_of_elements
+GmshInputHandler(file_name, dimension)
#readMesh (self, model, file_id)
-elementContainerState(self, line_data , old_state)
VTKOutputHandler
+VTKOutputHandler(file_name)
#writeResults(model, file_id)

Figure A.3.: nsIO

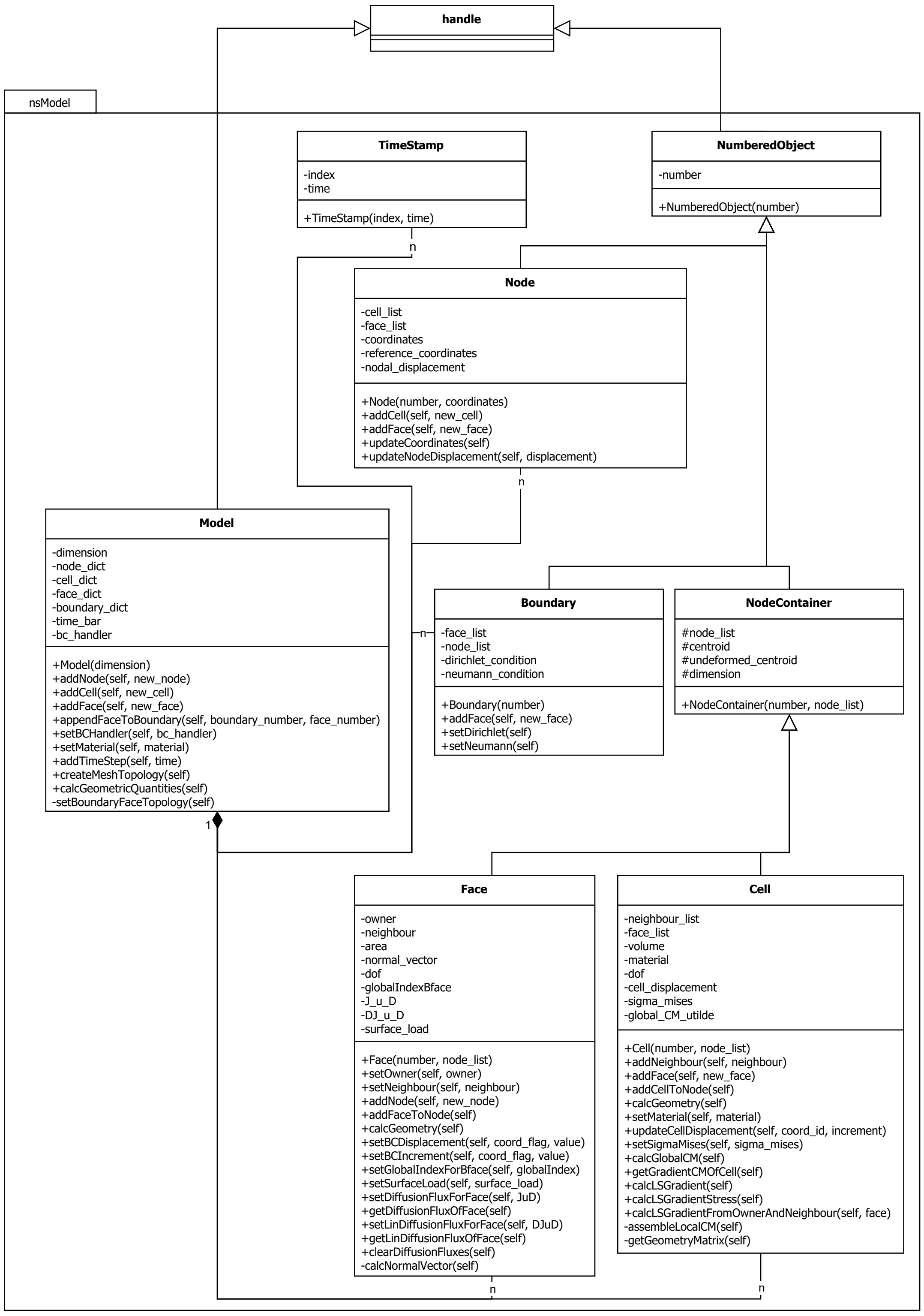


Figure A.4.: nsModel

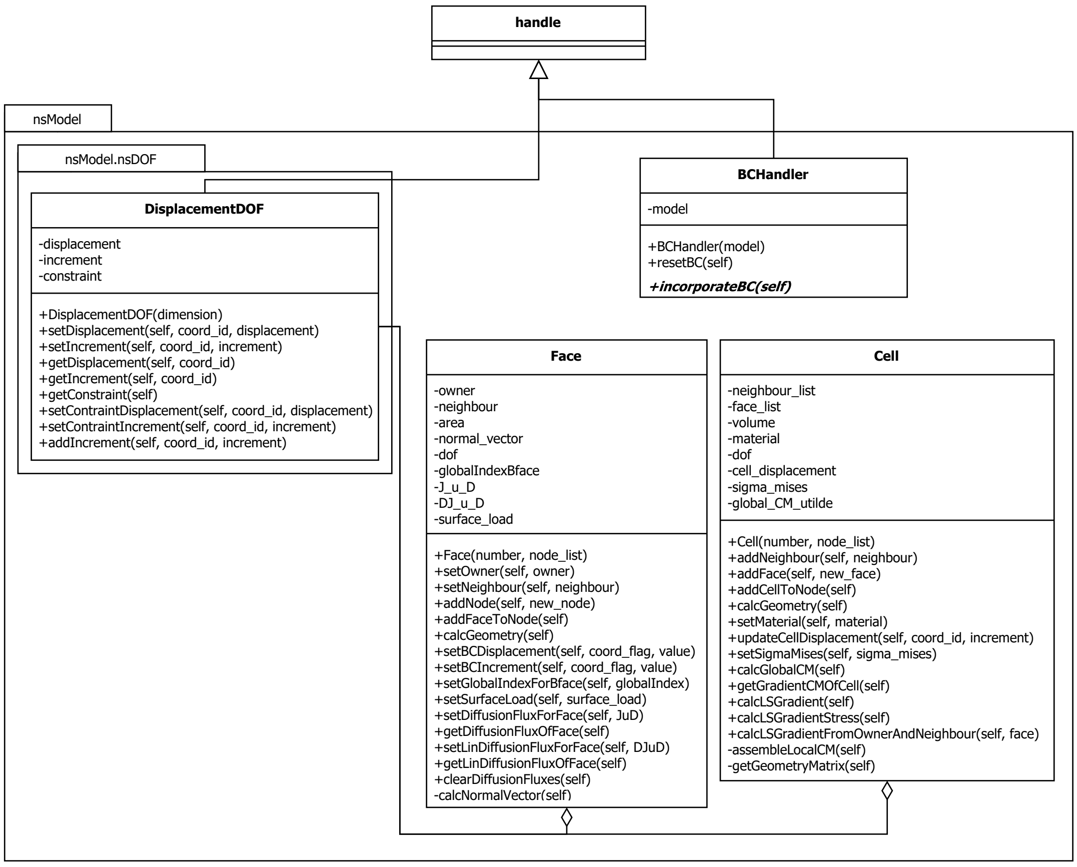

handle
nsModel
nsModel.nsDOF
DisplacementDOF
-displacement
-increment
-constraint
+DisplacementDOF(dimension)
+setDisplacement(self, coord_id, displacement)
+setIncrement(self, coord_id, increment)
+getDisplacement(self, coord_id)
+getIncrement(self, coord_id)
+getConstraint(self)
+setContraintDisplacement(self, coord_id, displacement)
+setContraintIncrement(self, coord_id, increment)
+addIncrement(self, coord_id, increment)
BCHandler
-model
+BCHandler(model)
+resetBC(self)
+incorporateBC(self)
Face
-owner
-neighbour
-area
-normal_vector
-dof
-globalIndexBface
-J_u_D
-DJ_u_D
-surface_load
+Face(number, node_list)
+setOwner(self, owner)
+setNeighbour(self, neighbour)
+addNode(self, new_node)
+addFaceToNode(self)
+calcGeometry(self)
+setBCDisplacement(self, coord_flag, value)
+setBCIncrement(self, coord_flag, value)
+setGlobalIndexForBface(self, globalIndex)
+setSurfaceLoad(self, surface_load)
+setDiffusionFluxForFace(self, JuD)
+getDiffusionFluxOfFace(self)
+setLinDiffusionFluxForFace(self, DJuD)
+getLinDiffusionFluxOfFace(self)
+clearDiffusionFluxes(self)
-calcNormalVector(self)
Cell
-neighbour_list
-face_list
-volume
-material
-dof
-cell_displacement
-sigma_mises
-global_CM_utilde
+Cell(number, node_list)
+addNeighbour(self, neighbour)
+addFace(self, new_face)
+addCellToNode(self)
+calcGeometry(self)
+setMaterial(self, material)
+updateCellDisplacement(self, coord_id, increment)
+setSigmaMises(self, sigma_mises)
+calcGlobalCM(self)
+getGradientCMOfCell(self)
+calcLSGradient(self)
+calcLSGradientStress(self)
+calcLSGradientFromOwnerAndNeighbour(self, face)
-assembleLocalCM(self)
-getGeometryMatrix(self)

Figure A.5.: nsModel.nsDOF

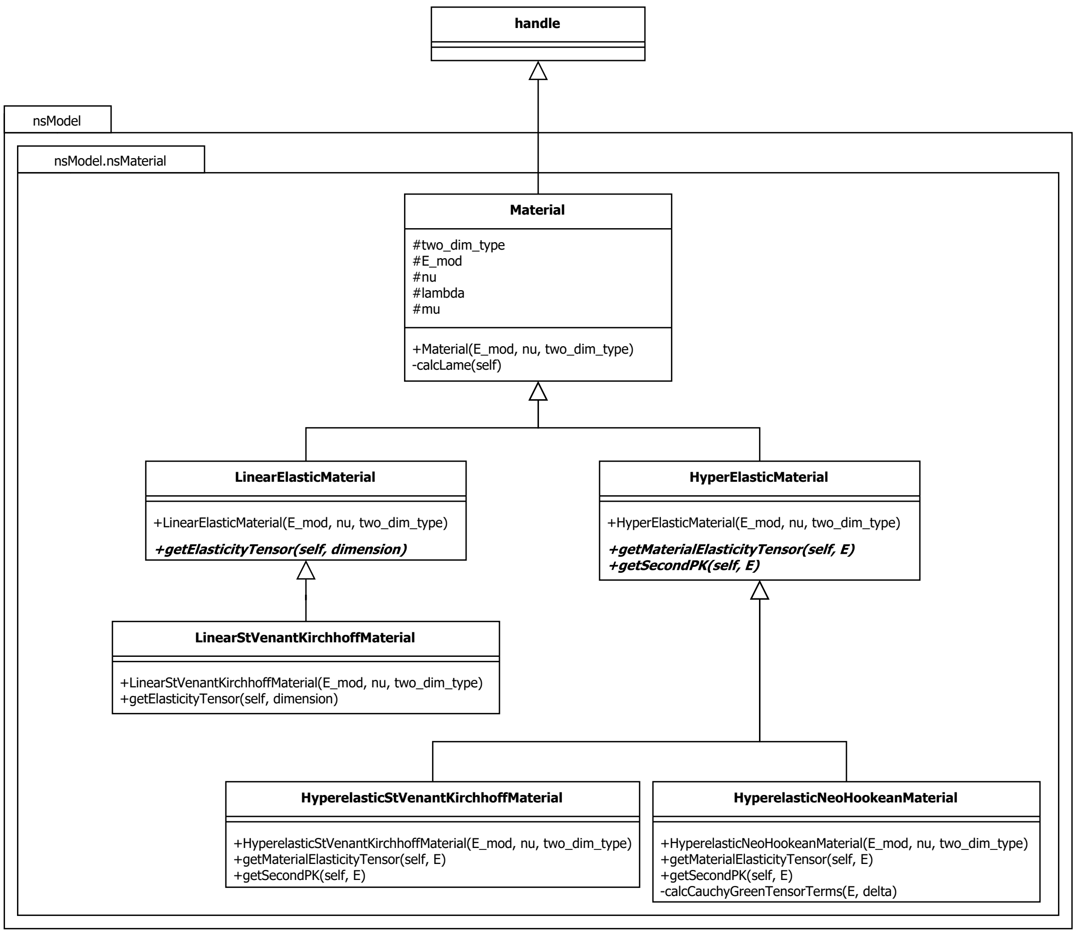

handle
nsModel
nsModel.nsMaterial
Material
#two_dim_type
#E_mod
#nu
#lambda
#mu
+Material(E_mod, nu, two_dim_type)
-calcLame(self)
LinearElasticMaterial
+LinearElasticMaterial(E_mod, nu, two_dim_type)
+getElasticityTensor(self, dimension)
HyperElasticMaterial
+HyperElasticMaterial(E_mod, nu, two_dim_type)
+getMaterialElasticityTensor(self, E)
+getSecondPK(self, E)
LinearStVenantKirchhoffMaterial
+LinearStVenantKirchhoffMaterial(E_mod, nu, two_dim_type)
+getElasticityTensor(self, dimension)
HyperelasticStVenantKirchhoffMaterial
+HyperelasticStVenantKirchhoffMaterial(E_mod, nu, two_dim_type)
+getMaterialElasticityTensor(self, E)
+getSecondPK(self, E)
HyperelasticNeoHookeanMaterial
+HyperelasticNeoHookeanMaterial(E_mod, nu, two_dim_type)
+getMaterialElasticityTensor(self, E)
+getSecondPK(self, E)
-calcCauchyGreenTensorTerms(E, delta)

Figure A.6.: nsModel.nsMaterial

Bibliography

[1] T. J. Barth, D. C. Jespersen, and Ames Research Center. *The Design and Application of Upwind Schemes on Unstructured Meshes*. AIAA-89/0366. American Institute of Aeronautics and Astronautics, 1989.

[2] I. Bijelonja, I. Demirdžić, and S. Muzaferija. A finite volume method for large strain analysis of incompressible hyperelastic materials. *International Journal for Numerical Methods in Engineering*, 64(12):1594–1609, 2005. ISSN 00295981. doi: 10.1002/nme.1413.

[3] I. Bijelonja, I. Demirdžić, and S. Muzaferija. A finite volume method for incompressible linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 195 (44-47):6378–6390, 2006. ISSN 00457825. doi: 10.1016/j.cma.2006.01.005.

[4] J. Blazek. *Computational fluid dynamics: Principles and applications*. Butterworth Heinemann, Amsterdam and San Diego, third edition edition, 2015. ISBN 9780080999951.

[5] J. Bonet and R. D. Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge University Press, Cambridge and New York, NY, 2nd ed. edition, 2008. ISBN 978-0-511-39468-3.

[6] B. H. G. Brady and E. T. Brown. *Rock mechanics for underground mining*. Springer, Dordrecht, 3. ed., repr. with corr edition, 2006. ISBN 9781402021169.

[7] P. Cardiff and I. Demirdžić. Thirty years of the finite volume method for solid mechanics, 2018. URL `https://www.researchgate.net/publication/328091509_Thirty_years_of_the_finite_volume_method_for_solid_mechanics`. (visited on 25/05/2020).

[8] P. Cardiff, Ž. Tuković, P. De Jaeger, M. Clancy, and A. Ivanković. A Lagrangian cell-centred finite volume method for metal forming simulation. *International Journal for Numerical Methods in Engineering*, 109(13):1777–1803, 2017. ISSN 00295981. doi: 10.1002/nme.5345.

[9] D. A. Caughey and A. Jameson. Basic Advances in the Finite-Volume Method for Transonic Potential-Flow Calculations. In Tuncer Cebeci, editor, *Numerical and Physical Aspects of Aerodynamic Flows*, pages 445–461. Springer Berlin

Heidelberg, Berlin, Heidelberg and s.l., 1982. ISBN 978-3-662-12612-7. doi: 10.1007/978-3-662-12610-3_26.

[10] M. A. A. Cavalcante and M.-J. Pindera. Generalized Finite-Volume Theory for Elastic Stress Analysis in Solid Mechanics—Part I: Framework. *Journal of Applied Mechanics*, 79(5), 2012. ISSN 0021-8936. doi: 10.1115/1.4006805.

[11] Community contributors. foam-extend 4.0, 2020. URL `https://sourceforge.net/projects/foam-extend/`. (visited on 25/05/2020).

[12] T. A. Davis. Algorithm 832. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):196–199, 2004. ISSN 0098-3500. doi: 10.1145/992200.992206.

[13] I. Demirdžić and S. Muzaferija. Numerical method for coupled fluid flow, heat transfer and stress analysis using unstructured moving meshes with cells of arbitrary topology. *Computer Methods in Applied Mechanics and Engineering*, 125(1-4):235–255, 1995. ISSN 00457825. doi: 10.1016/0045-7825(95)00800-G.

[14] E. N. Dvorkin and M. B. Goldschmit. *Nonlinear Continua.* Computational Fluid and Solid Mechanics. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 9783540249856. doi: 10.1007/3-540-29264-0.

[15] N. A. Fallah, C. Bailey, M. Cross, and G. A. Taylor. Comparison of finite element and finite volume methods application in geometrically nonlinear stress analysis. *Applied Mathematical Modelling*, 24(7):439–455, 2000. ISSN 0307-904X. doi: 10.1016/S0307-904X(99)00047-5.

[16] J. H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics.* Springer Berlin Heidelberg, Berlin, Heidelberg and s.l., third, rev. edition edition, 2002. ISBN 9783540420743. doi: 10.1007/978-3-642-56026-2.

[17] C. Geuzaine and J.-F. Remacle. Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, 2020. URL `https://gmsh.info/`. (visited on 25/05/2020).

[18] P. L. Gould and Y. Feng. *Introduction to Linear Elasticity.* Springer International Publishing, Cham, 4th ed. 2018 edition, 2018. ISBN 9783319738840. doi: 10.1007/978-3-319-73885-7.

[19] M. E. Gurtin. *An introduction to continuum mechanics*, volume 158 of *Mathematics in science and engineering.* Acad. Press, San Diego, Calif., transferred to digital printing edition, 2009. ISBN 9780123097507.

[20] J. Haider. *An upwind cell centred Finite Volume Method for large strain explicit solid dynamics in OpenFOAM.* Diss., Swansea University, Swansea, Wales, December 2017.

[21] C. Hirsch. *Numerical computation of internal and external flows: Introduction to the fundamentals of CFD.* Butterworth-Heinemann, Oxford, new ed. edition, 2007. ISBN 9780750665940.

[22] T. Hochrainer. *Elastizitätstheorie I.* Lecture notes, Graz University of Technology, Graz, 2019.

[23] G. A. Holzapfel. *Nonlinear solid mechanics: A continuum approach for engineering.* Wiley, Chichester, repr edition, 2010. ISBN 9780471823193.

[24] H. Jasak. *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows.* Dissertation, Imperial College of Science, Technology and Medicine, London, 1996.

[25] H. Jasak. OpenFOAM: Open source CFD in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, 1(2):89–94, 2009. ISSN 20926782. doi: 10.2478/IJNAOE-2013-0011.

[26] H. Jasak and H. G. Weller. Application of the finite volume method and unstructured meshes to linear elasticity. *International Journal for Numerical Methods in Engineering*, 48(2):267–287, 2000. ISSN 1097-0207. doi: 10.1002/(SICI) 1097-0207(20000520)48:2<267::AID-NME884>3.0.CO;2-Q.

[27] H. Jasak and H. G. Weller. OpenFOAM v1912: OpenCFD Ltd, 2020. URL `https://www.openfoam.com/`. (visited on 25/05/2020).

[28] P. K. Khosla and S. G. Rubin. A diagonally dominant second-order accurate implicit scheme. *Computers & Fluids*, 2(2):207–209, 1974. ISSN 00457930. doi: 10.1016/0045-7930(74)90014-0.

[29] G. Kirsch. Die Theorie der Elastizität und die Bedürfnisse der Festigkeitslehre. *Ver. Deutsch. Ing.*, 42, 1898.

[30] Kitware. ParaView, 2020. URL `https://www.paraview.org/`. (visited on 25/05/2020).

[31] A. C. Limache and S. R. Idelsohn. On the development of finite volume methods for computational solid mechanics. *Mecánica Computacional*, 26(11):827–843, 2007.

[32] L. E. Malvern. *Introduction to the mechanics of a continuous medium.* EPS, Prentice-Hall series in engineering of the physical sciences. Prentice-Hall, Englewood Cliffs, NJ, 2007. ISBN 9780134876030.

[33] K. Maneeratana. *Development of the Finite Volume Method for non-linear structural applications.* Dissertation, Imperial College of Science, Technology and Medicine, London, 2000.

[34] Mathworks. mldivide, \, 2020. URL `https://www.mathworks.com/help/matlab/ref/mldivide.html`. (visited on 25/05/2020).

[35] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab®*, volume 113 of *Fluid Mechanics and Its Applications*. Springer International Publishing, Cham, 1st ed. 2016 edition, 2016. ISBN 978-3-319-16873-9.

[36] MulPhys LLC. MULPHYS: Multi-Physics Solver For Continuum And Discrete Dynamics, 2020. URL `http://www.mulphys.com/mulphys/`. (visited on 25/05/2020).

[37] S. Muzaferija. *Adaptive Finite Volume Method for Flow Prediction using unstructured meshes and Multigrid Approach*. Dissertation, Imperial College of Science, Technology and Medicine, London, 1994.

[38] J. T. Oden and L. Demkowicz. *Applied functional analysis*. CRC Press, Boca Raton, Fla., 2. ed. edition, 2010. ISBN 9781420091953.

[39] E. Oñate, M. Cervera, and O. C. Zienkiewicz. A finite volume format for structural mechanics. *International Journal for Numerical Methods in Engineering*, 37(2): 181–201, 1994. ISSN 00295981. doi: 10.1002/nme.1620370202.

[40] M. Schäfer. *Computational Engineering - Introduction to Numerical Methods*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 9783540306856. doi: 10.1007/3-540-30686-2.

[41] A. K. Slone, C. Bailey, and M. Cross. Dynamic solid mechanics using finite volume methods. *Applied Mathematical Modelling*, 27(2):69–87, 2003. ISSN 0307-904X. doi: 10.1016/S0307-904X(02)00060-4.

[42] R. Suliman, O. F. Oxtoby, A. G. Malan, and S. Kok. An enhanced finite volume method to model 2D linear elastic structures. *Applied Mathematical Modelling*, 38 (7):2265–2279, 2014. ISSN 0307-904X. doi: 10.1016/j.apm.2013.10.028.

[43] G. A. Taylor, C. Bailey, and M. Cross. A vertex-based finite volume method applied to non-linear material problems in computational solid mechanics. *International Journal for Numerical Methods in Engineering*, 56(4):507–529, 2003. ISSN 00295981. doi: 10.1002/nme.574.

[44] S. P. Timošenko and J. N. Goodier. *Theory of elasticity*. Engineering societies monographs. McGraw-Hill, New York, 3. ed., mcgraw-hill classic textbook reissue edition, 1987. ISBN 0070647208.

[45] Z. Tukovic and H. Jasak. Updated Lagrangian finite volume solver for large deformation dynamic response of elastic body. *Transactions of FAMENA*, 31(1): 55–70, 2007.

[46] H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: The finite volume method*. Pearson/Prentice Hall, Harlow, 2005. ISBN 9780582218840.

[47] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in Physics*, 12(6):620, 1998. ISSN 08941866. doi: 10.1063/1.168744.

[48] Wikki Ltd. foam-extend: The community edition, 2020. URL `http://wikki.gridcore.se/foam-extend`. (visited on 25/05/2020).